UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Informáticos

Succinct Cryptographic Commitments with Fine-Grained Openings for Decentralized Environments

D O C T O R A L T H E S I S

Submitted for the degree of Doctor by:

Dimitrios Stylianos Kolonelos
M.Eng. in Electrical and Computer Engineering

Madrid, 2023


UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Informáticos

Doctoral Degree in Software, Systems and Computing

Succinct Cryptographic Commitments with Fine-Grained Openings for Decentralized Environments

D O C T O R A L T H E S I S

Submitted for the degree of Doctor by:

Dimitrios Stylianos Kolonelos
M.Eng. in Electrical and Computer Engineering

Under the supervision of:
Dr. Dario Fiore

Madrid, 2023

Title: Succinct Cryptographic Commitments with Fine-Grained Openings for Decentralized Environments

Author: Dimitrios Stylianos Kolonelos

Doctoral Programme: Doctorado en Software, Sistemas y Computación

Thesis Supervision:

Dr. Dario Fiore, Associate Research Professor, IMDEA Software Institute (Supervisor)

External Reviewers:

Thesis Defense Committee:

Thesis Defense Date:

To my family.

ACKNOWLEDGMENTS

This is probably the most difficult to write part of the thesis.

Undoubtedly many thanks to my advisor, Dario Fiore, for giving me the opportunity to pursue the PhD in such a wonderful place and the fully committed advising throughout these years. Your extreme and genuine kindness was the most important factor that made my, normally difficult and sentimentally daring, PhD procedure very smooth and pleasant. For your never-ending support and positivity. You were always enthusiastic for research, I had an amazing time during our (sometimes endless) research discussions. Your guidance was precious, especially in the periods when I was 'lost'. For all these reasons I am grateful, I hope all the advisors were as you!

I would like to thank my comittee members Dario Catalano, Pedro Moreno Sanchez, Charalambos Papamanthou, Carla Ràfols Salvador and Claudio Soriente, and the two substitute members Ignacio Cascudo and Maria Isabel Gonzalez Vasco. For accepting being in my committe, reading my thesis and going through the necessary paperwork. On top of that, thank you Dario Catalano and Babi Papamanthou for providing the external experts' evaluation of my thesis. Additionally, special thanks: Ignacio Cascuso and Pedro Moreno Sanchez for being in my predefense committee and providing that a prompt availability for it.

Thank you Mary Maller for giving me the opportunity to spend these four amazing months as a research intern at Ethereum Foundation, for your advising throughout these months and your genuine support overall these months and afterwards. I really enjoyed working with you and Misha Volkhov during these intense months (and later) on our challenging problem. My inherent stubborness deprived many hours of my sleep thinking about our problem and I am very happy that we finaly found a solution (despite the hereafter complications).

Thank you Sanjam Garg for hosting me in UC Berkeley, even if you didn't know me a-priori. I am grateful for the amazing and intense research summer I had with the crypto group of Berkeley, your insights on Cryptography and its impact are invaluable. I want to, also, deeply thank Arka Rai Choudhuri, Aarushi Goel, Guru-Vamsi Policharla, Sruthi Sekar and Mingyuan Wang for being extremely warm and welcoming with me as a newcomer and for having great discussions and fun.

I want to thank Giulio Malavolta, first for inviting me to MPI in Bochum, the most fruitful one-week visit that I could imagine. More importantly, for always being nice and supportive to

me without having to, just as an act of pure altruism. I am happy that people like you are in the community. I would also like to thank Hoeteck Wee for the great discussions and his advices. Your sharpness in research is beyond imagination and apart from that it is apparent that you are really interested in the continuance of the cryptographic community and in imparting to the new generation.

I would like to thank Esha Ghosh and Melissa Chase for hosting in Microsoft Research for a short stay. I really enjoyed discussing with you and all the bright cryptographers of MSR during these days. Furthermore, thank you Geoffroy Couteau for inviting me to IRIF in Paris where I spent an amazing week. Discussing about research with you for a week was wonderful to the point of overwhelming, I still don't understand how you handle all this information and at the same time you have awarness of things outside research. Special thanks to Paola de Perthuis and Christina Katsamaki for showing me around in Paris during these days and making me feel like home. Thank you Sherman Chow for inviting us together with David Balbas in Hong Kong to meet you and your group and for giving us the opportunity to see Hong Kong.

One of the most valuable experiences that I obtained from my PhD is working with other people in research. For this I would like to thank all my coauthors: Daniel Benarroch, Matteo Campanelli, Hien Chu, Dario Fiore, Nico Döttling, Sanjam Garg, Lydia Garms, Aarushi Goel, Nicola Greco, Kobi Gurkan, Noemi Glaeser, Semin Han, Jihye Kim, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, Mary Maller, Luca Nizzardo, Hyunok Oh, Paola de Perthuis, Guru-Vamsi Policharla, Ahmadreza Rahimi, Dominique Schröder, Rohit Sinha, Sina Shiehian, Claudio Soriente, Ida Tucker, Mikhail Volkhov, Mingyuan Wang and Hoeteck Wee. I frankly learnt something new from every single one of you and it seems that every individual has a (sometimes slightly) different perspective of cryptography.

I am deeply grateful to all IMDEA Cryptonians, for their kindness, the research discussions and the fun times that we had. The 'first generation': Miguel Ambrona, Matteo Campanelli, Antonio Faonio, Luca Nizzardo, Anaïs Querol, and the 'second generation': Hamza Abusalah, Gaspard Anthoine, Gennaro Avitabile, David Balbás, Claudia Bartoli, Diego Castejón, Peter Chvojka, Daniele Cozzo, Kasra Edalat, Lydia Garms, Emanuele Giunta, Miguel Morona, Chrysoula Oikonomou, Damien Robissout, István András Seres, Lucas Tabary, Ida Tucker and Dimitrios Vasilopoulos. Special thanks to Miguel Ambrona, Gaspard Anthoine, Anais Querol for being wonderful officemates. Special guest (literally): Arantxa Zapico with whom we shared the office for three vivid weeks in which maybe I worked less than I wanted to but had a great time. On top of that, Anais many thanks for always being extremely helpful with all the spanish/university paperwork, I sincerely wouldn't have survived without your help.

I would also like to thank all the IMDEA people that we have shared moments together, that I decided not to list because the list would be endless! The fact that IMDEA is a friendly and healthy environment was extremely crucial for me during all these years, and this is clearly its people's spawn. Especially for the thesis submission procedure I'd like to thank Silvia Sebastian for sharing her prior recent experience with us and being extremely helpful and the compañera Kyveli Doveri for going through all the stressful procedure together.

I want to especially thank Platon Kotzias and Luca Nizzardo for sharing me their opinion on whether to chase the PhD or not, at a time when I was unsure. Giving me your greatest feedback undoubtedly played a significant role for me to take the decision to eventually do it, a decision that now I am extremely happy to have taken.

For the end I left the, perharps, most important people, the ones that had nothing to do with my research or ever understood anything about it but were able to understand much more

significant things. My 'family' in Madrid: Alexandra, Christos, Dionisis, Elissavet, Kostis, Kyveli, Lydia; words are meaningless here... Without them I wouldn't have survived Madrid all these years and certainly I wouldn't be writing a thesis now. Sergio, Pablo and Carmela, and later Pablo and Anita: for being wonderful roomates and friends that made our home feel like family. My friends from Athens, that I will not enumerate because the list would be lenghty, for always being there and supporting me in every situation. And, of course, my family Roula, Stathis and Annoula for everything.

ABSTRACT OF THE DISSERTATION

Cryptography has always been the science of Secure Communication. The past decade has been epitomized by the emergence of Blockchain Technology, without leaving Cryptography unaffected. The manifesto of Blockchains is Decentralization, making it inevitable that the information is stored and verified in real-time by thousands of participants. This spotlighted two necessities: information should be as concise as possible and verification of processes should be fast.

From a cryptographic perspective, this translates to a central desideratum: Succinctness. A cryptographic construction is called Succinct if its algorithm is generating outputs that are (exponentially) smaller than the inputs. This allows the cryptosystem to treat large data and produce concise outputs that, nevertheless, preserve the desired functionality of the system.

In this thesis, we are concerned with a specific type of Succinct cryptographic primitives: Succinct Commitments. Cryptographic commitments are objects that allow one to commit to some data, providing a binding representative. Then at any later point they can open back the (committed) data providing an opening proof, but without being able to open differently the representative. In more detail, in our work we deal with commitments with more fine-grained openings, where one can generate an opening proof of the commitment to a function $f(m)$ of the initial data m.

Firstly, we deal with set commitments with private (non-)memberhsip openings. We construct succinct zero-knowledge proofs for the problem of set (non-)membership. Intuitively, a zero-knowledge proof is a cryptographic primitive that allows one to prove a statement, in a sound way, without leaking any other information except for the fact that the statement holds. In a zero-knowledge proof for set membership first one commits to a public set and then a party can prove membership to the set but without betraying which element of the set exactly is. Such (set) commitments with this type of fine-grained openings are the cornerstone of Anonymous Cryptocurrencies such as Zcash. In particular we provide efficient zero-knowledge proofs for the opening of RSA accumulators, one of the most popular set commitments. First, we show efficient protocols for membership and non-membership of a single element. Then we construct succinct zero-knowledge membership proofs for multiple elements, where the size of the proof is independent of the number of elements proven. The two techniques are qualitatively different.

Secondly, we switch our attention to Vector Commitments, with local positional openings.

We put forth the notion of Incremental Aggregation, in which one can arbitrarily aggregate opening proofs of any positions into a single (concise) proof and inversely disaggregate a proof of multiple points to many. We show applications of this notion (1) to speeding up the proof computation by using precomputation and moderate-sized precomputed values and (2) to Verifiable Decentralized Storage. Finally, we provide efficient construction of Incrementally Aggregatable Vector Commitments from Groups of Unknown Order.

Thirdly, we turn to Functional Commitments, for linear functions, where one commits to a vector $\boldsymbol{v}$ and then can open $f(\boldsymbol{v}) = y$, for a public f. We construct functional commitments that admit constant-sized public parameters and proofs. To this end, our core technique is a novel succinct protocol of cardinality for a set committed with an RSA accumulator, which is in turn based on a Range Proof.

Finally, we show a generic way to turn any Vector Commitment into a Key-Value Map Commitment for arbitrary keys. A Key-Value Map resembles a Vector but the ordering of the values is not characterized by subsequent indices but by arbitrary keys. Key-Value Maps are the core data-structures in Cryptocurrencies like Ethereum. Our construction of Key-Value Map Commitments is generic and is based on a novel cryptographic application of Cuckoo-Hashing.

RESUMEN DE LA TESIS DOCTORAL

La última década se ha caracterizado por la aparición de la tecnología Blockchain, afectando la criptografía. El manifiesto de Blockchains es la Descentralización, en la que la información es guardada y verificada en tiempo real por miles de participantes. Esto centra la atención en dos necesidades: la información debe ser lo más concisa posible y la verificación de los procesos debe ser rápida. Desde una perspectiva criptográfica, esto se traduce en un desiderátum central: la Compacidad.

En esta tesis, nos ocupamos de un tipo específico de primitivas criptográficas compactas: Compromisos Compactos. Los compromisos criptográficos son objetos que permiten comprometerse con algunos datos, proporcionando un representante vinculante, de modo que en cualquier momento posterior se pueden volver a abrir, proporcionando una prueba de apertura. En nuestro trabajo tratamos compromisos con aperturas más detalladas, donde se puede generar una prueba de apertura del compromiso con una función $f(m)$ de los datos iniciales m.

En primer lugar, nos ocupamos de compromisos de conjuntos con aperturas privadas de (no) pertenencia. Construimos pruebas compactas de conocimiento cero para el problema de la (no) pertenencia a conjuntos. Una prueba de conocimiento cero es una primitiva criptográfica que permite probar una afirmación, de forma sólida, sin filtrar ninguna otra información excepto el hecho de que la afirmación es cierta. En una prueba de conocimiento cero para la pertenencia a un conjunto, primero uno se compromete con un conjunto público y luego una parte puede demostrar la pertenencia al conjunto, pero sin revelar qué elemento del conjunto es exactamente. Estos compromisos de conjuntos con este tipo de aperturas detalladas son la piedra angular de las criptomonedas anónimas como Zcash. En particular, proporcionamos pruebas eficientes de conocimiento cero para la apertura de acumuladores RSA, uno de los compromisos establecidos más populares. Primero, mostramos protocolos eficientes para la membresía y no membresía de un solo elemento. Luego construimos pruebas compactas de membresía de conocimiento cero para múltiples elementos, donde el tamaño de la prueba es independiente del número de elementos probados. Las dos técnicas son cualitativamente diferentes.

En segundo lugar, centramos nuestra atención en los compromisos de vectores, con aperturas posicionales locales. Presentamos la noción de Agregación Incremental, en la que se pueden agregar arbitrariamente pruebas de apertura de cualquier posición en una prueba única (concisa) e inversamente desagregar una prueba de múltiples puntos en muchos. Mostramos aplicaciones

de esta noción (1) para acelerar el cálculo de la prueba mediante el uso de precómputo y valores precalculados de tamaño moderado y (2) para el Almacenamiento Descentralizado Verificable. Finalmente, proporcionamos una construcción eficiente de compromisos vectoriales incrementalmente agregables a partir de grupos de orden desconocido.

En tercer lugar, pasamos a los compromisos funcionales, para funciones lineales, donde uno se compromete con un vector $\boldsymbol{v}$ y luego puede abrir $f(\boldsymbol{v}) = y$, para un f público. Construimos compromisos funcionales que admiten pruebas y parámetros públicos de tamaño constante. Con este fin, nuestra técnica principal es un protocolo novedoso y compacto de cardinalidad para un conjunto comprometido con un acumulador RSA, que a su vez se basa en una prueba de rango.

Finalmente, mostramos una forma genérica de convertir cualquier compromiso de vector en un compromiso de mapa-de-valores-clave para claves arbitrarias. Un mapa-de-valores-clave se parece a un vector, pero el orden de los valores no se caracteriza por índices posteriores sino por claves arbitrarias. Los mapas-de-valores-clave son las estructuras de datos centrales en criptomonedas como Ethereum. Nuestra construcción de compromisos de mapas de valores clave es genérica y se basa en una novedosa aplicación criptográfica de Cuckoo-Hashing.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PUBLICATIONS

This thesis comprises 6 papers, of which five papers have been published in peer-reviewed academic conferences and one paper which has been published in a peer-reviewed Journal, which is an extended version of the corresponding conference paper:

- **Incrementally Aggregatable Vector Commitments and Applications to Verifiable Decentralized Storage**
  Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, Luca Nizzardo
  In proceedings of Advances in Cryptology – ASIACRYPT 2020
- **Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular**
  Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, Dimitris Kolonelos
  In proceedings of Financial Cryptography and Data Security 2021
- **Inner Product Functional Commitments with Constant-Size Public Parameters and Openings**
  Hien Chu, Dario Fiore, Dimitris Kolonelos, Dominique Schröder
  In proceedings of Security and Cryptography for Networks 2022
- **Succinct Zero-Knowledge Batch Proofs for RSA Accumulators**
  Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, Hyunok Oh
  In proceedings of 2022 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS 2022)
- **Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular**
  Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, Dimitris Kolonelos
  In Designs, Codes and Cryptography
- **Cuckoo Commitments: Registration-Based Encryption and Key-Value Map Commitments for Large Spaces**
  Dario Fiore, Dimitris Kolonelos, Paola de Perthuis
  In proceedings of Advances in Cryptology – ASIACRYPT 2023

Other papers co-authored during the term of my PhD grant, but not included in this dissertation:

- **Ring Signatures with User-Controlled Linkability**
  Dario Fiore, Lydia Garms, Dimitris Kolonelos, Claudio Soriente, Ida Tucker
  In proceedings of Computer Security – ESORICS 2022
- **Zero-Knowledge Arguments for Subverted RSA Groups**
  Dimitris Kolonelos, Mary Maller, Mikhail Volkhov
  In proceedings of Public-Key Cryptography – PKC 2023
- **Efficient Laconic Cryptography from Learning With Errors**
  Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, Ahmadreza Rahimi
  In proceedings of Advances in Cryptology – EUROCRYPT 2023
- **Efficient Registration-Based Encryption**
  Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, Ahmadreza Rahimi
  To appear in proceedings of 2023 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS 2023)
- **Distributed Broadcast Encryption from Bilinear Groups**
  Dimitris Kolonelos, Giulio Malavolta, Hoeteck Wee
  To appear in proceedings of Advances in Cryptology – ASIACRYPT 2023
- **harpocraTEs: Threshold Encryption with Silent Setup**
  Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, Mingyuan Wang
  Under Submission

Part I

INTRODUCTION AND PRIOR WORK

1 INTRODUCTION

1.1 Cryptography

This thesis could not begin in any alternative way than contextualizing Cryptography.

Historical remarks. Historically, Cryptography has been the art of secure communication. For many centuries, its central purpose was hiding the content of written messages, by producing a 'code'–that we call a 'cipher'–corresponding to the actual content of the message. The goal was that individuals could not make sense of the cipher, even if carefully reading it, unless knowing a method to de-cipher back to the message of origin.

Examples of ciphers that have been documented are numerous, starting from the early 1900 BC in ancient Egypt, where signs of ciphers have been found carved in tombs. In ancient Greece, the spartan military was using a special tool called 'scytale' to cipher and de-cipher messages. Julius Caesar has also also been documented to use the 'Caesar cipher' to securely communicate with his army generals. More recent specimens are, among others, Vigenère cipher and the famous Enigma machine used by the German military during World War II. Most of the early ciphers were using a simple substitution; each symbol was consistently substituted by another one at each appearance in the message. From the 15th century ciphering started getting more evolved diverging from the simplistic substitution-ciphering, but nevertheless didn't drastically change, the main principles remained the same: Concealing of the actual message from the ciphertext was anticipated based on (1) the heuristic belief that it is difficult for a human to revert a superficially intricate cipher and, additionally, (2) the general procedure of the code is secret.

Regarding the latter, in 1883 Auguste Kerckhoffs put forward the position that the security of *any* crypotsystem should not rely on the obscurity of the method [138]. Everything about the intelectual methodology and the material used by the cryptosystem should be considered public knowledge, except for some ephemeral variables of the system: The 'keys'. The logic behind this argument is that the design of a cryptosystem that is continuously used cannot remain secret and will eventually be leaked. Claude Shannon graphically re-produced this concept with the phrase "the enemy knows the system". Therefore, when designing a cryptosystem we should assume that our knowledge on the system is public up to the slightest detail. This position was and still is greatly embraced, forming the 'Kerckhoffs' principle' and is in contrary to the 'Security through obscurity' approach. Kerckhoffs' principle is a standard approach in today's Cryptography and henceforth, for this thesis, will be taken for granted.

The critical point that revolutionized Cryptography was the emergence of modern computing machines, during the 20th century. It was rapidly realized that advanced computers can break cryptosystems that were unthinkable to be broken by humans, or at least not in real-time. The manifestation of this (although not the first example of cryptanalysis-by-devices historically) was the celebrated use of the Bombe machine, designed by Alan Turing, to de-cipher the messages of the Enigma device.[1] The contribution of Alan turing is an early sign of modern Cryptography and, more generally, Computer Science. With computers available, the target security of cryptosystems is no longer against adversarial humans but against powerful computers. Gradually in the 20th century Cryptography started getting irreversibly bound to Computer Science.

The science of Cryptography. So computers are the ones that converted Cryptography from 'art' to 'science'. Now, Cryptography should follow a systematic process in order to achieve convincing results. Fundamental to Modern Cryptography is the paradigm of 'Provable Security' that consists of the following three:

- **Definition:** First we need to rigorously define what we expect from the cryptosystem. What does it even mean that a system is 'secure'?
- **Assumptions:** Then we need to state clearly what are the assumptions under which we are working. These can be seen as the equivalent of the mathematical 'axioms'.
- **Proof of Security:** Finally, we need to give a mathematical proof that our cryptosystem satisfies the aforementioned definition under the aforementioned assumptions.

A Definition should depict–with mathematical formalization–how the cryptosystem is abstractly working and, more importantly, a mathematical specification of what its 'security' means. It should also specify the "power" of the adversary against which we anticipate the system to be secure. Typically in today's standards we want our systems to be secure (at least) against adversaries that have access to the most powerful existing computer.[2] An example security definition could be: "For a ciphering system X, an adversary controling all the computational power on earth cannot retrieve more than 2% of the bits of the corresponding message, for any possible message". The provable security paradigm is not concerned with whether the security specifications of a definition are suitable enough or not; this is up to human interpretation. Provable security is a systematic method to claim that the definition is satisfied.

The Assumptions that we may take for a cryptosystem can largely vary. In many cases they are related to Computational Complexity Theory. For example one could put forth the assumption "there is no algorithm running in polynomial time that can solve the boolean satisfiability problem". This assumption is in turn connected to the Complexity Theory conjecture that $P \neq NP$ therefore we have some reassurance. Sometimes an assumption can be connected to the current inability of breaking it, for instance the assumption "factoring a composite integer cannot be done in polynomial time" can be justified by the fact that this problem has been widely studied for decades and we do not currently have a solution. Similarly to the Definition, whether

[1] The cryptosystem of Enigma was actually intelectually "broken" by the Polish mathematician Marian Rejewski in 1932. Alan Turing designed the computing machine that could efficiently perform de-ciphering in real-time.

[2] 'Fine-grained Cryptography' [161] is a field of Cryptography that diverges from that assuming less powerful adversaries. In this thesis we are not concerned with this model.

an assumption is plausible or not is orthogonal to the provable security paradigm, this can largely be a subject to human interpretation and, nevertheless, subjective.

The Security Proof is the part of provable security that should be undeniable, being unaffected by human interpretation. It should follow a clasic mathematical reasoning and in fact security proofs can be seen as classic mathematical proofs.

Throughout this thesis we will follow this methodology.

Basic Concepts. Although, as we discussed above, for centuries the primary focus of cryptography was predominantly centered on the creation of codes to conceal messages, the past century it tremendously expanded. In a very general sense we can say that its main goal it still Secure Communication but the significance of both "Secure" and "Communication" can widely vary. There are endless goals, concepts and classifications in Cryptography that we could discuss. Below we, very briefly, survey some central notions.

- **Symmetric-Key Encryption:** Firstly by 'Encryption' we refer to the process of tranforming a message (plaintext) to a cipher (ciphertext) in order to conceal the information of the message. Conversely by 'Decryption' we refer to the inverse process, of retrieving the message from its corresponding ciphertext.

  Symmetric-Key Encryption is a method to Encrypt a message so that it can later be Decrypted using the same key, but cannot get Decrypted otherwise. The distinguishing trait of Symmetric-Key Encryption is that both Encryption and Decryption use the same key. Symmetric-Key Encryption is secure if no computer can retrieve the message without knowing the key. It is crucial then that the key stays secret otherwise security is trivially compromised.

  For example the early ciphers in history discussed above can be seen as Symmetric-Key Encryption schemes, though insecure under the above definition.

- **Public-Key Encryption:** Public-Key Encryption [93, 184] is the asymmetric version of Symmetric-Key Encryption. Anyone can, without interaction, Encrypt a message under a public key that is known to everyone and produce a valid ciphertext. Then only the party knowing the corresponding secret key can decrypt the cipertext to retrieve the message. Of course, in order for this to be able to work, the public and secret keys should be somehow correlated.

  The security definition is the same as in Symmetric-Key Encryption except the public key can be known to the potential adversary.

- **Digital Signatures:** Digital signatures [184] is the analogue to physical signatures. One party–the Signer–can authenticate/'sign' a message using their secret signing key. This signing key is the analogue to the unique handwriting of physical signatures. Then anyone can verify that a signature is bound to that specific message by using a publicly known verification key corresponding to the secret key of the signer. Again, the signing (secret) and verification (public) keys are correlated.

  The Digital Signature is secure if no party can forge a valid–under a verification key–signature on any message without possessing the corresponding signing key.

- **Commitments:** Imagine that we want to make a claim but we would only like to reveal it after a specific event happens, for instance after 5 hours or after the next rainy day, but in the

meantime we wish to keep it secret. A physical way to implement it would be to write our claim in a paper, close it in a locked safe box and open the box only when the desired event happens.

A cryptographic Commitment scheme [38, 46] captures this scenario. It is a technique to 'commit' to an element by generating a representative, the 'commitment', so that at any later point we can 'open' it. By 'open' we mean that that we can demostrate that the representative 'commitment' is for the initial element committed.

The natural security properties that should hold for a commitment scheme are Binding and Hiding. Hiding captures that the commitment should not betray our element, before we choose to open it. Binding captures that after committing we are bound to open the original element and nobody can open it in any different way.

- **Zero-Knowledge Proofs:** A Zero-Knowledge Proof [122] is a cryptographic scheme that allows one party (the Prover) to prove a statement to some other party (the verifier) without leaking *any* other information except for the fact proven. The prover and the verifier may interact with each other some times before the verifier gets convinced.

  The security properties that are required are Soundness and Zero-Knowledge. Soundness is a property that protects the verifier from malicious provers: No prover–using whatever strategy–should be able to convince the verifier for an invalid statement. Zero-Knowledge, on the other hand, protects the prover: Even if the verifier is maliciously behaving they should not be able to learn anything else from their participation in the zero-knowledge protocol except for the fact that the proven statement holds.

- **Secure Multi-Party Computation:** Secure Multi-party Computation [217, 121] is the field of Cryptography that designs protocols so that multiple parties can securely compute a function. For example assume that party 1 has input x_1, party 2 has input x_2, up to party n having input x_n. Then these parties want to collaboratively compute a public function f with these inputs, i.e. to learn $y = f(x_1, \ldots, x_n)$. However, they do not desire to leak their inputs to the other parties. This (with different variations) is precisely the objective of Multi-party Computation.

- **Cryptanalysis:** Cryptanalysis is the field of Cryptography that analyzes cryptosystems in order to find flaws, primarily in the security of the system. Cryptanalysis of provably secure cryptographic schemes is typically in the assumptions taken, but can even be in the security proof. Another type of Cryptanalysis has to do with finding practical attacks in cryptosystems that are only heuristically (instead of provably) secure.

- **Foundations of Cryptography:** Foundations of Cryptography refers to the effort of identifying the (typically minimal) assumptions from which we can build Cryptography.

- **Advanced Cryptographic Primitives:** There are numerous cryptographic primitives providing advanced functionalities that do not fall in any of the above categories. Examples include Pseudorandom Functions [120], Anonymous Credentials [70], Fully Homomorphic Encryption [116], Functional Encryption [43] (e.g. Identity-based Encryption [193, 41], Attribute-based Encryption [186, 124]) and Indistinguishability Obfuscation [133] and many more.

Looking ahead, this thesis predominantly falls within the categories of Commitments and Zero-Knowledge Proofs. Furthermore, from now on when referring to "Cryptography" we will be meaning "Constructive Cryptography", otherwise we will crearly state "Cryptanalysis".

Quantum Computers and Cryptography. Cryptography flourished in the 20th century, as earlier described, due to the emergence of modern digital computers. Its development was, in consequence, built upon the computational model of digital computers. Both the capabilities of the cryptosystem and the adversary were modeled according to the classical binary computer model. In 1980 Paul Benioff proposed an alternative computational model based on quantum physics [34], the theoretical core of what we call today 'Quantum Computer'. In 1993 Peter Shor showed that in the quantum computation model there are efficient algorithms that can break fundamental problems in Cryptography that were assumed to be hard: the Factoring and Discrete-logarithm problems [195]. In essence, this directly showed that if quantum computers are physically built then a great part of cryptosystems known at the time (and even today a large fraction of them) would be broken.

Full-fledged Quantum Computers with reasonable–to break cryptographic schemes–computing power have not been produced to date. The challenge is in constructing the actual hardware for quantum computation. However, there are very intense efforts in developing them.

This could not leave Cryptography unaffected. The 'Quantum Threat' gave birth to two different developments in Cryptography: The 'Post-Quantum Cryptography' and the 'Quantum Cryptography'. Post-Quantum Cryptography is concerned with protecting classical cryptosystems against Quantum Computers. In that the model of the computation for the cryptosystem is still the one of binary electronic computers but the adversary of the cryptosystem is assumed to be a Quantum Algorithm (using a quantum computer). For example Post-Quantum Cryptography is based on problems that are (plausibly) secure against quantum algorithms, for example Lattice-based problems [4, 131, 175] or Isogeny-based problems [82, 185, 65, 90]. On the other hand in Quantum Cryptography everything works on quantum computers, both the cryptosystem and the adversary.

We note that most of the results of this thesis fall in neither of these two categories. Our results are in the 'Classical Cryptography' setting.

1.2 Desiderata of modern Cryptography: Succinctness and more

Today's Cryptography is all about efficiency.

This provocative opening statement is of course not entirely true, but graphically captures the recent trends in Cryptography. After decades of research it's safe to say that in terms of *functionality* most aspirations of Cryptography have been either met or proven impossible to be ever met. The examples of phenomenal success stories are endless, we have Symmetric-Key Encrpytion, Public-Key Encryption, Digital Signatures, Zero-Knowledge proofs for all NP, Secure Multi-party computation for all functions. The list can be enriched with much more advanced cryptographic primitives that were notoriously hard to be obtained like Identity-based Encryption [41], Fully Homomorphic Encryption [116] or even, recently, Indistinguishability Obfuscation [133].

The intriguing research questions in today's Classical Cryptography are to a great extent not of the form *"Can we we build X?"* but rather: *"Can we we build X with Y efficiency properties?"* or *"We know we can build X from Y assumptions, can we build it from Z assumptions?"*. The first category of questions is about *Efficiency* while the second is about *Assumptions*. These are very crucial concepts in recent developments in Cryptography, and in this work most of the questions that we answer are of these types.

By 'Efficiency' we refer to improving some metrics of the cryptosystem that are already in the feasibility (polynomial-time) realm. These metrics may include: the bit-size of some variable of the system, the running time of an algorithm, the number of rounds that some parties have to interact, etc. The Efficiency improvement can be either asymptotic (e.g. from $O(n^2)$ runnning time to $O(n \log n)$) or even concrete (e.g. ciphertext size from 10KB to 9KB or from 3 communication rounds to 2).

Before proceeding, it's important to acknowledge the existence of other significant research avenues beyond Efficiency and Assumptions, outside Clasical Cryptography. In Quantum Cryptography there are various unexplored directions about what can be achieved. Unlike Classical Cryptography, there it cannot be claimed that the field has reached the same level of maturity to have answers to most of the fundamental questions. Another field that has different objectives is the one of Cryptanalysis, where the efforts are focused on breaking assumptions, cryptosystems or implementations. Finally, we should note that there is the field of Secure Cryptographic Implementations that also has different objectives.

Succinctness. Succinctness (or conciseness) is a special Efficiency-property that is on the core of numerous cryptographic schemes. But what is Succinctness? By 'Succinctness' we typically refer to an (at least) exponential improvement[3] on the bit-size of a value of the cryptosystem. Assume that a cryptosystem A takes n inputs $x_1, \ldots, x_n$ and outputs a value y of size $|y| = O(n)$. We say that succinctness is achieved in cryptosystem B if it has the same functionality but with an output of size $O(\log n)$ or $O(1)$.

In general, the power of Succinct Cryptographic primitives is that they can process very large data producing at the end very condensed outcomes. This is of extreme importance since typically these outcomes are the elements that are communicated. As we will see in the next sections, in many occasions it is prohibitive to communicate all the data processed.

Succinctness is a highly non-trivial property to achieve and in most cases it requires a radical change in the approach. For some important fields of modern Cryptography, the entire field would not even exist if Succinctness was not a prerequisite.

Take for example the case of Verifiable Computation. There there two distinct parties with unbalanced computational power, one performing over a weak machine (e.g. a mobile phone) and another on a powerful one (e.g. on a cluster of computers). Then the computational weak party (W) wants to outsource an intensive comutation to the strong one (S). However, W wants to make sure that the computation was performed correctly by S. This is exactly the concept of Verifiable Computation.

Now, Cryptography comes in to ensure integrity: That the computation was performed correctly. Apparently, by definition of the problem, S cannot send the steps of the computation to W. So Cryptography should provide a Succinct solution to the problem: Integrity should be ensured in a way that the certificates "ensuring" it are much smaller than the size of the computation itself. Of course, without Succinctness a Verifiable Computation primitive makes no sense.

There are many more interesting scenarios in which we have unbalanced computational power between communicating parties like the above. In what follows we discuss more examples.

In this thesis Succinctness is a central notion. Almost all of the problems we deal with in this

[3] In the literature sometimes 'Succinctness' is also used for polynomial improvements, e.g. quadratic, cubic, etc. In this work we always use the term for exponential improvements.

work could be solved with basic results of Cryptography, if we didn't require the cryptosystems to be succinct.

1.3 The era of Decentralization

Enter Satoshi Nakamoto.

In October 2008, someone with the pseudonym 'Satoshi Nakamoto' published a whitepaper with the title "Bitcoin: A peer-to-peer electronic cash system" [165]. The whitepaper was introducing a digital currency with the name "Bitcoin". This was meant to be one of the most groundbreaking inventions of the 21st century.

Bitcoin is a decentralized digital currency. To begin with, think about the digital banking payment system; this can already be regarded as digital currency. However, apparently it is centralized, there is a clear central party that controls pretty much everything: the participants of the system, the transactions that occur and in some extreme cases even the balances of the participants. The main principles that accompany 'decentralization' are the extreme opposite, no single authority can control the system. Every single step that the system performs should be approved by "everyone" (or at least by the majority) participating in the system.[4] In Bitcoin, and in any decentralized digital currency, anyone can openly participate and none can prevent a transaction from happening (as long as it follows some elementary rules of validity).

Abstractly, a digital currency can be fully determined by a book that contains *all* the information about its variables: Some initial state of balances for each initial participant and afterwards the full history of all transactions. We call this book 'the ledger'. In centralized banking systems the ledger is a database that the bank has exclusive rights to. In contrast, in Bitcoin the ledger should be public and maintained collaboratively by the participants of the system. So on the core of Bitcoin is this public ledger which is called 'the Blockchain'. Bitcoin provides a mechanism to maintain consistently and securely 'the Blockchain', which of course should contain valid transactions.

After Bitcoin's deployment, at the beginning hesitantly but later massively, many other similar (or even identical) concepts arose. This generated the so-called 'Cryptocurrencies' or, more broadly, 'Blockchain Technology'. For instance, the first important milestone that evolved Bitcoin and Cryptocurrencies was Ethereum [54] that introduced the concept of 'Smart Contracts' (somehow analogously to physical contracts). Today, we can say that the Blockchain Technology has moved far beyond Bitcoin and Cryptocurrencies. We just mention a few examples of decentralized systems: Smart Contracts [54], Distributed Storage Networks [144], Decentralized Identity (DID) systems [205, 206, 204], Decentralized Autonomous Organization [130] and much more.

But how are Decentralized Systems related to Cryptography, and more generally to Computer Science? Roughly speaking there are primarly three fields of Computer Science that play an important role in Blockchain Technology: Cryptography, Distributed Systems and Game Theory. Cryptography's role is twofold, first to ensure the validity of the operations (e.g. transactions) and second to ensure the overall consistency of the "pages" of the ledger, in other words to "bind" all the states of the ledger. The role of Distributed Systems is to provide a consensus mechanism to the participants. That is, a way so that all the distributed users agree on the same

[4] This is a somewhat incorrect simplification of how Bitcoin works, used for the sake of a gentle introduction.

public ledger. Finally, Blockchain heavily realies on economic incentive mechanisms that Game Theory can analyze (or sometimes design).

These three fields and in particular Cryptography, the domain of this thesis, have been tremendously impacted the last decade (if not more) from the rise of Blockchain technology and Decentralized systems. This impact is also present is this thesis. Although our results are cryptographic, the scope and the applications of the results are influenced from the Blockchain world. One way to think about it is that the emergence of Blockchain gave a different significance to some of the cryptographic results.

In particular, a concrete aspect of blockchain technology that is closely relevant to our work is the 'State' of the blockchain. As overviewed above the blockchain is the ledger that contains all the history of the system, for example in cryptocurrencies all the transactions that have occured. But then for the continuous operation of the system there is a more specific information that is necessary, that can be extrapolated from the history. This is the state. For example in (some type of) cryptocurrencies the state is the balances of all the users. The set of balances allows us to say if a transaction is valid (if the balance is larger than the amount spent) or not. Of course, one could just go through the whole history of the blockchain to verify a single transaction but that would be an overkill. A similar concept of 'state' exists in pretty much all the decentralized systems. In conclusion, we could say that the state of the blockchain is the information needed to verify the processes of the system.

So the pattern is "given a state S I want to verify that X holds". Although this is something that can be naively checked if having S and X, it is not ideal in many cases. For example the state S itself is large and grows with the popularity of the Blockchain, this is a 'Scalability' problem. In other Decentralized Systems some part of the argument X should be kept private, this is a 'Privacy' problem. Cryptography comes into play in this pattern, trying to resolve both Scalability and Privacy. Succinctnes is once again crucial: we want to somehow treat the state S in a way that the cryptographic outputs are much smaller than its size.

We will concretize this is the next sections.

1.4 Succinct Commitments and Fine-Grained Openings

Warm-up.

Here we give a brief overview of the different notions of Succinct Commitments with Fine-Grained Openings. We elaborate more on the actual constructions and the rich literature in Chapter 2.

Commitment schemes are one of the most fundamental cryptographic primitives. They can be seen as the digital equivalent of a sealed envelop: committing to a message m is akin to putting m in the envelop; opening the commitment is like opening the envelop and revealing the value inside. They have two basic properties. *Hiding* guarantees that a commitment reveals no information about the underlying message. *Binding* instead ensures that one cannot change its mind about the committed message; namely, it is not possible to open a commitment to two distinct values $m \neq m'$. This is the simplest form of cryptographic commitments that we will also call 'plain' commitments.

Notably, there is an additional intruiging property that various commitment schemes happen to have; the commitment value to the message is Succinct! This means that for many popular

commitment schemes, as for example the Pedersen commitment [174], regardless of the size of the message m the resulting commitment C has the same size. For example, whether the bit-size of m is $|m| = 1$ bit or $|m| = 10$ GB the commitment C has the same size, for example 256 bits. Therefore, the commitment C, from an alternative standpoint, is a compressing representation of the message m. This view of commitments make them an extremely valuable building block for modern Cryptography.

This fascinating property may not have been a direct objective in the earliest realizations of commitments, but there is an elegant theoretical explanation behind the property. It is a well-known fact that cryptographic commitments cannot be both perfectly (or statistically) binding and perfectly (or statistically) hiding. Now assume that a commitment scheme is perfectly hiding, thus 'not perfectly'[5] binding. This in turn means that the commitment function that maps messages to commitment values is not injective (otherwise the binding would be perfect). So the function is compressing: The range of the function is smaller than the domain. An informal corollary is therefore that every 'not perfectly' binding (or any perfectly hiding) commitment is compressing m.

Until now the discussion was restricted to plain commitment schemes. In those, the opening functionality is the most straightforward one can imagine: I just want to show the message that was committed. In a way it is an all-or-nothing opening function. *But what if we want to provide more elaborate opening capabilities?* For example: I want to commit to m but later reveal only its first bit, or just reveal that $f(m) = 1$ for some public function f.

Succinct Commitments with Fine-Grained Openings (i.e. that do not restrict themselves to an all-or-nothing opening capability) have been objects of intense research the past decade and they are also in the core of this thesis. Below we briefly recall some popular types of Fine-Grained Openings for Succinct commitments.

1.4.1 Set Accumulators

Set accumulators are commitments to sets with a special type of opening: Set membership opening.

A naive approach to check if an element is in a set is to go through all its entries. The complexity of this approach, however, is linear in the size of the set. *How to efficiently verify set membership then?* Cryptographic *accumulators* [29] provide an elegant solution to this problem. They allow a set of elements to be compressed into a succinct set commitment (the accumulator value or digest) and to generate membership proofs that are succinct and fast to verify. As a security guarantee we naturally require that it should be computationally infeasible to generate a false membership proof.

As of today, accumulators can be built from the following settings: hash-based (Merkle Trees) [162], RSA-based [29, 18], pairing-based [167] and lattice-based [220].

1.4.2 Vector Commitments

A Vector commitments (henceforth VC) [150, 66] is a special class of commitment schemes in which one can commit to a vector of elements $\boldsymbol{v}$ of length n and to later open the commitment at any position $i \in [n]$. In other words, one can convince a verifier that v_i is the i-th committed

[5] By 'not perfectly' we mean 'computationally'. Even though one could potentially open a commitment to two different values m, m', we assume that this is computationally infeasible (e.g. exponential running time).

value. Typically in VCs, the opening requires an addtional–to v_i–value to be outputted in order to get the verifier convinced: the opening proof. One can observe that (plain) commitments are already eligible as vector commitments: Using a (plain) commitment scheme we can separately commit to each element of $\boldsymbol{v}$, $v_1, \ldots, v_n$, provide the n commitments as the commitment to the vector and later open the commitment at the position i. However, the distinguishing feature of VCs is Succinctness: Both the commitment and an opening for any position i have size either independent of or poly-logarithmic in n. This property is, as usual, the one that makes the primitive challenging to construct. In terms of security, VCs should be *position binding*, i.e., one cannot open a commitment at position i to two distinct values v_i, v_i'. Interestingly, although one can also define a *hiding* property (analogously to plain commitments), in most applications we do not require this property.

VCs were formalized by Catalano and Fiore [66] who also proposed two constructions based on the CDH assumption in bilinear groups and the RSA assumption respectively. Noteworthy is that Merkle trees [162] are VCs with $O(\log n)$-size openings. Vector Commitments can also be instantiated from lattices, either resembling the Merkle tree structure [171, 148] or, recently, purely algebraicly [176].

Subvector Commitments. Two recent works [40, 145] proposed new constructions of vector commitments that enjoy a new type of opening *subvector openings* (also called *batch openings* in [40]) generalizing the opening of VCs. A VC with subvector openings (called SVC, for short) allows one to open a commitment at a collection of positions $I = \{i_1, \ldots, i_m\}$ while keeping the proof-size unaffected, namely of size independent (or polylogarithmic) of the vector's length n and the subvector length m.

1.4.3 Key-Value Map Commitments

In a nutshell, a Key-Value Map Commitment (henceforth KVC) is a generalization of Vector Commitments in which one commits to a collection of key-value pairs $(\mathsf{k}_i, \mathsf{v}_i)$, where k_i indicates the 'position' of the element in the map and v_i is the actual value. Afterwards, one can open the value of any key of the committed map. As usual, we require both the commitment and the opening values to be Succinct. One may observe that VCs are a special case of KVCs where keys are the consecutive integers in $\{1, \ldots, n\}$. Hence, the challenge is to realize KVCs with large arbitrary keys.

Existing schemes are based on hidden-order groups [40, 3], hash functions [162] or, very recently, lattices [89].

1.4.4 Polynomial Commitments

In Polynomial Commitments [137] (PCs) one can commit to a polynomial f. The opening functionality says that later they can open the polynomial to any evaluation point x, that is they can provide a proof that the initially committed polynomial f if evaluated at x gives y, $f(x) = y$. The crucial property of polynomial commitments is succinctness, both the commitment and the opening should be at most polylogarithmic in the degree of the polynomial f. Again, without this strigent efficiency property polynomial commitments would be already be implied from plain commitments. We could produce one commitment for each coefficient of the polynomial

and at the opening phase just open all the coeffients and let the verifier evaluate the polynomial themselves.

The security properties related to polynomial commitments are Evaluation Binding and Hiding. Evaluation Binding says that it should be infeasible to open a polynomial commitment to an evaluation point x with two different claimed outputs y, y'. Hiding intuitively demands that both the commitment and the evaluation opening do not leak anything about the committed polynomial, except, of course, the evaluation opening leaking $f(x)$.

The notion of Polynomial Commitments was introduced by Kate, Zaverucha and Goldberg [137], where they also gave a pairing-based construction. Other realizations include constructions from hash functions [209, 27] and, recently, lattices [5].

1.4.5 Functional Commitments

Functional Commitments (henceforth FCs) are Succinct Commitments with the most general form of openings. FCs were proposed by Libert, Ramanna, and Yung [149]. In FC, the sender commits to a vector $\boldsymbol{v}$ of length n and can later open the commitment to *functions* $f(\boldsymbol{v})$ of the committed vector. A distinguishing feature of FCs is that commitment and openings should be short, namely of size logarithmic or constant in n. In terms of security, binding for FCs means that the sender cannot open the same commitment to two different outputs of the same function, i.e., to prove that both y and $y' \neq y$ are $f(\boldsymbol{v})$. Functional commitments generalize (Sub)VCs and PCs: Both are FCs for a special class of functions.

In terms of realizations, Libert et al. [149] proposed an FC construction for linear functions from Pairings. Very recently, three concurrent works constructed FCs for any function, from lattices [89, 214, 15] and pairings [15].

1.5 Our contributions

The main event.

In this section we summarize the techincal contributions of the thesis. In order to not overwhelm the introduction, the summary of this section is–to the best possible–kept high-level. We are mostly motivating the contributions of this work and contextualizing the actual results. A more technically elaborate statement of the contributions is provided at the beginning of each technical chapter.

1.5.1 Zero-Knowledge Proofs for Set Membership of Singletons

The problem of proving set membership—that a given element x belongs to some set S—arises in many applications, including white/black-lists, voting and anonymous credentials, among others. More recently, this problem also appears at the heart of currency transfer and identity systems over blockchains.

Core to blockchains is the maintenance of the global state of the system across its nodes. This state is usually large and is encoded in data structures such as a UTXO set (unspent transaction outputs, intuitively the unspent coins) in Bitcoin and Zcash [24, 207], the set of account-balances in Ethereum, or the set of identities in Decentralized Identity (DID) systems (e.g., Iden3, Sovrin, Hyperledger Indy) [205, 206, 204]. In these systems executing a transaction typically involves

two steps, one "local" and one "global": *(i)* checking a given property with respect to the current state (e.g., the transaction is properly signed, some coins are spendable, some credentials exist), and *(ii)* modifying the global state (e.g., updating balances, adding a credential) and checking its correct update. The validity checks that are local to the transaction can for example involve checking a digital signature. Checking against the global state typically translate into *set-membership* ($x \in S$) or *set-update* ($S' \stackrel{?}{=} S \setminus \{x\} \cup \{x'\}$). Blockchain systems grow through time and so do their global states. Verifying this state *at scale* is a challenging problem: Every user, even one that only "passively" looks at the history of transactions, must re-execute them and store them to verify future ones.

A naive approach is to check if an element is in a set is to go through all its entries. The complexity of this approach, however, is unacceptable in many scenarios. Especially in blockchains, most of the parties (the verifiers) should run quickly.

How to efficiently verify set membership then? Cryptographic *accumulators* [29] provide a solution to this problem, allowing a set of elements to be compressed into a short value (the accumulator) and to generate membership proofs that are short and fast to verify (see Section 1.4.1). This idea [187, 201, 95] splits users into two groups. More "passive" users (aka verifiers) store only a *succinct digest* of the large set. A user proposing a transaction, on the other hand, has more information on the state (e.g., their account information) that it can use to *prove* the membership of some elements with respect to the digest.

As of today, we can divide constructions for accumulators into three main categories: Merkle Trees [162]; RSA-based [18, 57, 147, 40]; pairing-based [167, 88, 56, 224]. Approaches based on Merkle Trees[1] allow for short (i.e., $O(1)$) public parameters and accumulator values, whereas the witness for membership proofs is of size $\log(n)$, where n is the size of the set. In RSA-based constructions (which can be actually generalized to any group of unknown order [152], including class groups) both the accumulator and the witness are each a single element in a relatively large hidden-order group $\mathbb{G}$,[2] and thus of constant-size. Schemes that use pairings in elliptic curves such as [167, 56] offer small accumulators and small witnesses (which can each be a single element of a prime order bilinear group, e.g., 256 bits) but require large parameters (approximately $O(n)$) and a trusted setup.

In anonymous cryptocurrencies, e.g. Zerocash [24] (but also in other applications such as Anonymous Credentials [69] and whitelists), we also require *privacy*. That is, parties in the system would not want to disclose *which* element in the set is being used to prove membership. Phrased differently, one desires to prove that $u \in S$ without revealing u, or: the proof should be *zero-knowledge* [122] for u. As an example, in Zerocash users want to prove that a coin exists (i.e. belongs to the set of previously sent coins) without revealing which coin it is that they are spending.

In practice it is common that this privacy requirement goes beyond proving membership. In fact, these applications often require proving further properties about the accumulated elements, e.g., that for some element u in the set, property $P(u)$ holds. And this without leaking any more information about u other than what is entailed by P. In other words, we desire zero-knowledge for the statement $R^*(S, u) :=$ "$u \in S$ and $P(u)$".

One way to solve the problem, as done in Zerocash, is to directly apply general-purpose zero-knowledge proofs for R^*, e.g., [172, 126]. This approach, however, tends to be computationally

[1] We can include under this class currently known lattice-based accumulators such as [171, 148].

[2] The group $\mathbb{G}$ is typically $\mathbb{Z}_N^*$ where N is an RSA modulus. The size of an element in this group for a standard 128-bit security parameter is of 3072 bits.

intensive for the prover and ad-hoc. One of the questions we aim to tackle is that of providing a more efficient proof systems for set membership relations, that can also be modular.

Specifically, as observed in [61], the design of practical proof systems can benefit from a more modular vision. A modular framework such as the one in [61] not only allows for separation of concerns, but also increases reusability and compatibility in a plug-and-play fashion: the same proof system is designed once and can be reused for the same sub-problem regardless of the context[3]; it can be replaced with a component for the same sub-problem at any time. Also, as [61] shows, this can have a positive impact on efficiency since designing a special-purpose proof system for a specific relation can lead to significant optimizations. Finally, this compositional approach can also be leveraged to build general-purpose proof systems.

In this work we focus on applying this modular vision to designing *succinct zero-knowledge proofs for set membership*. Following the abstract framework in [61] we investigate how to apply commit-and-prove techniques [64] to our setting. Our approach uses commitments for composability as follows. Consider an efficient zero-knowledge proof system Π for property $P(u)$. Let us also assume it is commit-and-prove, i.e. the verifier can test $P(u)$ by simply holding a commitment $c(u)$ to u. Such Π could be for example a commit-and-prove NIZK such as Bulletproofs [52] or a commit-and-prove zkSNARK such as LegoGroth16 from [61] that are able to operate on Pedersen commitments $c(\cdot)$ over elliptic curves. In order to obtain a proof gadget for set membership, all one needs to design is a commit-and-prove scheme for the relations "$u \in S$" where *both* u and S are committed: u through $c(u)$ and S through some other commitment for sets, such as an accumulator.

Our main contribution is to propose a formalization of this approach and new constructions of succinct zero-knowledge commit-and-prove systems for set membership. In addition, we also extend our results to capture proofs of non-membership, *i.e., to show that $u \notin S$. For our constructions we focus on designing schemes where $c(u)$ is a Pedersen commitment in a prime order group $\mathbb{G}_q$. We focus on linking through Pedersen commitments as these can be (re)used in some of the best state-of-the-art zero-knowledge proof systems for general-purpose relations that offer for example the shortest proofs and verification time (see, e.g.,[126] and its efficient commit-and-prove variant [61]), or transparent setup and logarithmic-size proofs [52].*

Prior Approaches for Proving Set Membership for Pedersen Commitments. The accumulator of Nguyen [167], by the simple fact of having a succinct pairing-based verification equation, can be combined with standard zero-knowledge proof techniques (e.g., Sigma protocols or the celebrated Groth-Sahai proofs [128]) to achieve a succinct system with reasonable proving and verification time. The main drawbacks of using [167], however, are the large public parameters (i.e. requiring as many prime group elements as the elements in the set) and a high cost for updating the accumulator to the set, in order to add or remove elements (essentially requiring to recompute the accumulator from scratch).

By using general-purpose zkSNARKs one can obtain a solution with constant-size proofs based on Merkle Trees: prove that there exists a valid path which connects a given leaf to the root; this requires proving correctness of about $\log n$ hash function computations (e.g., SHA256). This solution yields a constant-size proof and requires $\log n$-size public parameters if one uses preprocessing zkSNARKs such as [172, 126]. On the other hand, often when proving a relation

[3]For instance, one can plug a proof system for matrix product $C = A \cdot B$ in any larger context of computation involving matrix multiplication. This regardless of whether, say, we then hash C or if A, B are in turn the output of a different computation

such as $R^*(S, u) :=$ "$u \in S$ and $P(u)$" the bulk of the work stems from the set membership proof. This is the case in Zcash or Filecoin[4] where the predicate $P(\cdot)$ is sufficiently small.

Finally, another solution that admits constant-size public parameters and proofs is the protocol of [57]. Specifically, Camenisch and Lysyanskaya showed how to prove in zero-knowledge that an element u committed in a Pedersen commitment over a prime order group $\mathbb{G}_q$ is a member of an RSA accumulator. In principle this solution would fit the criteria of the gadget we are looking for. Nonetheless, its concrete instantiations show a few limitations in terms of efficiency and flexibility. The main problem is that, for its security to hold, we need a prime order group (the commitment space) and the primes (the message space) to be quite large, for example[5] $q > 2^{519}$. But having such a large prime order group may be undesirable in practice for efficiency reasons. In fact the group $\mathbb{G}_q$ is the one that is used to instantiate more proof systems that need to interact and be linked with the Pedersen commitment.

1.5.2 Zero-Knowledge Proofs for Batch Set Membership

The above approach only supports membership proofs of a *single* element. *But what if we want, for instance, prove m transactions at once?* The above approach scales poorly when proving many transactions: m transactions require m proofs.

We continue the study of privacy-preserving proofs for RSA (more precisely Hidden Order Groups) accumulators. Privacy aside, RSA accumulators natively support efficient batching of membership proofs: The size of a membership proof for m elements is $O(1)$, exactly the same as for 1 element, regardless of m. However, this requires that during verification the m elements are known, which, of course, lacks privacy. Prior, to our work it was not known how to enhance batch proofs for RSA accumulators with zero-knowledge.

We provide a solution to this problem introducing a succinct zero-knowledge proof technique for proving membership of a batch of m elements. The proof size and verification time of our protocol are independent, $O(1)$, of both the size of the set n and the size of the batch m. Analogously to our previous contribution, the proof is modular and, following the commit-and-prove paradigm, is composable with any arbitrary statement for the elements that proving membership for.

1.5.3 Incrementally Aggregatable Vector Commitments

As mentioned in Section 1.4.2, Vector commitments (VCs) [150, 66] are a special class of commitment schemes in which one can commit to a vector $\boldsymbol{v}$ of length n and to later open the commitment at any position $i \in [n]$.

VCs were formalized by Catalano and Fiore [66] who also proposed two constructions based on the CDH assumption in bilinear groups and the RSA assumption respectively. Both schemes have constant-size commitments and openings but suffer from large public parameters that are $O(n^2)$ and $O(n)$ for the CDH- and RSA-based scheme respectively. Noteworthy is that Merkle trees [162] are VCs with $O(\log n)$-size openings.

[4] https://filecoin.io

[5] More specifically: the elements of a set need to be prime numbers in a range (A, B) such that $q/2 > A^2 - 1 > B \cdot 2^{2\lambda_{st}+2}$. If aiming at 128 bits of security level one can meet this constraint by choosing for example $A = 2^{259}$, $B = 2^{260}$ and $q > 2^{519}$.

Then in 2019, two concurrent works introduced the notion of subvector openings [40, 145] in which one can open a commitment at a collection of positions $I = \{i_1, \ldots, i_m\}$ without increasing the size of the proof (or of the commitment). This property has been shown useful for reducing communication complexity in several applications, such as PCP/IOP-based succinct arguments [145, 40] and keyless Proofs of Retrievability (PoR) [105].

Here, we continue the study of VCs with subvector openings with two main goals: (1) improving their efficiency, and (2) enabling their use in decentralized systems.

With respect to efficiency, although the most attractive feature of SVCs is the constant size of their opening proofs, a drawback of all constructions is that generating each opening takes at least time $O(n)$ (i.e., as much as committing). This is costly and may harm the use of SVCs in applications such as the ones mentioned above.

When it comes to decentralization, VCs have been proposed as a solution for integrity of a distributed ledger (e.g., blockchains in the account model [40]): the commitment is a succinct representation of the ledger, and a user responsible for the i-th entry can hold the corresponding opening and use it to prove validity of v_i. In this case, though, it is not obvious how to create a succinct subvector opening for, say, m positions held by *different* users each responsible *only* of its own position/s in the vector. We elaborate more on the motivation around this problem in Section 1.5.3.2.

1.5.3.1 A new notion for SVCs: incremental aggregation

To address these concerns, we define and investigate a new property of vector commitments with subvector openings called incremental aggregation*. In a nutshell, aggregation means that different subvector openings (say, for sets of positions I and J) can be merged together into a single* concise *(i.e., constant-size) opening (for positions $I \cup J$). This operation must be doable* without *knowing the entire committed vector. Moreover, aggregation is incremental if aggregated proofs can be further aggregated (e.g., two openings for $I \cup J$ and K can be merged into one for $I \cup J \cup K$, and so on an unbounded number of times) and disaggregated (i.e., given an opening for set I one can create one for any $K \subset I$).*

While a form of aggregation is already present in the VC of Boneh et al. [40], there it can be performed only once. In contrast, *we define (and construct) the first VC schemes where openings can be aggregated an unbounded number of times*. This incremental property is key to address efficiency and decentralized applications of SVCs, as we detail below.

Incremental aggregation for efficiency. We show that any Incrementally Aggregatable SVCs admits efficient online proof computation, using offline precomputation. We demonstrate this for *any* Incrementally Aggregatable SVCs by introducing *generic* algorithms for efficient precomputation and online proof computation, assuming the Incremental Aggregation property. Notably, the precomputation technique is flexible allowing for tradeoffs between precomputed information stored and online computation time.

Incremental aggregation for decentralization. Essentially, by its definition, incremental aggregation enables generating subvector openings in a distributed fashion. Namely, consider a scenario where different parties each hold an opening of some subvector; using aggregation they can create an opening for the union of their subvectors, moreover the incremental property allows them to perform this operation in a non-coordinated and asynchronous manner, i.e. without the need of a central aggregator. We found this application of incrementally aggregatable SVCs

to decentralized systems worth exploring in more detail. To fully address this application, we propose a new cryptographic primitive called verifiable decentralized storage which we discuss next.

From Updatable VCs to Verifiable Decentralized Storage. In their seminal work on VCs, Catalano and Fiore [66] also defined updatable VCs. This means that if one changes the i-th value of a vector from v_i to v'_i it is possible to update: a commitment C to $\boldsymbol{v}$ into a commitment C' to $\boldsymbol{v}'$, a valid opening for C (at any position) into a valid opening for C'. And importantly, these updates can be done without knowing the entire vector and in time that depends only on the number of modified positions. As an application, in [66] it is shown how updatable VCs can be used to realize verifiable databases (VDB) [28], a primitive that enables a client to outsource a database to an untrusted server in such a way that the client can retrieve (and update) a DB record and be assured that it has not been tampered with by the server.

In this work we study how to extend this model to a scenario where storage is distributed across different nodes of a decentralized network. This problem is motivated by the emerging trend of *decentralized storage networks* (DSNs), a decentralized and open alternative to traditional cloud storage and hosting services. Filecoin (which is built on top of IPFS), Storj, Dat, Freenet and general-purpose blockchains like Ethereum[6] are some emerging projects in this space.

Our contribution is to put forward a new cryptographic primitive called verifiable decentralized storage *(VDS) that can be used to obtain data integrity guarantees in DSNs. We propose a definition of VDS and a construction obtained by extending the techniques of our VC scheme; in particular, both incremental aggregation and the arguments of knowledge are key ingredients for building a cost-effective VDS solution.*

In the following section we elaborate on the VDS problem: we begin by discussing the requirements imposed by DSNs, and then give a description of our VDS primitive and realization.

1.5.3.2 Verifiable Decentralized Storage

Decentralized Storage Networks. *Openness* and *decentralization* are the main characteristics of DSNs: anyone can enter the system (and participate as either a service provider or a consumer) and the system works without any central management or trusted parties. Abstracting from the details of each system, a DSN consists of participants called *nodes* that can be either a storage provider (aka *storage node*) or a *client node*. Akin to centralized cloud storage, a client can outsource the storage of large data; the key difference of DSN however is that storage is provided by, and distributed across, a collection of nodes that can enter and leave the system at their wish. Also, DSNs can have some reward mechanism to economically incentivize storage nodes.

The openness and the presence of economic incentives raise a number of security questions that need to be solved in order to make these systems viable. In this work, we focus on the basic problem of ensuring that the storage nodes of the DSN are doing their job properly, namely:

How can any client node check that the whole DSN
is storing correctly its data (in a distributed fashion)?

[6] https://filecoin.io, https://storj.io, https://datproject.org, https://freenetproject.org, https://www.ethereum.org

While this question is well studied in the centralized setting where the storage provider is a single server, for decentralized systems the situation is less satisfactory. In what follows we elaborate on the problem and the desired requirements, and then on our solution.

The Problem of Verifiable Decentralized Storage in DSNs. Consider a client who outsources the storage of a large file F, consisting of blocks $(F_1, \ldots, F_N)$, to a collection of storage nodes. A storage node can store a portion of F and the network is assumed to be designed in order to self-coordinate so that the whole F is stored, and to be fault-resistant (e.g., by having the same data block stored on multiple nodes). Once the file is stored, clients can request to the network to retrieve or modify a data block F_i (or more), as well as to append (resp. delete) blocks to (resp. from) the file.

In this scenario, our goal is to formalize a cryptographic primitive that can provide clients with the guarantee of *integrity of the outsourced data and its modifications*. The basic idea of VDS is that: (i) the client retains a short *digest* δ_F that "uniquely" points to the file F; (ii) any operation performed by the network, be it a retrieval or a file modification, can be proven by generating a short *certificate* that is publicly verifiable given δ_F.

This problem is similar in scope to the one addressed by authenticated data structures (ADS) [199]. But while ADS is centralized, VDS is not. In VDS nodes act as storage in a distributed and uncoordinated fashion. This is more challenging as VDS needs to preserve some basic properties of the DSN:

Highly Local. The file is stored across multiple nodes and no node is required to hold the entire F: in VDS every node should function with only its own local view of the system, which should be much smaller than the whole F, e.g., logarithmic or constant in the size of F. Another challenge is dynamic files: in VDS both the digest and the local view must be *locally* updatable, possibly with the help of a short and publicly verifiable update advice that can be generated by the node who holds the modified data blocks.

Decentralized Keyless Clients. In a decentralized system the notion of a client who outsources the storage of a file is blurry. It may for example be a set of mutually distrustful parties (even the entire DSN in the most extreme case, e.g., the file is a blockchain), or a collection of storage nodes themselves that decide to make some data available to the network. This comes with two implications:

1. *VDS must work without any secret key* on the clients side, so that everyone in the network can delegate and verify storage. This *keyless* setting captures not only clients requiring no coordination, but also a stronger security model. Here the attacker may control both the storage node and the client, yet it must not be able to cheat when proving correctness of its storage. The latter is crucial in DSNs with economic rewards to well-behaving storage nodes[7].
2. *In VDS a file F exists as long as some storage nodes provide its storage* and a pointer to the file is known to the network through its digest. When a file F is modified into F' and its digest δ_F is updated into $\delta_{F'}$, both versions of the file may coexist. Forks are possible and it is left to each client (or the application) to choose which digest to track: the old one, the new one, or both.

Non-Coordinated Certificates Generation. There are multiple ways in which data retrieval queries can be answered in a DSN. In some cases, e.g., IPFS, after executing a P2P protocol

[7] Since in a decentralized system a storage node may also be a client, an attacker could "delegate storage to itself" and use the client's secret key to cheat in the proof in order to steal rewards (akin to the so-called "generation attack" in Filecoin [144]).

to discover the storage nodes holding the desired data blocks, one gets such blocks from these nodes. In other cases (e.g., Freenet [77] or the original Gnutella protocol), data retrieval is also answered in a peer-to-peer non-coordinated fashion. When a query for blocks $i_1, \ldots, i_m$ propagates through the network, every storage node replies with the blocks that it owns and these answers are aggregated and propagated in the network until they reach the client who asked for them. Notably, data aggregation and propagation may follow different strategies.[8] To accommodate flexible aggregation strategies, in VDS we consider the incremental aggregation of query certificates in an arbitrary and bandwidth-efficient fashion. For example, short certificates for file blocks F_i and F_j should be mergeable into a *short* certificate for (F_i, F_j) and this aggregation process should be carried on and on. Noteworthy that having certificates that stay short after each aggregation keeps the communication overhead of the VDS integrity mechanism at a minimum.[9]

1.5.4 Inner Product Functional Commitments From Set Accumulators

We consider the problem of realizing functional commitments that admit constant-size public parameters generated using a transparent public-coin setup. It is not hard to see that this question has a positive answer if one is willing to rely on the random oracle heuristic. In this case, one can build a functional commitment by using a succinct commitment scheme and a SNARK with transparent setup [163, 6, 210, 25, 23, 75, 190, 191, 223] thanks to which one can generate an opening through a SNARK proof for the NP statement that $y = f(\boldsymbol{v})$ and the commitment C opens to $\boldsymbol{v}$. For an NP statement of size N, most existing SNARKs with a transparent setup have proofs of length at least $O(\lambda \log N)$, where λ stands for the security parameter. The only exception are SNARKs based on the approach of [163] instantiated with a constant-query PCP and constant-size vector commitments over class groups [40, 145]. Such an approach however involves the non-black-box use of the code of the commitment scheme and relies on the heavy machinery of PCPs and SNARKs. Notably, this has to be the case even if one wants to construct an FC for simple functionalities like inner products or polynomial evaluations (aka polynomial commitments). In contrast, we ask whether we can build FC schemes for inner products in a 'simple' way, i.e., without relying on powerful (and computationally burdensome) primitives.

Indeed, when considering FCs for inner products or polynomial commitments, all 'simple' constructions in the literature have logarithmic opening size. For instance, an inner product argument such as Bulletproofs [52] yields an FC for linear functions in which openings consist of $2 \cdot \log n$ elements of a group $\mathbb{G}$ where the discrete logarithm problem is hard.

To summarize, to the best of our knowledge, there is no simple functional commitment (including polynomial commitments) that admits constant-size and transparent public parameters and constant-size openings in the literature. The only exceptions are a few vector commitment constructions [40, 59] in groups of unknown order, which, however, are functional commitments for the very specific, non-algebraic, functionality of position-opening. Therefore, the main question we ask in this work is:

> *Can we build a simple inner-product functional commitments with constant-size public parameters consisting of a uniformly random string and with constant-size openings?*

[8] E.g., in Freenet data is sent back along the same route the query came through, with the goal of providing anonymity between who requests and who delivers data.

[9] The motivation of this property is similar to that of sequential aggregate signatures, see e.g., [157, 48].

The main result of this part of the thesis is the construction of the first functional commitments that answer the above question in the affirmative.

1.5.5 Key-Value Maps from Any Vector Commitments

We present a construction that compiles any *vector commitment into a key-value map commitment (KVC) [40, 3] for arbitrary-size keys. Existing schemes are based on hidden-order groups [40, 3], Merkle trees or, very recently, lattices [89].*[10] *In Section 8.3, we present a generic and black-box construction of (updatable) KVC obtained by combining any (updatable) VC and cuckoo hashing [169], a probabilistic data structure. Through this generic construction, we obtain new efficient KVCs; notably, the first updatable KVCs for large keys based on pairings.*

Furthermore, we observe that KVCs (for large keys) imply accumulators (for large set-universes). By putting this observation together with our VC-to-KVC compiler, we obtain a way to convert VCs into accumulators. This connection was previously shown by Catalano and Fiore in [66] *but only for small set-universes*. Our results thus bridge this gap. We close the circle in showing the equivalence of VCs and universal accumulators, since the reversed implication (i.e., building VCs from universal accumulators) has been recently shown by Boneh, Bunz and Fisch [40]. An outstanding implication is that our result yields the first accumulator for large universe based on the CDH problem in bilinear groups. Prior to our work, this result could only be achieved by using non-black-box techniques (e.g., a Merkle tree with a CDH-based VC).

Cuckoo hashing applications. Cuckoo Hashing has been used extensively in many contexts in cryptography, mainly to boost efficiency in oblivious two-party computations (*e.g.* in [177, 178, 7, 173]). However, in most of these contexts, due to the oblivious security model, the adversary does not have direct access to the cuckoo hash functions. Only recently, a new work has discussed cuckoo hashing in this perspective [219].

We propose a new cryptographic application where cuckoo hashes can be publicly computed. In a way, our result show how vector commitment techniques can mitigate the shortcomings of publicly computable cuckoo hashing, as combining them with vector commitments enable their use while keeping constructions succinct and efficient. We believe that this approach can serve as inspiration for future applications.

[10] One can also use polynomial commitments, e.g., [137], in combination with interpolation but to the best of our knowledge this KVC is not updatable.

2 RELATED WORK

2.1 Commitments schemes

Commitment schemes have a long history in Cryptography. The idea of committing through cryptography appeared in the early 80s starting with the seminal work of Blum [37], in works by Even [99] and Shamir, Rivest and Adleman [194]. The first work that formalized the notion dates back to 1988 by Brassard, Chaum and Crépeau [46]. In 1991 Naor showed their connection with Pseudorandom Generators [166].

Regarding concrete constructions, the most popular commitment schemes in the discrete-logarithm setting are Elgamal [97] and Pedersen [174] commitments. Elgamal commitment scheme is perfectly binding and computationally hiding. Pedersen commitments are perfectly hiding and computationally binding, therefore they are compressing (Succinct). One can also obtain a commitment scheme from the RSA assumption [184]. The same scheme can be generalized to work over any group of unknown order [111, 85]. Latticed-based commitment schemes include the one obtained from the Ajtai function [4] and the BDLOP commitment [20] (that in turn improved on [33]). Finally, a commitments schemes can be generically obtained by any collision-resistant hash function, where hiding holds if it is modeled as a random oracle [21].

2.2 Vector Commitments

Here, we briefly survey the developments in the area of vector (and functional) commitments. In this section we mostly refer to 'algebraic' schemes, that do not make use of more powerful tools such as SNARKs. We discuss the latter in Section 2.6.

The first vector commitment construction first appeared in a paper by Libert and Yung [150], providing a construction from bilinear groups. Catalano and Fiore [66] formally defined the primitive and showed realizations from the RSA and Computational Diffie-Hellman assumptions. We note that Merkle trees [162] can serve as a vector commitments although initially not seen as such. Papamanthou et al. [171] and Libert et al. [148] gave lattice-based constructions based on the Merkle-tree paradigm. A generalization of Merkle trees are Verkle trees, formally appearing in [143] (though the core techinque was already present in [67, 150]). A Verkle-tree

is essentially a Merkle tree with larger arity, where instead of a 2-to-1 hash function an n-to-1 vector commitment is used.

More recently the field of algebraic VCs exploded. In 2018 Chepurnoy et al. [73] showed a pairing-based construction using multi-linear extension polynomials [227]. Lai and Malavolta [145] and Boneh et al. [40] introduced the notion of vector commitments with subvector openings,[11] where one can open a commitment in multiple positions with a single (succinct) proof. The former work presented constructions from pairings and groups of unknown order (extending the constructions of [66]) and the latter from groups of unknown order (using RSA accumulators [29, 18]). Tomescu et al. [202] also introduced a pairing-based vector commitment using the KZG polynomial commitment [137] and Lagrange interpolation. Gorbunov et al. [123] put forth the notion of cross-commitment aggregation, where one can provide a single (succinct) opening proof for multiple committed vectors, and provided a construction from pairings using the [150] VC. Tomescu et. al. [203] extended the notion to cross-incremental commitment (dis)aggregation that admits both incremental aggregation (see Section 1.5.3) and cross-commitment aggregation, providing a construction from groups of unknown order. Srinivasan et. al. [197] showed a pairing-based tree-construction that provides trade-offs between proof size and proof-update time, formalizing this as Maintainable VCs. Campanelli et al. [62] constructed another pairing-based tree-construction that is Maintainable, and additionally showed a restricted form of incremental aggregation from pairings that does not though support disaggregation and is not aggregation-history oblivious. Another construction on the same spirit came later by Liu et al. [156]. Furthermore, Wang et al. [213] showed generic techniques for VCs to achieve Maintainability. Peikert et al. showed the first algebraic lattice-based VC construction [176]. More lattice-based VC constructions came very recently by Albrecht et al. [5], de Castro and Peikert [89] and Wee and Wu [214].

Functional Commitments. Funcional Comitments were introduced By Libert et al. on 2016 [149], where they also provided a construction for linear function-openings from bilinear groups. Until recently this was the only (algebraic) construction of FCs known. On 2020, Lipmaa and Pavlyk [154] demonstrated techniques to overcome the linearity barrier, giving pairing based FC constructions for a larger class of functions, namely for the class of semi-sparse polynomials. On 2021, Peikert et al. [176] showed a lattice-based FC for all functions, however in a weaker model where a trusted party should stay online providing helping information on-the-fly for the opening-functions of interest. Arun et al. [8] proposed a functional commitment construction for inner products with transparent setup and constant-size openings. On 2022, three independent works appeared concurrently [89, 214, 15] showing construction of FCs for all (NP) functions. Balbás et al. [15] provided constructions from Pairings and Lattices using the notion of 'chainable' functional commitments. De Castro and Peikert [89] gave lattice-based schemes using Fully Homomorphic Encryption techniques [117], however the opening proofs are not succinct on the size of the vector but on the size of the function representation. Finally, Wee and Wu [214] also used Fully Homomorphic Encryption techniques [117] to construct lattice-based FCs for all functions with fully-succinct openings, introducing a new class of lattice assumptions.

Functional Commitments for linear functions can also be realized through folding-techniques that build inner-product arguments [44, 52].

Applications. There are numerous possible applications of vector commitments, here we mention the most prominent examples in the literature. [66] discussed applications of VCs to

[11] In [40] dubbed as vector commitments with 'batching'.

Verifiable Databases. [171] constructed VCs for Streaming Authenticated Data Structures. [73], [40], [202] and [123] discussed the use of Vector Commitments in Stateless Blockchains.

On the cryptographic side, already in 1994 Micali [163] showed applications of Merkle trees to succinct non-interactive proofs of knowledge. [145] and [40] elaborated on this idea, demonstrating that subvector VCs can produce (zk) proofs with smaller size. [5] also constructed lattice-based SNARKs based on FCs. [8] showed extensions and applications of their FC scheme to building constant-size SNARKs with transparent setup. [15] and [68] of FCs to homomorphic signatures [134, 42]. Finally, [105] shows applications of VCs to proofs of space and proofs/retrievability.

2.3 Polynomial Commitments

Polynomial commitments were introduced in [137], where a pairing-based construction with constant-sized openings was also presented. Vlasov and Panarin [209] (further studied in [27]) built a polynomial commitment scheme from Merkle trees and Interactive Oracle Proofs of Proximity, in concrete the FRI protocol, [22].

Furthermore, we note that polynomial commitments are implied by functional commitments, therefore all the aforementioned FC schemes imply a PC scheme.

2.4 Key-Value Map Commitments

The notion Key-Value Map Commitments was introduced by Boneh et al. [40] where they also presented a construction from Groups of Unknown order. Different KVC constructions from Groups of Unknown order exist [3, 203]. Recently deCastro and Peikert [89] showed a construction from Lattices.

KVCs are implied by any sparse vector commitment as Merkle Trees. In [73] a generic construction from VCs and hash function was shown and in [40] a generic construction from Accumulators and hash functions. Both compilers though give KVCs with a weaker security property, where we need to make sure that the commitments was honestly generated for the scheme to be secure.

2.5 Set Membership protocols

Accumulators. Accumulators were introduced by Benaloh and de Mare [29]. Constructions for large universe exist from RSA groups [18, 57], Groups of Unknown Order [152, 40], q-type assumptions in bilinear groups [167, 56], and Merkle trees. The recent work of de Castro and Peikert [89] also implies an accumulator from lattices.

Privacy-preserving Accumulators. Different notions of privacy for accumulators have been proposed in the literature. Derler et al. [91] formalized the notion of indistinguishability for accumulators, where the digest of the set should leak no information about the set to a verifier. Ghosh et al. [119] strenghtened the notion, introducing ‘zero-knowledge accumulators’, where even after updates the digest and the (non)-membership proofs preserve the privacy of the set. Later [228, 136, 72] built on this model. Finally, very recently, Baldimtsi et al. [16] introduced

the notion of 'Oblivious Accumulators', enhancing the above, preserving the privacy even to parties that actively participate in the system (and hold (non-)membership proofs). We note that in all these the (non-)membership proof itself leaks the element that is proving (non-)membership; the objective is to preserve the privacy of the rest of the elements of the set.

Regarding anonymous set membership the first construction appears in [57] where the authors provide a zero-knowledge protocol for set membership for RSA accumulators. More recently, in the bilinear groups setting, Srinivasan et al. [198], among other improvements on the functionalities and security properties of the actual pairing-based accumulator, provide zero-knowledge (batch) proofs for membership and non-membership over the Nguyen accumulator [167]. Another relevant, rapidly developing, line of work has to do with succinct zero-knowledge lookup arguments. That is, given a committed *vector* of n elements, one proves that a number m of committed elements are all values of the vector in some hidden position, while retaining the elements secret. The proofs are succinct in both n and m. This line of work was initiated by the seminal work of Zapico et al. [221] followed by a number of works improving the prover's complexity [182, 112, 222, 96]. All these constructions work over bilinear groups. Finally, Lipmaa and Parisella [153] (building on [80] and [79]) construct succinct set (non-)membership NIZKs from falsifaible assumptions. That is, the objective of their work is constructing efficient NIZKs for set (non-)membership that can be proven secure in the standard model and assuming only falsifiable assumptions.

Succinct Set mebership proofs. Ozdemir et al. [168] recently proposed a solution to scale operations on RSA accumulators inside a SNARK. In particular, their approach scales when these operations are *batched* (i.e., when proving membership of many elements at the same time); for example, they surpass a 2^{20}-large Merkle tree when proving batches of at least 600 elements. This approach is attractive in settings where we can delegate a *large* quantity of these checks to an untrusted server as there is a high constant proving cost. Their solution is not zero-knowledge and does not provide privacy guarantees.

In the lattice-based setting Lyubashevsky et al. [158] in 2021 presented zero-knowledge proofs for set membership from (ideal) lattices. The proofs are post-quantum, however with linear (in the size of the set) verification and, as typical in the lattice-based cryptography, the concrete proof size is still bigger than the one in classical settings.

2.6 (zk)-SNARKs

zkSNARKs [163, 36] is one of the most rapidly developed areas in Cryptography during the last decade. There exist different categorizations for example according to the setup assumptions: relation-specific-setup SNARKs e.g. [114, 172, 126], universal SNARKs e.g. [127, 159, 113, 74, 58, 183, 155, 71] or SNARKs with transparent setup [163, 6, 210, 25, 23, 75, 190, 191, 223]. The most widely deployed SNARKs are Groth16, Plonk and STARKs [126, 113, 22].

As noted in Section 1.5, SNARKs are very powerful tools with which one can generically provide solutions to all the aforementioned problems. With a succinct plain commitment scheme and a (zk)SNARK one can construct: a vector/polynomial/key-value map commitment and in general functional commitments for all functions and succinct privacy-preserving set (non-)membership proofs. However, as also argued in Section 1.5, this comes at high proving-cost, since, that way, one has to decompose cryptographic operations, encode them in arithmetic

circuits and then prove them. Another, obstacle is that SNARKs are typically non-malleable meaning that any updatability property becomes infeasible.

2.7 Other Related Work

Cuckoo hashing in cryptography. Cuckoo hashing has been used in Cryptography in oblivous access primitives such as Oblivious RAM [177], Private Set Intersection [178], Private Infomration Retrieval [7], and Searchable Encryption [173]. Recently, Yeo gave a formal treatment from a cryptographic perspective [219], again with the objective of discussing applications to PIR.

ZKP-batch Verifiable computation with state. Verifiable computation and zkSNARKs have a vast literature; a complete coverage goes beyond the scope of this paper, e.g., see this recent survey [211] and references therein. More relevant to our work are some works that address the problem of verifiable computation (or zkSNARKs) with respect to succinct digests. Pantry [47] use Merkle trees to model RAM computations. Fiore et al. [103] propose hash&prove as well as accumulate&prove protocols that avoid expensive hash encodings in the circuit, but their solutions require the SNARK prover to do work *linear* in the hashed/accumulated set, which limits their scalability to large sets. The same limitation applies to the efficient commit-and-prove SNARKs [78, 61] as well as to the vSQL scheme of Zhang et al. [225] and TRUESET by Kosba et al. [141]. Also, all these schemes [103, 61, 225] require public parameters linear in the largest set. ADSNARK [14] can generate proofs on authenticated data; this setting is similar to accumulated sets except that inserting data in the set requires a secret authentication key; proofs in [14] are succinct *only* when verifiers know the secret authentication key.

3

BACKGROUND

In this chapter we present some basic notions and results of Cryptography that we build on in this thesis. Most of the results presented in this chapter are based on prior, to this thesis, works. We include them in order to ease the presentation of our technical contributions and to serve the self-containedness of the thesis.

3.1 Basic Mathematics and Notation

To begin with, we recall some relevant basic mathematical concepts and notation.

Throughout the thesis, we denote the security parameter with a natural number $\lambda \in \mathbb{N}$ and its unary representation with 1^λ and we assume that all the algorithms of the cryptographic schemes take as input 1^λ, which may be omitted from the list of inputs. The security parameter indicates the level of security of the cryptographic schemes: security parameter λ means that one needs $\Theta(2^\lambda)$ computational power to break the scheme.

The set of all univariate polynomial functions, with formal variable λ, is denoted by $\mathsf{poly}(\lambda)$. Similarly $\mathsf{polylog}(\lambda)$ is the set of poly-logarithmic functions in λ, for example $3.1\log^7(\lambda)$ or $\log^3(\lambda) + 100\log(\lambda)$. A function $\epsilon(\lambda)$ is called *negligible* – denoted $\epsilon(\lambda) \in \mathsf{negl}(\lambda)$ – if it vanishes faster than the inverse of any polynomial, that is $\epsilon(\lambda) < 1/f(\lambda)$ for every $f \in \mathsf{poly}(\lambda)$. Sometimes we will abuse the notation and write $\kappa = \mathsf{poly}(\lambda)$ or $\kappa = \mathsf{negl}(\lambda)$ with '$=$' instead of '$\in$'. By $O_\lambda(n)$ will mean $O(\lambda n)$ (and *not* $O(\mathsf{poly}(\lambda)n)$).

If D is a distribution, we denote by $x \leftarrow D$ the process of sampling x according to D. More usually, if S is a set, we denote we denote by $x \leftarrow\!\!\$\, S$ the process of sampling x uniformly at random from S (in other words this is the uniform distribution with domain S). An ensemble $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ is a family of probability distributions over a family of domains $\mathcal{S} = \{S_\lambda\}_{\lambda\in\mathbb{N}}$, and we say that two ensembles $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}' = \{D'_\lambda\}_{\lambda\in\mathbb{N}}$ are statistically indistinguishable (denoted by $\mathcal{D} \approx_s \mathcal{D}'$) if $\frac{1}{2}\sum_x |D_\lambda(x) - D'_\lambda(x)| < \mathsf{negl}(\lambda)$. If $\mathcal{A} = \{\mathcal{A}_\lambda\}$ is a (possibly non-uniform) family of circuits and $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ is an ensemble, then we denote by $\mathcal{A}(\mathcal{X})$ the ensemble of the outputs of $\mathcal{A}_\lambda(x)$ when $x \leftarrow X_\lambda$. We say two ensembles $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}' = \{D'_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable (denoted by $\mathcal{D} \approx_c \mathcal{D}'$) if for every non-uniform polynomial time distinguisher $\mathcal{A}$ we have $\mathcal{A}(\mathcal{D}) \approx_s \mathcal{A}(\mathcal{D}')$.

An algorithm $\mathcal{A}$ is said PPT if it is modeled as a probabilistic Turing machine that runs in time $T = \mathsf{poly}(\lambda)$. We denote by $y \leftarrow \mathcal{A}(x)$ the process of running $\mathcal{A}$ on input x and assigning

the output to y. For the rest of the thesis by $\mathcal{A}$ we will denote the 'adversary' of the cryptographic schemes.

We use bold lowercase letters to denote a vector, e.g. $\boldsymbol{v} = (v_1, \ldots, v_n)$, where v_i is its entry at position i. Sometimes we also write $(v_j)_{j\in[n]}$ (or even $(v_j)_j$ when there is no ambiguity for the set $[n]$) for the vector $\boldsymbol{v} = (v_1, \ldots, v_n)$. Given vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$, $\mathbf{cat}((\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m))$ will be the vector of concatenated vectors $\boldsymbol{v}_1 \| \ldots \| \boldsymbol{v}_m$. For any positive integer $n \in \mathbb{N}$, we write $[n]$ for $\{1, \ldots, n\}$ and for any positive integers $A, B \in \mathbb{Z}, A < B$ we write $[A, B]$ for the set $\{A, A+1, \ldots, B-1, B\}$. For a set S, $|S|$ denotes its cardinality. The operator $\|\cdot\|$ is used for the bit-size, i.e. $\|x\| = \lceil \log(x) \rceil$, for $x \in \mathbb{N}$, which intuitively denotes the number of bits needed to represent x.

We denote $\mathbb{P} := \{e \in \mathbb{N} : p \text{ is prime}\}$ the set of all positive integers $p > 1$ such that they do not have non-trivial (i.e. different than e and 1) factors. More specifically, given two positive integers $A, B > 0$ such that $A < B$, we denote with $\mathbb{P}[A, B]$ the subset of $\mathbb{P}$ of numbers lying in the interval $[A, B]$, i.e., $\mathbb{P}[A, B] = \{p \in \mathbb{N} : e \text{ is prime } \wedge \ A \leq p \leq B\}$. According to the well known prime number theorem $|\mathbb{P}[1, B]| = \Theta(\frac{B}{\log B})$ which results to $|\mathbb{P}[A, B]| = \Theta(\frac{B}{\log B}) - \Theta(\frac{A}{\log A})$. Furthemore, we write $\mathbb{P}(\lambda)$ for the set of all primes of bit-size λ, i.e. $\mathbb{P}(\lambda) = \{p : p \text{ prime } \wedge \|p\| = \lambda\} = \{p : p \text{ prime } \wedge 2^{\lambda-1} < p < 2^{\lambda}\}$.

3.2 Bilinear Groups

Some of our cryptographic constructions in the subsequent chapters work over Bilinear (aka Pairing) Groups of prime order.

Formally, a Bilinear Group generator algorithm $\mathcal{BG}$ takes as input a security parameter 1^{λ} and outputs a description $\mathsf{bg} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$, where p is a prime of $\Theta(\lambda)$ bits, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of order p, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map. We require that the group operations in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and the bilinear map e are computable in deterministic polynomial time in λ. Let $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $g_T = e(g_1, g_2) \in \mathbb{G}_T$ be the respective generators.

3.3 Groups of Unknown Order

For some our constructions we use groups of unknown (aka hidden) order $\mathbb{G}$, i.e., groups where computing the order is hard. We let $\mathsf{Ggen}(1^{\lambda})$ be an efficient probabilistic algorithm that generates such a group $\mathbb{G}$ with order in a specific range $[\mathsf{minord}(\mathbb{G}), \mathsf{maxord}(\mathbb{G})]$ such that $\frac{1}{\mathsf{minord}(\mathbb{G})}, \frac{1}{\mathsf{maxord}(\mathbb{G})}, \frac{1}{\mathsf{maxord}(\mathbb{G}) - \mathsf{minord}(\mathbb{G})} \in \mathsf{negl}(\lambda)$. Potential candidates are class groups of imaginary quadratic order [49] and RSA groups [184] where the factorization is unknown. The instantiation through class groups is the one that admits a public-coin (aka transparent) setup.

For the most part of our constructions we make black-box use of the groups, their properties and the relevant computational assumptions, without going into the concrete instantiation or implementation of the group.

3.3.1 Hardness Assumptions

Now we recall some assumption that are commonly believed to be true in (carefully chosen) groups of unknown order. By "hardness assumption" in Cryptography (and in Complexity

Theory) we mean that a computational problem is believed not to admit a PPT algorithm. An example is the Factoring assumption, that the problem of Factoring a large integer is assumed to be hard. Typically, the hardness assumptions come with strong evidence from Cryptanalysis, that intense cryptanalysis' efforts have failed to solve the problem in polynomial time.

In particular, we (partially) base some of our constructions in one or more of the assumptions that are formally defined below. The 2-Strong RSA assumption [53], the Adaptive Root assumption [215], the Low Order assumption [39] and the Strong Distinct-Prime-Product Root assumption [145].

Definition 1 (2-Strong RSA assumption [53])**.** *We say that the* 2-strong RSA assumption *holds for* Ggen *if for any PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{l} u^e = g \\ \wedge e \neq 2^k,\ k \in \mathbb{N} \end{array} : \begin{array}{l} \mathbb{G} \leftarrow \mathsf{Ggen}(\lambda) \\ g \leftarrow_{\$} \mathbb{G} \\ (u, e) \leftarrow \mathcal{A}(\mathbb{G}, g) \end{array}\right] = \mathsf{negl}(\lambda)$$

The 2-Strong RSA assumption is a special case of r-Strong RSA assumption, introduced in [53]. The latter is in turn a generalization of (and trivially reduces to) the standard Strong RSA assumption [18] (where $r = 1$). Taking square roots can be done efficiently in Class Groups of imaginary quadratic order [45], but for higher order roots it is believed to be hard. Thus 2-Strong RSA assumption is believed to hold. For RSA groups the (plain) Strong RSA is a standard assumption.

Definition 2 (Adaptive Root assumption [215])**.** *We say that the* adaptive root assumption *holds for* Ggen *if for any PPT adversary* $(\mathcal{A}_1, \mathcal{A}_2)$*:*

$$\Pr\left[\begin{array}{l} u^\ell = w \\ \wedge w \neq 1 \end{array} : \begin{array}{l} \mathbb{G} \leftarrow \mathsf{Ggen}(\lambda) \\ (w, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathbb{G}) \\ \ell \leftarrow_{\$} \mathsf{Primes}(\lambda) \\ u \leftarrow \mathcal{A}_2(\ell, \mathsf{state}) \end{array}\right] = \mathsf{negl}(\lambda)$$

Definition 3 (Low Order assumption [39])**.** *We say that the* low order assumption *holds for* Ggen *if for any PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{l} u^\ell = 1 \\ \wedge u \neq 1 \\ \wedge 1 < \ell < 2^{\mathsf{poly}} \end{array} : \begin{array}{l} \mathbb{G} \leftarrow \mathsf{Ggen}(\lambda) \\ (u, \ell) \leftarrow \mathcal{A}(\mathbb{G}) \end{array}\right] = \mathsf{negl}(\lambda)$$

Firstly, we note that the Low Order Assumption holds unconditionally in RSA groups. Secondly, the Low Order Assumption is implied by the more commonly known Adaptive Root assumption, which is defined below. For the reduction we refer to [39]. We also notice that the definition of the Low Order assumption given in [39] is for smaller ℓ, $1 < \ell < 2^\lambda$, which was sufficient for the application in the paper, whereas ours is for any polynomial-size ℓ. We note that the same reduction to the Adaptive Root assumption described in [39] also holds for our definition of the problem.

Definition 4 (Strong Distinct-Prime-Product Root assumption [145])**.** *We say that the* Strong Distinct-Prime-Product Root assumption *holds for* Ggen *if for any PPT adversary $\mathcal{A}$:*

$$\Pr\left[\begin{array}{ll} u^{\prod_{i\in S} e_i} = g & \mathbb{G} \leftarrow \mathsf{Ggen}(\lambda) \\ \wedge \forall i\, e_i \in \mathsf{Primes}(\lambda) & : \; g \leftarrow_{\$} \mathbb{G} \\ \wedge \forall i \neq j, e_i \neq e_j & (u, \{e_i\}_{i\in S}) \leftarrow \mathcal{A}(\mathbb{G}, g) \end{array}\right] = \mathsf{negl}(\lambda)$$

The assumption is implied by the strong RSA assumption over RSA groups.

3.3.2 Concrete Instantiations of Groups of Unknown Order

Two concrete instantiations of a group, $\mathbb{G}$, of hidden order are RSA groups [184] and class groups [49]. We briefly discuss these instantiations below.

3.3.2.1 RSA groups

We say that $N = pq$ is an RSA modulus for some primes p, q, such that $|p| = |q|$. We further say that N is a strong RSA modulus if there are primes p', q' such that $p = 2p' + 1, q = 2q' + 1$. We call $\mathbb{Z}_N^*$ an RSA group, corresponding to an RSA modulus N. With $\phi : \mathbb{N} \to \mathbb{N}$ we denote the Euler's totient function, $\phi(N) := |\mathbb{Z}_N^*|$. In particular for RSA modulus $\phi(N) = (p-1)(q-1)$. An RSA Group generator $N \leftarrow_{\$} \mathsf{GenSRSAmod}(1^\lambda)$ is a probabilistic algorithm that outputs a strong RSA modulus N of bit-length $\ell(\lambda)$ for an appropriate polynomial $\ell(\cdot)$.

For any N we denote by $\mathsf{QR}_N := \{Y : \exists X \in \mathbb{Z}_N^* \text{ such that } Y = X^2 \pmod N\}$, the set of all the quadratic residues modulo N. QR_N is a subgroup (and thus closed under multiplication) of $\mathbb{Z}_N^*$ with order $|\mathsf{QR}_N| = |\mathbb{Z}_N^*|/2$. In particular for a strong RSA modulus $|\mathsf{QR}_N| = \frac{4p'q'}{2} = 2p'q'$.

3.3.2.2 Computational Assumptions in RSA Groups.

The most fundamental assumption for RSA groups is the factoring assumption which states that given an RSA modulus $N \leftarrow \mathsf{GenSRSAmod}(1^\lambda)$ it is hard to compute its factors p and q. We further recall the Discrete Logarithm and strong RSA [18] assumptions:

Definition 5 (DLOG Assumption for RSA groups)**.** *We say that the* Discrete Logarithm *(DLOG) assumption holds for* GenSRSAmod *if for any PPT adversary $\mathcal{A}$:*

$$\Pr\left[G^{x'} = Y \pmod N \; : \; \begin{array}{l} N \leftarrow \mathsf{GenSRSAmod}(1^\lambda) \\ G \leftarrow_{\$} \mathbb{Z}_N^*; x \leftarrow_{\$} \mathbb{Z} \\ Y \leftarrow G^x \pmod N \\ x' \leftarrow \mathcal{A}(\mathbb{Z}_N^*, G, Y) \end{array}\right] = \mathsf{negl}(\lambda)$$

Definition 6 (Strong-RSA Assumption [18])**.** *We say that the* strong RSA assumption *holds for* GenSRSAmod *if for any PPT adversary $\mathcal{A}$:*

$$\Pr\left[\begin{array}{ll} U^e = G & N \leftarrow \mathsf{GenSRSAmod}(1^\lambda) \\ \wedge e \neq 1, -1 & : \; G \leftarrow_{\$} \mathbb{Z}_N^* \\ & (U, e) \leftarrow \mathcal{A}(\mathbb{Z}_N^*, G) \end{array}\right] = \mathsf{negl}(\lambda)$$

If we desire the Adaptive Root Assumption (and the Low-Order Assumption that is implied from it) to hold we have to take a variant of the RSA group, the quotient group $\mathbb{Z}_N^*/\{1,-1\}$ of an RSA group [215]. The reason why we cannot directly use the RSA group is that the order of $-1 \in \mathbb{Z}_N^*$ is known, and thus the adaptive root assumption does not hold. In the quotient group, $\{-1,1\}$ is the identity element; hence, knowing the order of -1 does not help in finding a root for a non-identity element and thus solving the adaptive root assumption.

3.3.2.3 Class Groups

Class groups of imaginary quadratic order were introduced in Cryptography by Buchmann and Williams on 1988 [50]. Class groups are groups of advanced algebraic structures. In general, a class group $\mathbb{G} = \mathsf{Cl}(\Delta)$ of an imaginary quadratic order is the quotient group of fractional ideals by principal ideals of the group $Q(\sqrt{\Delta})$ with ideal multiplication. It is defined by its discriminant Δ which must satisfy $\Delta = 1 \pmod 4$ and $-\Delta$ must be a prime. Note that the Δ can be generated from public coins for a given security parameters λ.

In this thesis we treat Class Groups as black-box objects that instantiate Groups of Unknown oreder, without making use of their concrete representation. The properties that we assume that hold are the aforementioned of Groups of Unknown Order. For a Survey on Cryptography and Class Groups we refer to [49]

3.3.2.4 Other candidates

Recently Dobson et al. [94] initiated the study of another candidate category of groups where it is conjectured that computing the order is computationally hard, that can be used to instatiate cryptographic scheme over groups of unknown order. They are based on Jacobian of Hyperelliptic curves. The proposal is acknowledged by the authors as currently speculative and cannot yet be considered mature to instantiate cryptographic schemes.

3.3.3 Shamir's Trick

Here we recall a Shamir's trick [192], an algorithm over elemets of groups of unknown order that we will extensively use in subsequent chapters of the thesis.

Informally speaking, Shamir's trick is a way to compute an xy-root of a group element g given an x-root and a y-root of it in groups of unknown order, when x and y are co-prime. That is, given $\rho_x = g^{\frac{1}{x}}$, $\rho_y = g^{\frac{1}{y}}$, x and y, one can compute a, b st $ax + by = 1$ using the extended gcd algorithm. Then $g^{\frac{1}{xy}} = g^{\frac{ax+by}{xy}} = g^{\frac{a}{y}+\frac{b}{x}} = \rho_y^a \cdot \rho_x^b$. More formally, we recall the following algorithm:

ShamirTrick(ρ_x, ρ_y, x, y)

if $\rho_x^x \neq \rho_y^y$ **then return** $\perp$

Use the extended Euclidean Algorithm to compute a, b, d s.t. $ax + by = d = \gcd(x, y)$

if $d \neq 1$ **then return** $\perp$

return $\rho_x^b \rho_y^a$

This technique is remarkable because normally computing a root $g^{\frac{1}{xy}}$ (without having $g^{1/x}, g^{1/y}$) is considered a hard problem (see Section 3.3.1).

3.3.4 Generic Group Model for Groups of Unknown Order

The Generic Group Model (GGM) [196, 160] is a model that restricts the power of the adversary. In particular, a 'generic' adversary does not have concrete representations of the elements of the group, and can only use generic group operations, as group additions and inversions. GGM was extended to groups of unknown order by Damgard and Koprowski [87].

This model captures the possible 'algebraic' attacks that an adversary can perform. Security of a cryptorgaphic primitive in the GGM can be seen as the minimum reassurance we can get. Notably, in many cryptographic settings, as for example in some bilinear group and hidden order group instantiations, there is no clear way to maliciously exploit the group representation and generic attacks are the best currently known technique to attack a scheme.

3.4 Commitments

Here we provide a brief background on (plain) cryptographic commitment schemes.

Commitment schemes allow one to commit to a value, or a collection of values (e.g., a vector), in a way that is *binding*—a commitment cannot be opened to two different values—and *hiding*—a commitment leaks nothing about the value it opens to. In our work we also consider commitment schemes that are *succinct*, meaning informally that the commitment size is fixed and shorter than the committed value.

3.4.1 Commitments to singletons

Here is a brief description of the syntax we use in our work: $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ returns a commitment key ck; $\mathsf{Commit}(\mathsf{ck}, x; o) \to c$ produces a commitment c on input a value x and randomness o (which is also the opening).

3.4.1.1 Pedersen Commitments

Pedersen Commitment [174] is undeniably the most popular commitment scheme. It works over groups of prime order, for example–but not necessarily–bilinear groups. The setup outputs the group representation and two generators, $\mathsf{Setup}(1^\lambda) \to \mathsf{ck} = (\mathbb{G}, g, h)$. Then the commitment is $\mathsf{Commit}(\mathsf{ck}, x; o) \to c = g^x h^r$, where $r \leftarrow_\$ \mathbb{Z}_p$ is a freshly sampled randomness. Both the committed message and the randomness should be elements of $\mathbb{Z}_p$. For the opening the verifier is provided with x and r and simply checks if $c = g^x h^r$ holds.

We note that Pedersen Commitment can generalize to commit to multiple values, $x_1, x_2, \ldots, x_n \in \mathbb{Z}_p$, in a similar manner $c = g_1^{x_1} g_2^{x_2} \ldots g_n^{x_n} h^r$, where $g_1, g_2, \ldots, g_n$ are distinct random generators.

3.4.1.2 Pedersen Commitments of Integer values

A straightforwrd variant of this scheme is for committing to integers (instead of field elements); we describe it in Figure 3.1. Since we commit to an integer x whose size is potentially larger than the order of the group, p, we split the integer into several "chunks", of size $\mathsf{ChkSz} \leq p$ specified in the parameters, and then we apply the standard vector-Pedersen on this split representation. We let the setup algorithm take as input a bound B denoting the max integer that we can commit to.

The construction is perfectly hiding, and computationally binding under the discrete logarithm assumption.

$\underline{\mathsf{Setup}(1^\lambda, \mathsf{B} \in \mathbb{N}, \mathsf{ChkSz} \in \mathbb{N}, n \in \mathbb{N}) :}$

$(\mathbb{G}_p, f) \leftarrow \mathsf{GGen}_\mathsf{p}(1^\lambda)$; If $\mathsf{ChkSz} > \mathsf{p}$ then output $\perp$

Let $N := n \cdot \left\lceil \frac{\mathsf{B}}{\mathsf{ChkSz}} \right\rceil$

Sample $g_1, \ldots, g_N, h \leftarrow\!\!\$\, \mathbb{G}_p$

return $\mathsf{ck} := (\mathbb{G}, \mathsf{B}, \mathsf{ChkSz}, n, g_1, \ldots, g_N, h)$

$\underline{\mathsf{Commit}(\mathsf{ck}, \boldsymbol{x} \in \mathbb{Z}^n; r \in \mathbb{Z}_p) :}$

If $\exists i : x_i > \mathsf{B}$ then output $\perp$

Let $\left(x_1^{(i)}, \ldots, x_m^{(i)}\right)$ be the representation of x_i in base ChkSz for $i \in [n]$

$\boldsymbol{y} := \left(x_1^{(1)}, \ldots, x_m^{(1)}, \ldots, x_1^{(n)}, \ldots, x_m^{(n)}\right)$

return $h^r \prod_{i=1}^{N} g_i^{y_i}$

Figure 3.1: Pedersen Commitment for integer values

3.5 Set Commiments - Accumulators

Cryptographic Accumulators [29, 18] are primitives that allow one to commit to a set $S = \{x_1, \ldots, x_n\}$ that they can later provide a short proof W of membership for any element on the set. They can be seen as commitments to set with a fine-grained opening that allows for membership tests.

3.5.1 Accumulators Definitions

Below is the definition of Accumulators, following the definition of [100]. In this thesis we are only concerned with public key accumulators, meaning that after the key generation phase no party has access to the secret key.

Definition 7 (Accumulators)**.** *A static (non-Universal) Accumulator with domain $\mathbb{X}$ is a tuple of four algorithms,* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Witness}, \mathsf{VerWit})$

- $\mathsf{Setup}(1^\lambda, t) \rightarrow \mathsf{pp}$: *is a (probabilistic) algorithm that takes the security parameter λ and a parameter t for the upper bound of the number of elements to be accumulated. If $t = \infty$ there is no upper bound. Returns some public parameters* pp.
- $\mathsf{Accum}(\mathsf{pp}, S) \rightarrow \mathsf{acc}$: *takes the public parameters and a set S and in case $S \subseteq \mathbb{X}$ outputs the accumulated value* acc *and some auxiliary information* aux*. If $S \nsubseteq \mathbb{X}$ outputs $\perp$.*
- $\mathsf{WitGen}(\mathsf{p}, x, S) \rightarrow W$: *takes the public parameters* pp*, the value x and the set S and outputs either a witness W of $x \in S$ or $\perp$ if $x \notin S$.*
- $\mathsf{Ver}(\mathsf{pp}, \mathsf{acc}, x, w) \rightarrow b$: *takes the public parameters* pp*, the accumulation value* acc*, a value x and a witness w and outputs 1 if W is a witness of $x \in S$ and 0 otherwise.*

Further, we give the definition of Dynamic Accumulators, a notion that was introduced by Camenisch and Lysyanskaya [57]. Dynamic Accumulators are Accumulators that additionally provide the ability to update the accumulated value and the witnesses when the set is updated, either on addition of a new element or on deletion.

Definition 8 (Dynamic Accumulators)**.** *A Dynamic Accumulator with domain $\mathbb{X}$ is a static Accumulator that additionally provides three algorithms* $(\mathsf{Add}, \mathsf{Delete}, \mathsf{WitUpdate})$.

- $\mathsf{Add}(\mathsf{pp}, \mathsf{acc}, y, S) \to (\mathsf{acc}', S')$: *takes the public parameters* pp*, the accumulated value* acc*, the value to be added to the set y and the set S. If $y \notin S \wedge y \in \mathbb{X}$ outputs the new accumulation value* acc' *for the corresponding new set $S' = S \cup \{y\}$ and the new set S'. In case $y \in S$ or $y \notin \mathbb{X}$ outputs $\perp$.*
- $\mathsf{Del}(\mathsf{pp}, \mathsf{acc}, y, S) \to (\mathsf{acc}', S')$: *takes the public parameters* pp*, the accumulated value* acc*, the value to be deleted from the set y and the set S. If $y \in S \wedge y \in \mathbb{X}$ outputs the new accumulation value* acc' *for the corresponding new set $S' = S \setminus \{y\}$ and the new set S'. In case $y \notin \mathcal{X}$ or $y \notin \mathbb{X}$ outputs $\perp$.*
- $\mathsf{WitUpdate}(\mathsf{pp}, W, y, S) \to W'$: *takes the public parameters* pp*, a witness W to be updated, the value y that was either added or deleted from S and the set S. In case $x \in S'$ outputs the updated witness W', otherwise outputs $\perp$.*

Typically, it is required that update algorithms, Add and Del are more efficient than recomputing the accumulation value from scratch with Accum.

Correctness. For every $t = \mathsf{poly}(\lambda)$ and $|S| \leq t$:

$$Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, t); \\ \mathsf{acc} \leftarrow \mathsf{Accum}(\mathsf{pp}, S); \quad : \mathsf{Ver}(\mathsf{pp}, \mathsf{acc}, x, S) \\ W \leftarrow \mathsf{WitGen}(\mathsf{pp}, S, x) \end{array}\right] = 1$$

Soundness. A cryptographic accumulator is sound if for all $t = \mathsf{poly}(\lambda)$ and for all PPT adversaries $\mathcal{A}$ there is a negligible function $\mathsf{negl}(\cdot)$ such that:

$$Pr\left[\begin{array}{rl} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, t); \\ (y^*, W^*, S^*) \leftarrow & \mathcal{A}(1^\lambda, \mathsf{pp}); \qquad : \mathsf{Ver}(\mathsf{pp}, \mathsf{acc}^*, y^*, W^*) = 1 \wedge y^* \in S^* \\ & \mathsf{acc}^* \leftarrow \mathsf{Accum}(\mathsf{pp}, S^*) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Succinctness Furthermore we require that both acc and W are short ($\mathsf{polylog}(n)$) in the size of the set $|S| := n$ and that verification takes time $\mathsf{polylog}(n)$.

3.5.2 Dynamic RSA Accumulators

Below we give the formal description of Dynamic Strong RSA Accumulators [29, 18, 57]. In Section 4.4.1 they are informally recalled.

The domain of RSA accumulators is $\mathbb{X} = \mathbb{P}$.

- $\mathsf{Setup}(1^\lambda, \infty) \to \mathsf{pp}$: samples an RSA modulus $(N, (q_1, q_2)) \leftarrow \mathsf{GenSRSAmod}(1^\lambda)$ and a random group element $F \leftarrow_\$ \mathbb{Z}_N^*$ and computes a quadratic residue $G \leftarrow F^2 \pmod N$.

  Return $\mathsf{pp} \leftarrow (N, G)$

- $\mathsf{Accum}(\mathsf{pp}, S) \to (\mathsf{acc}, S)$: parses $\mathsf{pp} := (N, G)$. If $S \nsubseteq \mathbb{P}$ return $\perp$, otherwise computes $\mathsf{prod}_S := \prod_{x_i \in \mathcal{X}} x_i$ and

  Return $(\mathsf{acc}, S) \leftarrow (G^{\mathsf{prod}_\mathcal{X}} \mod N, S)$
- $\mathsf{WitGen}(\mathsf{ek}, x, \mathsf{aux}) \to W$ parses $\mathsf{pp} := (N, G)$ and computes $\mathsf{prod}_{S\setminus\{x\}} := \prod_{x_i \in \mathcal{X}\setminus\{x\}} x_i$

  Return $W \leftarrow G^{\mathsf{prod}_{\mathcal{X}\setminus\{x\}}} \mod N$
- $\mathsf{Ver}(\mathsf{pp}, \mathsf{acc}, x, W \to b$ parses $\mathsf{pp} := (N, G)$

  Return $b \leftarrow (W^x = \mathsf{acc} \pmod N)$

3.5.2.1 Security of strong RSA Accumulator and Batch-Verification

Security of the above Accumulator comes directly from the strong RSA assumption [18]. What is more interesting is that the same scheme allows for many memberships to be verified at the same time, what is called batch-verification. That is, given $x_1, \ldots, x_m \subseteq \mathbb{P}$ one can compute a batch-witness $W = G^{\mathsf{prod}_{S\setminus\{x_1,\ldots,x_m\}}}$ and the verification will be $b \leftarrow (W^{x_1 \ldots x_m} = \mathsf{acc})$. Again the security of the batch-verification comes from strong RSA assumption and it allows us argue that for any W, x if $W^x = \mathsf{acc} := G^{\mathsf{prod}_S}$ then $x \in \mathsf{prod}_S$, meaning that x is a product of primes of the set S.

3.5.2.2 RSA Accumulators for general Groups of Unkown Order

We note that RSA accumulators work generically over any Group of Unknown Order where the 2-strong RSA Assumption holds [151, 40]. In Figure 3.2 we summarize the RSA accumulator's description over a Groups of Unknown Order $\mathbb{G}$. We include the batching property in the algorithms' description.

$\underline{\mathsf{Setup}(1^{\lambda,\infty})}$:
$(\mathbb{G}, g) \leftarrow \mathsf{GGen}(1^\lambda)$
return $\mathsf{pp} := (\mathbb{G}, g)$

$\underline{\mathsf{Accum}(\mathsf{pp}, S)}$:
$\mathsf{prod} \leftarrow \prod_{x \in S} x$
return $\mathsf{acc} := g^{\mathsf{prod}}$

$\underline{\mathsf{Add}(\mathsf{pp}, \mathsf{acc}, S')}$:
$\mathsf{prod}' \leftarrow \prod x \in S' x$
return $\mathsf{acc}' := \mathsf{acc}^{\mathsf{prod}'}$

$\underline{\mathsf{WitGen}(\mathsf{pp}, S, X)}$:
$\mathsf{prod} \leftarrow \prod_{x \in S} x$, $\mathsf{prod}_X \leftarrow \prod_{x \in X} x$
return $W := g^{\mathsf{prod}/\mathsf{prod}_X}$

$\underline{\mathsf{Ver}(\mathsf{pp}, \mathsf{acc}, X, W)}$:
$\mathsf{prd}_X \leftarrow \prod_{x \in X} x$
Accept iff $W^{\mathsf{prod}_X} = \mathsf{acc}$

Figure 3.2: The RSA Accumulator over groups of unknown order.

3.6 Vector Commitments

A Vector commitment (henceforth VC) scheme [150, 66] is a cryptgraphic primitive that allows a party to compute a commitment to a large vector $\boldsymbol{v}$ of ordered elements and later to locally open a specific position v_i. A VC guarantees that it is hard to open a commitment to two distinct values at the same position – what is called "position binding". The non-triviality of VCs that distinguishes them from plain commitments(Section 3.4) is that they have short (i.e.,

polylogarithmic in the size of the vector $|\boldsymbol{v}|$) commitment and openings-size. We call the latter property "Succinctness".

3.6.1 Vector Commitments

First we recall the basic Vector Commitment syntax and properties [150, 66].

Definition 9 (Vector Commitment [66])**.** *A Vector Commitment (VC) scheme* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *consists of the following algorithms:*

- $\mathsf{Setup}(1^\lambda, n) \to \mathsf{crs}$: *on input the security parameter* λ *and an integer* n *expressing the length of the vectors to be committed, returns the common reference string* crs*.*
- $\mathsf{Com}(\mathsf{crs}, \boldsymbol{v}) \to (C, \mathsf{aux})$: *on input a common reference string* crs *and a vector* $\boldsymbol{v}$*, returns a commitment* C*.*
- $\mathsf{Open}(\mathsf{crs}, \mathsf{aux}, i) \to \Lambda$: *on input an auxiliary information as produced by* Com *and a position* $i \in [n]$*, returns an opening proof* Λ*.*
- $\mathsf{Ver}(\mathsf{crs}, C, \Lambda, i, v) \to b$: *on input a commitment* C*, returns a bit* $b \in \{0; 1\}$ *to check whether* Λ *is a valid opening of* C *to* v *at position* i*.*

Correctness. VC *is* perfectly correct *if for any vector* $\boldsymbol{v}$*:*

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, C, \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, i), i, v_i)) = 1 : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \boldsymbol{v}) \end{array}\right] = 1$$

Position binding. VC *satisfies position binding if for any PPT* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{crs}, C, \Lambda, i, v)) = 1 \\ \wedge\, \mathsf{Ver}(\mathsf{crs}, C, \Lambda, i, v')) = 1 \\ \wedge\, v \neq v' \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (C, i, v, \Lambda, v', \Lambda') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda)$$

Succinctness. VC *is* succinct *if for any* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$*, any vector* $\boldsymbol{v}$*, any* $(C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \boldsymbol{v})$*, any* $i \in [n]$ *and* $\Lambda \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, i)$*, the bitsize of* C *and* Λ *is polylogarithmic in* n*,* i.e.*, is bounded by a fixed polynomial* $p(\lambda, \log n)$*.*

We will also use the notion of *updatable* vector commitments [66], which informally provides the functionality that, given a commitment C and opening Λ corresponding to a vector $\boldsymbol{v}$, one can update them into values C' and Λ' corresponding to a vector $\boldsymbol{v}'$. Notably, this update should be efficient, i.e., in time proportional to the number of different positions in $\boldsymbol{v}$ and $\boldsymbol{v}'$, and thus faster than recomputing them from scratch. More formally:

Definition 10 (Updatable VCs [66])**.** *A vector commitment scheme* VC *is updatable if there are two algorithms* $(\mathsf{ComUpdate}, \mathsf{ProofUpdate})$ *such that:*

- $\mathsf{ComUpdate}(\mathsf{crs}, C, i, v, v') \to C'$: *on input a commitment* C*, a position* i *and two values* v*,* v'*, outputs an updated commitment* C'*.*
- $\mathsf{ProofUpdate}(\mathsf{crs}, \Lambda, i, v, v') \to \Lambda'$: *on input an opening proof* Λ *(for some position* j*), a position* i *and two values* v*,* v'*, returns an updated opening* Λ'*.*

Correctness. *An updatable VC is perfectly correct if for honestly generated* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$*, any vector* $\boldsymbol{v}$*, initial commitment* $(C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \boldsymbol{v})$*, position* $i \in [n]$*,* $\Lambda \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, i)$*, and any sequence of valid updates* $\{(i_k, v_{i_k}, v'_{i_k})\}_{k\in[m]}$ *that result into a vector* $\boldsymbol{v}^*$*, commitment* C^* *and opening* Λ^**,* $\mathsf{Ver}(\mathsf{crs}, C^*, \Lambda^*, i, v_i^*)) = 1$ *holds with probability 1.*

Efficiency. *An updatable VC is* efficient *if its algorithms* ComUpdate *and* ProofUpdate *run in polylogarithmic time given polylogarithmic inputs.*

3.6.2 Vector Commitments with Subvector Openings

A generalization of vector commitments proposed by Lai and Malavolta [145] is Vector Commitments with subvector openings,[12] in which one can open the commitment to an ordered collection of multiple positions with a short proof. In this section we recall this generalization of vector commitments with subvector openings (that for brevity we call SVC). It is easy to see that the original notion of Catalano and Fiore [66] is a special case when the opened subvector includes one position only.

We begin by recalling the notion of subvectors from [145].

Definition 11 (Subvectors [145])**.** *Let* $\mathcal{M}$ *be a set,* $n \in \mathbb{N}$ *be a positive integer and* $I = \{i_1, \ldots, i_{|I|}\} \subseteq [n]$ *be an ordered index set. For a vector* $\boldsymbol{v} \in \mathcal{M}^n$*, the* I*-subvector of* $\boldsymbol{v}$ *is* $\boldsymbol{v}_I := (v_{i_1}, \ldots, v_{i_{|I|}})$*.*

Let $I, J \subseteq [n]$ be two sets, and let $\boldsymbol{v}_I, \boldsymbol{v}_J$ be two subvectors of some vector $\boldsymbol{v} \in \mathcal{M}^n$. The *ordered union* of $\boldsymbol{v}_I$ and $\boldsymbol{v}_J$ is the subvector $\boldsymbol{v}_{I\cup J} := (v_{k_1}, \ldots, v_{k_m})$, where $I \cup J = \{k_1, \ldots, k_m\}$ is the ordered sets union of I and J.

Definition 12 (Vector Commitments with Subvector Openings)**.** *A vector commitment scheme with subvector openings (SVC) is a tuple of algorithms* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *that work as follows and satisfy* correctness, position binding *and* conciseness *defined below.*

- $\mathsf{Setup}(1^\lambda, \mathcal{M}) \to \mathsf{pp}$: *Given the security parameter* λ*, and description of a message space* $\mathcal{M}$ *for the vector components, the probabilistic setup algorithm outputs a common reference string* pp*.*
- $\mathsf{Com}(\mathsf{pp}, \boldsymbol{v}) \to (C, \mathsf{aux})$: *On input* pp *and a vector* $\boldsymbol{v} \in \mathcal{M}^n$*, the committing algorithm outputs a commitment* C *and an auxiliary information* aux*.*
- $\mathsf{Open}(\mathsf{pp}, I, \boldsymbol{y}, \mathsf{aux}) \to \pi_I$: *On input the CRS* pp*, a vector* $\boldsymbol{y} \in \mathcal{M}^m$*, an ordered index set* $I \subset \mathbb{N}$ *and auxiliary information* aux*, the opening algorithm outputs a proof* π_I *that* $\boldsymbol{y}$ *is the* I*-subvector of the committed message.*
- $\mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}, \pi_I) \to b \in \{0, 1\}$: *On input the CRS* pp*, a commitment* C*, an ordered set of indices* $I \subset \mathbb{N}$*, a vector* $\boldsymbol{y} \in \mathcal{M}^m$ *and a proof* π_I*, the verification algorithm accepts (i.e., it outputs 1) only if* π_I *is a valid proof that* C *was created to a vector* $\boldsymbol{v} = (v_1, \ldots, v_n)$ *such that* $\boldsymbol{y} = \boldsymbol{v}_I$*.*

[12] This is also called VCs with batchable openings in an independent work by Boneh et al. [40].

Correctness. *A SVC scheme* VC *is (perfectly)* correct *if for all $\lambda \in \mathbb{N}$, any vector length n any ordered set of indices $I \subseteq [n]$, and any $\boldsymbol{v} \in \mathcal{M}^n$, we have:*

$$\Pr\left[\mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{v}_I, \pi_I) = 1 \quad : \quad \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M}) \\ (C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, \boldsymbol{v}) \\ \pi_I \leftarrow \mathsf{Open}(\mathsf{pp}, I, \boldsymbol{v}_I, \mathsf{aux}) \end{array}\right] = 1$$

Position Binding. *A SVC scheme* VC *satisfies position binding if for all PPT adversaries $\mathcal{A}$ we have:*

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}, \pi) = 1 \\ \wedge\ \boldsymbol{y} \neq \boldsymbol{y}' \wedge \\ \mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}', \pi') = 1 \end{array} : \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M}) \\ (C, I, \boldsymbol{y}, \pi, \boldsymbol{y}', \pi') \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \in \mathsf{negl}(\lambda)$$

Conciseness. *A vector commitment is* concise *if there is a fixed polynomial $p(\lambda)$ in the security parameter such that the size of the commitment C and the outputs of* Open *are both bounded by $p(\lambda)$, i.e., they are independent of n.*

3.6.3 Functional Commitments

Functional commitments (FC), introduced by Libert, Ramanna and Yung [149], allow a sender to commit to a vector $\boldsymbol{v}$ and then to open the commitment to a function $y = f(\boldsymbol{v})$. As in vector commitments, what makes this primitive non-trivial is a succinctness property, which requires commitments and openings to be "short", that is constant or logarithmic in the length of $\boldsymbol{v}$. In our work we use a slight generalization of the FC notion of [149] considering *universal specializable public parameters*. This is a model, akin to the universal CRS of [127], where Setup creates length-independent public parameters pp, which one can later specialize to a specific length n by using a deterministic algorithm Specialize.

Definition 13 (Functional Commitments)**.** *A functional commitment scheme for a class of functions $\mathcal{F}$ is a tuple of algorithms* $\mathsf{FC} = (\mathsf{Setup}, \mathsf{Specialize}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *with the following syntax and that satisfies correctness, succinctness, and function binding.*

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$: *given the security parameter λ, outputs public parameters* pp*, which contain the description of a domain $\mathcal{D}$ and a universal class of functions $\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}$, where $\mathcal{F}_n$ is a class of n-input functions $\{f : \mathcal{D}^n \rightarrow \mathcal{R}\}$.*
- $\mathsf{Specialize}(\mathsf{pp}, \mathcal{F}_n) \rightarrow \mathsf{pp}_n$: *given public parameters* pp *and a description of the function class $\mathcal{F}_n$, outputs specialized parameters* pp_n*.*
- $\mathsf{Com}(\mathsf{pp}_n, \boldsymbol{v}) \rightarrow C$: *on input a vector $\boldsymbol{v} \in \mathcal{D}^n$ outputs a commitment C.*
- $\mathsf{Open}(\mathsf{pp}_n, \boldsymbol{v}, f) \rightarrow \Lambda$: *on input a vector $\boldsymbol{v} \in \mathcal{D}^n$ and an admissible function $f \in \mathcal{F}_n$, outputs an opening Λ.*
- $\mathsf{Ver}(\mathsf{pp}_n, C, f, y, \Lambda) \rightarrow b \in \{0, 1\}$: *on input a commitment C, a function $f \in \mathcal{F}_n$, a value $y \in \mathcal{R}$, and an opening Λ, accepts ($b = 1$) or rejects ($b = 0$).*

Correctness. FC *is correct if, for any public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, any length* $n \in \mathbb{N}$ *and specialized* $\mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$*, any vector* $\boldsymbol{v} \in \mathcal{D}^n$ *and any admissible function* $f \in \mathcal{F}_n$*, it holds*

$$\mathsf{Ver}(\mathsf{pp}_n, \mathsf{Com}(\mathsf{pp}_n, \boldsymbol{v}), f, f(\boldsymbol{v}), \mathsf{Open}(\mathsf{pp}_n, \boldsymbol{v}, f)) = 1$$

Succinctness. FC *is* succinct *if there exists a fixed polynomial* $p(\cdot)$ *such that for any* $n = \mathsf{poly}$*, commitments and openings generated in the scheme have size at most* $p(\lambda, \log n)$.

Function binding. *For any PPT adversary* $\mathcal{A}$ *and any* $n = \mathsf{poly}$*, we have*

$$\Pr\begin{bmatrix}\mathsf{Ver}(\mathsf{pp}_n, C, f, y, \Lambda) = 1 & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \wedge\ y \neq y' \wedge & : (C, f, y, \Lambda, y', \Lambda') \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathsf{Ver}(\mathsf{pp}_n, C, f, y', \Lambda') = 1 & \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, \mathcal{F}_n)\end{bmatrix} = \mathsf{negl}(\lambda)$$

Remark 1. *The* Specialize *algorithm is deterministically computed from* pp *and* $\mathcal{F}_n$*. For this reason, it suffices for Function Binding that the adversary* $\mathcal{A}$ *takes as input* pp *(instead of* pp_n*).*

Remark 2 (Preprocessing-based verification)**.** *Our subsequent FC constructions (Chapter 7) enjoy a preprocessing model of verification, similar to that of preprocessing SNARKs [127]. This means that one, given* pp_n *and a function* f*, can generate a verification key* vk_f *and the latter can be later used to verify any opening for* f*. In particular, while the cost of computing* vk_f *can depend on the complexity of the function, e.g., it is* $O(n)$ *for a linear function with* n *coefficients, the subsequent cost of verifying openings using* vk_f *depends only on the succinctness of the scheme, e.g., it is a fixed* $p(\lambda)$.

3.6.4 Definition of Key-Value Map Commitments

Here we recall the notion of Key-Value Map Commitments [40, 3] and give a formal definition. Key-Value maps are a generization of vectors where a value is associated with a key instead of a position, the main difference being that keys can be arbitrary elements of a (typically) exponentially large space. While in a vector the values are associated with subsequent indices. A Key-Value Map Commitment (henceforth KVC) works analoguesly to a VC but commiting to (and opening) a Key-Value Map. Below is the formal syntax and properties.

Given a key-value map $\mathcal{M}$, we write $(\mathsf{k}, \epsilon) \in \mathcal{M}$ to denote that $\mathcal{M}$ does not contain the key k.

Definition 14 (Key-Value Map Commitment)**.** *A Key-Value Map Commitment* $\mathsf{KVM} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *consists of the following algorithms:*

- $\mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V}) \rightarrow \mathsf{crs}$*: on input the security parameter* λ*, an upper bound* n *on the cardinality of the key-value maps to be committed, a key-space* $\mathcal{K}$*, and a value-space* $\mathcal{V}$*, the setup algorithm returns the common reference string* crs.
- $\mathsf{Com}(\mathsf{crs}, \mathcal{M}) \rightarrow (C, \mathsf{aux})$ *: on input a key-value map* $\mathcal{M} = \{(\mathsf{k}_1, \mathsf{v}_1), \ldots, (\mathsf{k}_m, \mathsf{v}_m)\} \subset \mathcal{K} \times \mathcal{V}$*, computes a commitment* C *and auxiliary information* aux.
- $\mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k}) \rightarrow \Lambda$*: on input auxiliary information* aux *as produced by* Com*, and a key* $\mathsf{k} \in \mathcal{K}$*, the opening algorithm returns an opening* Λ.

- $\mathsf{Ver}(\mathsf{crs}, C, \Lambda, (\mathsf{k}, \mathsf{v})) \rightarrow b$: *accepts (*i.e., *outputs* $b \leftarrow 1$*) if* Λ *is a valid opening of the commitment* C *to the key* $\mathsf{k} \in \mathcal{K}$ *and value* $\mathsf{v} \in \mathcal{V} \cup \{\epsilon\}$*, else rejects (*i.e., *outputs* $b \leftarrow 0$*).*

Intuitively, a KVC scheme should be correct in the sense that, for honest execution of the algorithms, an opening to a $(\mathsf{k}, \mathsf{v}) \in \mathcal{M}$ should correctly verify for a commitment to $\mathcal{M}$. While usual definitions for VCs consider perfect correctness, our work aims at also capturing constructions that have a negligible probability of failing correctness. To capture this, we introduce a strong notion called *robust correctness*, which essentially means that the expected correctness condition holds with overwhelming probability even for key-value maps that are adversarially chosen after seeing the public parameters. We note that such definition is strictly stronger than a 'classical' correctness definition that measures the probability over any choice of input but over the random and independent choice of the public parameters.

Definition 15 (Robust Correctness)**.** KVM *is* robust *if for any PPT* $\mathcal{A}$ *the following probability is overwhelming in* λ*:*

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, C, \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k}), (\mathsf{k}, \mathsf{v})) = 1 : \begin{array}{r} \mathsf{crs} \leftarrow_{\$} \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V}) \\ (\mathcal{M}, \mathsf{k}, \mathsf{v}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ |\mathcal{M}| \leq n, (\mathsf{k}, \mathsf{v}) \in \mathcal{K} \times \mathcal{V} \cup \{\epsilon\} \\ (C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \mathcal{M}) \end{array}\right]$$

Definition 16 (Key-binding)**.** KVM *is* key-value binding *if for any PPT* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{crs}, C, \Lambda, (\mathsf{k}, \mathsf{v})) = 1 \\ \wedge\, \mathsf{Ver}(\mathsf{crs}, C, \Lambda, (\mathsf{k}, \mathsf{v}')) = 1 \\ \wedge\, \mathsf{v} \neq \mathsf{v}' \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow_{\$} \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V}) \\ (C, \mathsf{k}, \mathsf{v}, \Lambda, \mathsf{v}', \Lambda') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda)$$

Below we define an efficiency notion for KVCs, which aim to rule out "uninteresting" constructions, e.g., schemes where either commitments or openings have size linear in the size of the map or the key space. More formally,

Definition 17 (Efficient KVC)**.** *A key-value map commitment* KVM *as defined above is* efficient *if for any* $\mathsf{crs} \leftarrow_{\$} \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V})$*, any key-value map* $\mathcal{M} \subset \mathcal{K} \times \mathcal{V}$*, any* $(C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \mathcal{M})$*, any* $\mathsf{k} \in \mathcal{K}$ *and* $\Lambda \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k})$*, the bitsize of* C *and* Λ *is polylogarithmic in* n*, i.e., it is bounded by a fixed polynomial* $p(\lambda, \log n)$*.*

We provide our definitions of updatable Key-Value Map Commitments in Section 8.4.1, along with the corresponding robust correctness and efficiency notions.

3.7 Zero-Knowledge Proofs

Zero-Knolwedge Proofs [122] is an important notion in Cryptography where one can prove a statement without revealing any other information apart from the validity of the statement. They come in numerous variants and constructions. In the section we recall the variants of the notion and the notation that we will be using throughout the thesis.

3.7.1 Relations

First we define formally *what* zero-knowledge proofs are going to be proving.

We describe this through the notion of *Relations*. A relation R is characterized by a predicate P working with inputs $x_1, x_2, \ldots, x_n$. We say that a relation holds for $(x_1, x_2, \ldots, x_n)$ iff $P(x_1, x_2, \ldots, x_n) = 1$. In the context of Zero-Knowledge proofs we typically separate the inputs into two parts: The statement $\mathbb{x}$ and the witness $\mathbb{w}$. We write a relation as:

$$R = \{(\mathbb{x}; \mathbb{w}) : P(\mathbb{x}, \mathbb{w}) = 1\}.$$

Equivalently, sometimes we will write

$$R(\mathbb{x}; \mathbb{w}) = 1 \Leftrightarrow P(\mathbb{x}, \mathbb{w}) = 1$$

to denote the same relation. We use these two equivalent notations interchangeably throughout the thesis.

Typically the relations we are treating are for NP-predicates where verifying the predicate having $\mathbb{x}$ and $\mathbb{w}$ is efficient, while computing $\mathbb{w}$ given $\mathbb{x}$ is (assumed to be) computationally hard. For instance a (conjectured) NP-relation is

$$R_{\mathsf{RSA\text{-}factoring}} = \{N; (p, q)) : N = p \cdot q\}$$

where verifying that p, q are factors of N is easy while computing p and q is (pressumably) hard.

We write $\mathcal{R}_\lambda$ for a family of relation parametrized by the security parameter λ.

3.7.2 Non-Interactive Zero-Knowledge (NIZK)

We recall the definition of zero-knowledge non-interactive arguments of knowledge (NIZKs, for short).

Definition 18 (NIZK)**.** *A NIZK for $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of three algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerProof})$ that work as follows and satisfy the notions of* completeness, knowledge soundness *and (composable)* zero-knowledge *defined below.*

- $\mathsf{Setup}(R) \to (\mathsf{ek}, \mathsf{vk})$ *takes the security parameter λ and a relation $R \in \mathcal{R}_\lambda$, and outputs a common reference string consisting of an evaluation and a verification key.*
- $\mathsf{Prove}(\mathsf{ek}, \mathbb{x}, \mathbb{w}) \to \pi$ *takes an evaluation key for a relation R, a statement $\mathbb{x}$, and a witness $\mathbb{w}$ such that $R(\mathbb{x}, \mathbb{w})$ holds, and returns a proof π.*
- $\mathsf{VerProof}(\mathsf{vk}, \mathbb{x}, \pi) \to b$ *takes a verification key, a statement $\mathbb{x}$, and either accepts ($b = 1$) or rejects ($b = 0$) the proof π.*

Remark 3. *Sometimes, when there is no ambiguity, we will call the output of the setup as 'common reference string' $\mathsf{crs} := (\mathsf{ek}, \mathsf{vk})$ without separating the evaluation and verification keys.*

Completeness. *For any $\lambda \in \mathbb{N}$, $R \in \mathcal{R}_\lambda$ and $(\mathbb{x}, \mathbb{w})$ such that $R(\mathbb{x}, \mathbb{w})$, it holds* $\Pr[(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(R), \pi \leftarrow \mathsf{Prove}(\mathsf{ek}, \mathbb{x}, \mathbb{w}) : \mathsf{VerProof}(\mathsf{vk}, \mathbb{x}, \pi) = 1] = 1$.

Knowledge Soundness. *Let $\mathcal{RG}$ be a relation generator such that $\mathcal{RG}_\lambda \subseteq \mathcal{R}_\lambda$. Π has computational knowledge soundness for $\mathcal{RG}$ and auxiliary input distribution $\mathcal{Z}$, denoted* $\mathsf{KSND}(\mathcal{RG}, \mathcal{Z})$

for brevity, if for every (non-uniform) efficient adversary $\mathcal{A}$ there exists a (non-uniform) efficient extractor $\mathcal{E}$ such that $\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} = 1] = \mathsf{negl}$. *We say that* VC *is knowledge sound if there exists benign $\mathcal{RG}$ and $\mathcal{Z}$ such that* VC *is* $\mathsf{KSND}(\mathcal{RG}, \mathcal{Z})$.

$\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} \to b$

$(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)$; $\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(R)$

$\mathcal{Z} \leftarrow \mathcal{Z}(R, \mathsf{aux}_R, \mathsf{crs})$; $(\mathbb{x}, \pi) \leftarrow \mathcal{A}(R, \mathsf{crs}, \mathsf{aux}_R, \mathcal{Z})$

$\mathbb{w} \leftarrow \mathcal{E}(R, \mathsf{crs}, \mathsf{aux}_R, \mathcal{Z})$; $b = \mathsf{VerProof}(\mathsf{vk}, \mathbb{x}, \pi) \wedge \neg R(\mathbb{x}, \mathbb{w})$

Composable Zero-Knowledge. *A scheme* VC *satisfies composable zero-knowledge for a relation generator $\mathcal{RG}$ if there exists a simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{prv}})$ such that both following conditions hold:*

Keys Indistinguishability *For all adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)\\ \mathsf{crs} \leftarrow \mathsf{Setup}(R)\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R) = 1\end{array}\right] \approx \Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)\\ (\mathsf{crs}, \mathsf{td}_\mathsf{k}) \leftarrow \mathcal{S}_{\mathsf{kg}}(R)\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R) = 1\end{array}\right]$$

Proof Indistinguishability *For all adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*

$$\Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)\\ (\mathsf{crs}, \mathsf{td}_\mathsf{k}) \leftarrow \mathcal{S}_{\mathsf{kg}}(R)\\ (\mathbb{x}, \mathbb{w}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs}, \mathsf{aux}_R) \ : \ R(\mathbb{x}, \mathbb{w})\\ \pi \leftarrow \mathsf{Prove}(\mathsf{ek}, \mathbb{x}, \mathbb{w})\\ \mathcal{A}_2(\mathsf{st}, \pi) = 1\end{array}\right] \approx \Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)\\ (\mathsf{crs}, \mathsf{td}_\mathsf{k}) \leftarrow \mathcal{S}_{\mathsf{kg}}(R)\\ (\mathbb{x}, \mathbb{w}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs}, \mathsf{aux}_R) \ : \ R(\mathbb{x}, \mathbb{w})\\ \pi \leftarrow \mathcal{S}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td}_\mathsf{k}, \mathbb{x})\\ \mathcal{A}_2(\mathsf{st}, \pi) = 1\end{array}\right]$$

Remark 4 (On Knowledge-Soundness)**.** *In the NIZK definition above we use a non black-box notion of extractability. Although this is virtually necessary in the case of zkSNARKs [118], NIZKs can also satisfy stronger (black-box) notions of knowledge-soundness.*

3.7.3 Succinct Non-Interactive Arguments of Knowledge (SNARKs)

Succinct Non-Interactive Arguments of Knowledge (henceforth SNARKs) we call a specific type of NIZKs. First they are 'arguments' (in place of 'proof') of knowledge, meaning that knowledge soundness hold only against polynomially bounded advesaries. Second, their distinguishing property is 'succinctness', defined formally below. We note that SNARKs do not necessarily have 'zero-knoweldge'; in case they do we call them zkSNARKs.

3.7.3.1 SNARKs definition

Definition 19 (zkSNARKs)**.** *A NIZK is called* zero-knowledge succinct non-interactive argument of knowledge *(zkSNARK) if it is a NIZK as per Definition 18 enjoying an additional property,* succinctness*, i.e., if the running time of* VerProof *is* $\mathsf{poly}(\lambda + |\mathbb{x}| + \log |\mathbb{w}|)$ *and the proof size is* $\mathsf{poly}(\lambda + \log |\mathbb{w}|)$.

3.7.3.2 Commit-and-Prove SNARKs (CP-SNARKs)

We use the framework for black-box modular composition of commit-and-prove SNARKs (or CP-SNARKs) in [61]. Informally a CP-SNARK is a SNARK that can efficiently prove properties of inputs committed through some commitment scheme C. In more detail, a CP-SNARK for a relation $R_{\text{inner}}(\mathbb{x}; u, \omega)$ is a SNARK that for a given commitment c can prove knowledge of $\mathbb{w} := (u, \omega, o)$ such that $c = \mathsf{Commit}(u; o)$ and $R_{\text{inner}}(\mathbb{x}; u, \omega)$ holds. We can think of ω as a non-committed part of the witness. In a CP-SNARK, besides the proof, the verifier's inputs are $\mathbb{x}$ and c.

3.7.3.3 Modular SNARKs through CP-SNARKs

We use the following folklore composition of (zero-knowledge) CP-SNARKs (cf. [61, Theorem 3.1]). Fixed a commitment scheme and given two CP-SNARKs $\mathsf{CP}_1, \mathsf{CP}_2$ respectively for two "inner" relations $\tilde{R}_1$ and $\tilde{R}_2$, we can build a (CP) SNARK for their conjunction (for a shared witness u) $\tilde{R}^*(\boxed{c_u}, \mathbb{x}_1, \mathbb{x}_2; \omega_1, \omega_2) = R_1(\boxed{c_u}, \mathbb{x}_1; \omega_1) \wedge R_2(\boxed{c_u}, \mathbb{x}_2; \omega_2)$ like this: the prover commits to u as $c_u \leftarrow \mathsf{Commit}(u, o)$; generates proofs π_1 and π_2 from the respective schemes; outputs combined proof $\pi^* := (c_u, \pi_1, \pi_2)$. The verifier checks each proof over respective inputs $(\mathbb{x}_1, c_u)$ and $(\mathbb{x}_2, c_u)$, with shared commitment c_u.

3.7.4 Interactive Arguments of Knowledge

In Chapters 6 and 7 we will additionally be treating Interactive Arguments of Knowledge (henceforth AoK). An (Interactive) AoK is a slight variant of NIZK where (1) the prover and the verifier interact in order to produce the proof (2) the knowledge-soundness holds only against computationally bounded adversaries. The notions of interactive AoKs and NIZKs are very close; for completeness and ease of presentation we recall the definitions of AoKs, that appear in Chapters 6 and 7, here.

Let $R : \mathcal{X} \times \mathcal{W} \to \{0, 1\}$ be an NP relation for a language $\mathcal{L} = \{x : \exists w \text{ s.t. } R(x, w) = 1\}$. An argument system for R is a triple of algorithms $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ such that: $\mathsf{Setup}(1^\lambda)$ takes as input a security parameter λ and outputs a common reference string crs; the prover $\mathsf{P}(\mathsf{crs}, x, w)$ takes as input the crs, the statement x and witness w; the verifier $\mathsf{V}(\mathsf{crs}, x)$ takes in the crs, the statement x, and after interacting with the prover outputs 0 (reject) or 1 (accept). An execution between the prover and verifier is denoted with $\langle \mathsf{P}(\mathsf{crs}, x, w), \mathsf{V}(\mathsf{crs}, x) \rangle = b$, where $b \in \{0, 1\}$ is the output of the verifier. If V uses only public randomness, we say that the protocol is public coin.

Definition 20 (Completeness)**.** *We say that an argument system* $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *for a relation* $R : \mathcal{X} \times \mathcal{W} \to \{0, 1\}$ *is complete if, for all* $(x, w) \in \mathcal{X} \times \mathcal{W}$ *such that* $R(x, w) = 1$ *we have*

$$\Pr\left[\langle \mathsf{P}(\mathsf{crs}, x, w), \mathsf{V}(\mathsf{crs}, x) \rangle = 1 : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)\right] = 1.$$

Consider an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ modeled as a pair of algorithms such that $\mathcal{A}_0(\mathsf{crs}) \to (x, \mathsf{state})$ (i.e. outputs an instance $x \in \mathcal{X}$ after $\mathsf{crs} \leftarrow \mathsf{Setup}(\lambda)$ is run) and $\mathcal{A}_1(\mathsf{crs}, x, \mathsf{state})$ interacts with a honest verifier. We want an argument of knowledge to satisfy the following properties:

Soundness. We say that an argument $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ is sound if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ we have

$$\Pr\left[\begin{array}{c}\langle\mathcal{A}_1(\mathsf{crs},x,\mathsf{state}),\mathsf{V}(\mathsf{crs},x)\rangle=1\\ \text{and } \nexists w: R(x,w)=1\end{array}\;\middle|\;\begin{array}{c}\mathsf{crs}\leftarrow\mathsf{Setup}(\lambda)\\ (x,\mathsf{state})\leftarrow\mathcal{A}_0(\mathsf{crs})\end{array}\right]\in\mathsf{negl}(\lambda).$$

Knowledge Extractability. We say that $(\mathsf{Setup},\mathsf{P},\mathsf{V})$ is an *argument of knowledge* if for all polynomial time adversaries $\mathcal{A}_1$ there exists an extractor $\mathcal{E}$ running in polynomial time such that, for all adversaries $\mathcal{A}_0$ it holds

$$\Pr\left[\begin{array}{c}\langle\mathcal{A}_1(\mathsf{crs},x,\mathsf{state}),\mathsf{V}(\mathsf{crs},x)\rangle=1\\ \text{and } (x,w')\notin\mathcal{R}\end{array}\;\middle|\;\begin{array}{c}\mathsf{crs}\leftarrow\mathsf{Setup}(\lambda)\\ (x,\mathsf{state})\leftarrow\mathcal{A}_0(\mathsf{crs})\\ w'\leftarrow\mathcal{E}(\mathsf{crs},x,\mathsf{state})\end{array}\right]\in\mathsf{negl}(\lambda).$$

Succinctness. Finally we informally recall the notion of succinct arguments, which requires the communication and verifier's running time in a protocol execution to be independent of the witness length.

Part II

ZERO-KNOWLEDGE PROOFS FOR SET MEMBERSHIP

4 ZERO-KNOWLEDGE PROOFS FOR SET MEMBERSHIP OF SINGLETONS

The results of this chapter appear in a paper under the title "Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular" published at the Financial Cryptography and Data Security 2021 conference [31] and the Designs, Codes and Cryptography Journal [32].

4.1 Technical Contributions

In this chapter we investigate the problem of designing commit-and-prove zero-knowledge systems for set membership and non-membership that can be used in a modular way and *efficiently* composed with other zero-knowledge proof systems for potentially arbitrary relations. Our main results are the following.

First, building upon the view of recent works on composable proofs [2, 61], we define a formal framework for commit-and-prove zkSNARKs (CP-SNARKs) for set (non-)membership. The main application of this framework is a compiler that, given a CP-SNARK $\mathsf{CP}_{\mathsf{mem}}$ for set membership and any other CP-SNARK CP_R for a relation R, yields a CP-SNARK CP for the composed relation "$u \in S \wedge \exists \omega : R(u, \omega)$". As a further technical contribution, our framework extends the one in [61] in order to work with commitments from multiple schemes (including set commitments, e.g., accumulators).

Second, we propose new efficient constructions of CP-SNARKs for set membership and non-membership, in which elements of the accumulated set can be committed with a Pedersen commitment in a prime order group $\mathbb{G}_q$—a setting that, as argued before, is of practical relevance due to the widespread use of these commitments and of proof systems that operate on them. In more detail, we propose: four schemes (two for set membership and two for non-membership) that enjoy constant-size public parameters and are based on RSA accumulators for committing to sets, and a scheme over pairings that has public parameters linear in the size of the set, but where the set can remain hidden.

Finally, we implement our solutions in a software library and experimentally evaluate their performance.

Like the recent works [2] and [61], our work can be seen as showing yet another setting—set membership— where the efficiency of SNARKs can benefit from a modular design.

RSA-based constructions. Our first scheme, a CP-SNARK for set membership based on RSA

accumulators, supports a large domain for the set of accumulated elements, represented by binary strings of a given length η. Our second scheme, also based on RSA accumulators, supports elements that are prime numbers of exactly μ bits (for a given μ). Neither scheme requires an a-priori bound on the cardinality of the set. Both schemes improve the proof-of-knowledge protocol by Camenisch and Lysyanskaya [57]: (i) we can work with a prime order group $\mathbb{G}_q$ of "standard" size, e.g., 256 bits, whereas [57] needs a much larger $\mathbb{G}_q$ (see above). We note that the size of $\mathbb{G}_q$ affects not only the efficiency of the set membership protocol but also the efficiency of any other protocol that needs to interact with commitments to alleged set members; (ii) we can support flexible choices for the size of set elements. For instance, in the second scheme, we could work with primes of about 50 or 80 bits,[13] which in practice captures virtually unbounded sets and can make the accumulator operations 4–$5\times$ faster compared to using ≈ 256-bits primes as in [57].

Our main technical contribution here involves a new way to link a proof of membership for RSA accumulators to a Pedersen commitment in a prime order group, together with a careful analysis showing this can be secure under parameters *not requiring a larger prime order group* (as in [57]). See Section 4.4 for further details.

Pairing-based construction. Our pairing-based scheme for set membership supports set elements in $\mathbb{Z}_q$, where q is the order of bilinear groups, while the sets are arbitrary subsets of $\mathbb{Z}_q$ of cardinality less than a fixed a-priori bound n. This scheme has the disadvantage of having public parameters linear in n, but has other advantages in comparison to previous schemes with a similar limitation (and also in comparison to the RSA-based schemes above). First, the commitment to the set can be hiding and untrusted for the verifier, i.e., the set can be kept hidden and it is not needed to check the opening of the commitment to the set; this makes it composable with proof systems that could for example prove global properties on the set, i.e., that $P(S)$ holds. Second, the scheme works entirely in bilinear groups, i.e., no need of operating over RSA groups. The main technical contribution here is a technique to turn the EDRAX vector commitment [73] into an accumulator admitting efficient zero-knowledge membership proofs.

Extensions to Set Non-Membership. We propose extensions of both our CP-SNARK framework and RSA constructions to deal with proving *set non-membership*, namely proving in zero-knowledge that $u \notin S$ with respect to a commitment $c(u)$ and a committed set S. Our two RSA-based schemes for non-membership have the same features as the analogous membership schemes mentioned above: the first scheme supports sets whose elements are strings of length η, the second one supports elements that are prime numbers of μ bits, and both work with elements committed using Pedersen in a prime order group and sets committed with RSA accumulators. A byproduct of sharing the same parameters is that we can easily compose the set-membership and non-membership schemes, via our framework, in order to prove statements like $u \in S_1 \wedge u \notin S_2$. Our technical contribution in the design of these schemes is a zero-knowledge protocol for non-membership witnesses of RSA accumulators that is linked to Pedersen commitments in prime order groups.

Transparent Instantiations. We generalize our building blocks for RSA groups to any hidden-order group (appendix 4.8). By instantiating the latter with class groups and by using a transparent CP-NIZK such as Bulletproofs, we obtain variants of our RSA-based schemes with *transparent setup*.

[13] When prime representation is suitable for the application, distinct primes can be generated without a hash fuction (e.g. by using sequential primes).

4.2 Definitions

We provide some additional to Chapter 3 definitions and formalization we will be specifically using in this chapter.

4.2.1 Type-Based Commitments

We recall the notion of Type-Based Commitment schemes introduced by Escala and Groth [98]. In brief, a Type-Based Commitment scheme is a normal commitment scheme with the difference that it allows one to commit to values from different domains. More specifically, the Commit algorithm (therefore the VerCommit algorithm also) depends on the domain of the input, while the commitment key remains the same. For example, as in the original motivation of [98], the committer can use the same scheme and key to commit to elements that may belong to two different groups $\mathbb{G}_1, \mathbb{G}_2$ or a field $\mathbb{Z}_p$. In our work we use type-based commitments. The main benefit of this formalization is that it can unify many commitment algorithms into one scheme. In our case this is useful to formalize the notion of commit-and-prove NIZKs that work with commitments from different groups and schemes.

More formally, a Type-Based Commitment is a tuple of algorithms $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ that works as a Commitment scheme defined above with the difference that Commit and VerCommit algorithms take an extra input t that represent the type of u. All the possible types are included in the type space $\mathcal{T}$[14].

Definition 21. *A type-based commitment scheme for a set of types $\mathcal{T}$ is a tuple of algorithms $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ that work as follows:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ *takes the security parameter and outputs a commitment key* ck*. This key includes $\forall \mathsf{t} \in \mathcal{T}$ descriptions of the input space $\mathcal{D}_\mathsf{t}$, commitment space $\mathcal{C}_\mathsf{t}$ and opening space $\mathcal{O}_\mathsf{t}$.*
- $\mathsf{Commit}(\mathsf{ck}, \mathsf{t}, u) \to (c, o)$ *takes the commitment key* ck*, the type* t *of the input and a value $u \in \mathcal{D}_\mathsf{t}$, and outputs a commitment c and an opening o.*
- $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}, c, u, o) \to b$ *takes as a type* t*, a commitment c, a value u and an opening o, and accepts ($b = 1$) or rejects ($b = 0$).*

Furthermore, the security properties depend on the type, in the sense that binding and hiding should hold with respect to a certain type.

Definition 22. *Let $\mathcal{T}$ be a set of types, and* Com *be a type-based commitment scheme for $\mathcal{T}$. Correctness,* t*-Type Binding and* t*-Type Hiding are defined as follows:*

Correctness. *For all $\lambda \in \mathbb{N}$ and any input $(\mathsf{t}, u) \in (\mathcal{T}, \mathcal{D}_\mathsf{t})$ we have:*

$$\Pr[\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda), (c, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{t}, u) : \mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}, c, u, o) = 1] = 1.$$

t-Type Binding. *Given $\mathsf{t} \in \mathcal{T}$, for every polynomial-time adversary $\mathcal{A}$:*

$$\Pr\begin{bmatrix} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda) \\ (c, u, o, u', o') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{t}) \end{bmatrix} : \begin{matrix} u \neq u' \wedge \ \mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}, c, u, o) = 1 \\ \wedge \ \mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}, c, u', o') = 1 \end{matrix} = \mathsf{negl}$$

[14] Normally $\mathcal{T}$ is finite and includes a small number of type, e.g. $\mathcal{T} = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{Z}_p\}$.

In case Com *is* t*-Type Bidning for all* $\mathsf{t} \in \mathcal{T}$ *we will say that it is Binding.*

t**-Type Hiding.** *Given a* $\mathsf{t} \in \mathcal{T}$*, for* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ *and every pair of values* $u, u' \in \mathcal{D}_\mathsf{t}$*, the following two distributions are statistically close:* $\mathsf{Commit}(\mathsf{ck}, \mathsf{t}, u) \approx \mathsf{Commit}(\mathsf{ck}, \mathsf{t}, u')$.
In case Com *is* t*-Type Hiding for all* $\mathsf{t} \in \mathcal{T}$ *we say it is Hiding.*

Composing Type-Based Commitments. For simplicity we now define an operator that allows to compose type-based commitment schemes in a natural way.

Definition 23. *Let* C *and* C' *be two commitment schemes respectively for (disjoint) sets of types* $\mathcal{T}$ *and* $\mathcal{T}'$*. Then we denote by* $\mathsf{C} \bullet \mathsf{C}'$ *the commitment scheme* $\bar{\mathsf{C}}$ *for* $\mathcal{T} \cup \mathcal{T}'$ *such as:*

- $\bar{\mathsf{C}}.\mathsf{Setup}(\mathsf{secpar}, \mathsf{secpar}') \rightarrow \overline{\mathsf{ck}}$: *compute* $\mathsf{ck} \leftarrow \mathsf{C}.\mathsf{Setup}(\mathsf{secpar})$ *and* $\mathsf{ck}' \leftarrow \mathsf{C}'.\mathsf{Setup}(\mathsf{secpar}')$; $\overline{\mathsf{ck}} := (\mathsf{ck}, \mathsf{ck}')$.
- $\bar{\mathsf{C}}.\mathsf{Commit}(\overline{\mathsf{ck}} := (\mathsf{ck}, \mathsf{ck}'), \mathsf{t}, u)$: *If* $t \in \mathcal{T}$ *then output* $\mathsf{C}.\mathsf{Commit}(\mathsf{ck}, \mathsf{t}, u)$*; otherwise return* $\mathsf{C}'.\mathsf{Commit}(\mathsf{ck}', \mathsf{t}, u)$.
- $\bar{\mathsf{C}}.\mathsf{VerCommit}(\overline{\mathsf{ck}} := (\mathsf{ck}, \mathsf{ck}'), \mathsf{t}, c, u, o)$: *If* $t \in \mathcal{T}$ *then return* $\mathsf{C}.\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}, c, u, o)$*; otherwise return* $\mathsf{C}'.\mathsf{VerCommit}(\mathsf{ck}', \mathsf{t}, c, u, o)$.

The following property of $\bullet$ follows immediately from its definition.

Lemma 1. *Let* C *and* C' *be two commitment schemes with disjoint sets of types. For all types* t *if* C *or* C' *is* t*-hiding (resp.* t*-binding) then* $\mathsf{C} \bullet \mathsf{C}'$ *is* t*-hiding (resp.* t*-binding).*

Remark 5. *We observe that a standard non type-based commitment scheme with input space* $\mathcal{D}$ *induces directly a type-based commitment scheme with the same input space and a type we denote by* $\mathbb{T}[\mathcal{D}]$.

4.2.2 Commit-and-Prove NIZKs with Partial Opening

We now define a variant of commit-and-prove NIZKs with a weaker notion of knowledge-soundness. In particular we consider the case where part of the committed input is not assumed to be extractable (or hidden)[15], i.e., such input is assumed to be opened by the adversary. This models scenarios where we do not require this element to be input of the verification algorithm (the verifier can directly use a digest to it).

The motivation to define and use this notion is twofold. First, in some constructions commitments on sets are compressing but not knowledge-extractable. Second, in many applications this definition is sufficient since the set is public (e.g., the set contain the valid coins).

The definition below is limited to a setting where the adversary opens only one input in this fashion[16]. We will assume, as a convention, that in a scheme with partial opening this special input is always the first committed input of the relation, i.e. the one denoted by u_1 and corresponding to $\mathcal{D}_1$. We note that the commitment to u_1 does not require hiding for zero-knowledge to hold.

[15] This is reminiscent of the soundness notions considered in [103]

[16] We can easily generalize the notion for an adversary opening an arbitrary subset of the committed inputs.

4.2.3 Commit-And-Prove NIZKs

We give the definition of *commit-and-prove NIZKs* (CP-NIZKs) following the definition given in [61, 30] and we extend it to type-based commitments. The main benefit of such extension is that we can formalize CP-NIZKs working with commitments over different domains. In a nutshell, a CP-NIZK is a NIZK that can prove knowledge of $(\mathbb{x}, \mathbb{w})$ such that $R(\mathbb{x}, \mathbb{w})$ holds with respect to a witness $\mathbb{w} = (u, \omega)$ such that u opens a commitment c_u. As done in [61], we explicitly considers the input domain $\mathcal{D}_u$ at a more fine grained-level splitting it over ℓ subdomains. We call them *commitment slots* as each of the $\mathcal{D}_i$-s intuitively corresponds to a committed element[17]. The description of the splitting is assumed part of R's description.

In the remainder of this work we use the following shortcut definition. If C is a type-based commitment scheme over set of types $\mathcal{T}$, we say that a relation R over $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ is $\mathcal{T}$-compatible if for all $j \in [\ell]$ it holds that $\mathbb{T}[\mathcal{D}_j] \in \mathcal{T}$. We say a relation family $\mathcal{R}$ is $\mathcal{T}$-compatible if every R in $\mathcal{R}$ is $\mathcal{T}$-compatible; a relation generator $\mathcal{RG}$ is $\mathcal{T}$-compatible if $\mathsf{range}(\mathcal{RG})$ is $\mathcal{T}$-compatible.

Definition 24 (CP-NIZKs [61])**.** *Let $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of relations R over $\mathcal{D}_\mathbb{x} \times \mathcal{D}_u \times \mathcal{D}_\omega$ such that $\mathcal{D}_u$ splits over ℓ arbitrary domains $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ for some arity parameter $\ell \geq 1$. Let $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ be a commitment scheme (as per Definition 21) over set of types $\mathcal{T}$ such that $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is $\mathcal{T}$-compatible. A commit and prove NIZK for C and $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is a NIZK for a family of relations $\{\mathcal{R}^{\mathsf{C}}_\lambda\}_{\lambda\in\mathbb{N}}$ such that:*

- *every $\mathbf{R} \in \mathcal{R}^{\mathsf{C}}$ is represented by a pair (ck, R) where $\mathsf{ck} \in \mathsf{C.Setup}(1^\lambda)$ and $R \in \mathcal{R}_\lambda$;*
- *$\mathbf{R}$ is over pairs $(\mathbb{x}, \mathbb{w})$ where the statement is $\mathbb{x} := (\mathbb{x}, (c_j)_{j\in[\ell]}) \in \mathcal{D}_\mathbb{x} \times \mathcal{C}^\ell$, the witness is $\mathbb{w} := ((u_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell \times \mathcal{O}^\ell \times \mathcal{D}_\omega$, and the relation $\mathbf{R}$ holds iff*

$$\bigwedge\nolimits_{j\in[\ell]} \mathsf{VerCommit}(\mathsf{ck}, \mathbb{T}[\mathcal{D}_j], c_j, u_j, o_j) = 1 \wedge R(\mathbb{x}, (u_j)_{j\in[\ell]}, \omega) = 1$$

We denote knowledge soundness of a CP-NIZK for commitment scheme C and relation and auxiliary input generators $\mathcal{RG}$ and $\mathcal{Z}$ as $\mathsf{CP\text{-}KSND}(\mathsf{C}, \mathcal{RG}, \mathcal{Z})$.

We denote a CP-NIZK as a tuple of algorithms $\mathsf{CP} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerProof})$. For ease of exposition, in our constructions we adopt the following explicit syntax for CP's algorithms.

- $\mathsf{Setup}(\mathsf{ck}, R) \to \mathsf{crs} := (\mathsf{ek}, \mathsf{vk})$
- $\mathsf{Prove}(\mathsf{ek}, \mathbb{x}, (c_j)_{j\in[\ell]}, (u_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega) \to \pi$
- $\mathsf{VerProof}(\mathsf{vk}, \mathbb{x}, (c_j)_{j\in[\ell]}, \pi) \to b \in \{0, 1\}$

Definition 25 (CP-NIZK with Partial Opening)**.** *A commit and prove NIZK with partial opening for C and $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is a NIZK for a family of relations $\{\mathcal{R}^{\mathsf{C}}_\lambda\}_{\lambda\in\mathbb{N}}$ (defined as in Definition 24)such that the property of knowledge soundness is replaced by* knowledge soundness with partial opening *below.*

Knowledge Soundness with Partial Opening. *Let $\mathcal{RG}$ be a relation generator such that $\mathcal{RG}_\lambda \subseteq \mathcal{R}_\lambda$. VC has knowledge soundness with partial opening for C, $\mathcal{RG}$ and auxiliary input*

[17] Each of the "open" elements in the $\mathcal{D}_i$-s (together with any auxiliary opening information) should also be thought of as the witness to the relation as we require them to be extractable. On the other hand, the commitments themselves are part of the public input.

distribution $\mathcal{Z}$, denoted $\mathsf{CP\text{-}poKSND}(\mathsf{C}, \mathcal{RG}, \mathcal{Z})$ *for brevity, if for every (non-uniform) efficient adversary $\mathcal{A}$ there exists a (non-uniform) efficient extractor $\mathcal{E}$ such that* $\Pr[\mathsf{Game}^{\mathsf{CP\text{-}poKSND}}_{\mathsf{C},\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} = 1] = \mathsf{negl}$. *We say that* VC *is knowledge sound for* C *if there exists benign $\mathcal{RG}$ and $\mathcal{Z}$ such that* VC *is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}, \mathcal{RG}, \mathcal{Z})$[18].

$\mathsf{Game}^{\mathsf{CP\text{-}poKSND}}_{\mathsf{C},\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} \rightarrow b$

$\mathsf{ck} \leftarrow \mathsf{C.Setup}(1^\lambda);\ (R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\ \mathbf{R} := (\mathsf{ck}, R)$

$\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(\mathbf{R})$

$\mathcal{Z} \leftarrow \mathcal{Z}(\mathbf{R}, \mathsf{aux}_R, \mathsf{crs})$

$\big(\mathbb{x}, (c_j)_{j\in[\ell]}, u_1, o_1, \pi\big) \leftarrow \mathcal{A}(\mathbf{R}, \mathsf{crs}, \mathsf{aux}_R, \mathcal{Z})$

$\big((u_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega\big) \leftarrow \mathcal{E}(\mathbf{R}, \mathsf{crs}, \mathsf{aux}_R, \mathcal{Z})$

$b = \mathsf{VerProof}(\mathsf{vk}, \mathbb{x}, (c_j)_{j\in[\ell]}, \pi) \wedge$

$\quad \neg\big(\bigwedge_{j\in[\ell]} \mathsf{C.VerCommit}(\mathsf{ck}, \mathbb{T}[\mathcal{D}_j], c_j, u_j, o_j) = 1 \wedge R(\mathbb{x}, (u_j)_{j\in[\ell]}, \omega) = 1\big)$

Remark 6 (On Weaker ZK in the Context of Partial Opening)**.** *The notion of zero-knowledge for CP-NIZKs with partial opening that is implied by our definition above implies that the simulator does not have access to the opening of the first input (as it is the case in zero-knowledge for CP-NIZKs in general). Since this first commitment is opened, in principle one could also consider and define a weaker notion of zero-knowledge where the simulator has access to the first opened input. We leave it as an open problem to investigate if it can be of any interest.*

Remark 7 (Full Extractability)**.** *If a CP-NIZK has an empty input u_1 opened by the adversary in the game above, then we say that it is* fully extractable. *This roughly corresponds to the notion of knowledge soundness in Definition 18.*

4.2.3.1 Composition Properties of Commit-and-Prove Schemes

In [61] Campanelli et al. show a compiler for composing commit-and-prove schemes that work for the same commitment scheme in order to obtain CP systems for conjunction of relations. In this section we generalize their results to the case of typed relations and type-based commitments. This generalization in particular can model the composition of CP-NIZKs that work with different commitments, as is the case in our constructions for set membership in which one has a commitment to a set and a commitment to an element.

We begin by introducing the following compact notation for an augmented relation generator.

Definition 26 (Augmented Relation Generator)**.** *Let $\mathcal{RG}$ be a relation generator and $\mathcal{F}(1^\lambda)$ an algorithm taking as input a security parameter. Then we denote by $\mathcal{RG}[\mathcal{F}]$ the relation generator returning $(R, (\mathsf{aux}_R, \mathsf{out}_\mathcal{F}))$ where $\mathsf{out}_\mathcal{F} \leftarrow \mathcal{F}(1^\lambda)$ and $(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)$.*

[18] We point out that, although in the game below we make explicit the commitment opening in the relation, this is essentially the same notion of knowledge soundness as in CP-NIZKs (i.e. Definition 18) where the only tweak is that the adversary gives explicitly the first input in the commitment slot. We make commitments explicit hoping for the definition to be clearer. This is, however, in contrast to the definition of CP-NIZKs where the commitment opening is completely abstracted away inside the relation.

The next lemma states that we can (with certain restrictions) trivially extend a CP-NIZK for commitment scheme C to an extended commitment scheme $\mathsf{C} \bullet \mathsf{C}'$.

Lemma 2 (Extending to Commitment Composition)**.** *Let* C, C' *be commitment schemes defined over disjoint type sets* $\mathcal{T}$ *and* $\mathcal{T}'$*. If* CP *is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}, \mathcal{RG}[\mathsf{C.Setup}], \mathcal{Z})$ *for some relation and auxiliary input generators* $\mathcal{RG}, \mathcal{Z}$*. Then* CP *is* $\mathsf{CP\text{-}poKSND}(\mathsf{C} \bullet \mathsf{C}', \mathcal{RG}[\mathsf{C.Setup}], \mathcal{Z})$ *if* $\mathcal{RG}$ *is* $\mathcal{T}$*-compatible.*

We now define relation generators and auxiliary input generators for our composition constructions.

$\underline{\mathsf{Aux}^{\mathcal{RG}}(1^\lambda):}$
$(R_1, \mathsf{aux}_R^{(1)}) \leftarrow \mathcal{RG}_1(1^\lambda)$
$(R_2, \mathsf{aux}_R^{(2)}) \leftarrow \mathcal{RG}_2(1^\lambda)$
return $(R_b, \mathsf{aux}_R^{(b)})_{b\in\{1,2\}}$

$\underline{\mathcal{RG}^*(1^\lambda):}$
$(R_b, \mathsf{aux}_R^{(b)})_{b\in\{1,2\}} \leftarrow \mathsf{Aux}^{\mathcal{RG}}(1^\lambda)$
return $(R^\wedge_{R_1,R_2}, (\mathsf{aux}_R^{(b)})_{b\in\{1,2\}})$

$\underline{\overline{\mathcal{RG}}_b(1^\lambda):}$
$(R_b, \mathsf{aux}_R^{(b)})_{b\in\{1,2\}}$
$\leftarrow \mathsf{Aux}^{\mathcal{RG}}(1^\lambda)$
return $(R_b, \overline{\mathsf{aux}}_R^{(b)}$
$:= (R_{3-b}, (\mathsf{aux}_R^{(b)})_{b\in\{1,2\}}))$

$\underline{\mathsf{Aux}^{\mathcal{Z}}(\mathsf{ck}, (\mathsf{crs}_b, R_b, \mathsf{aux}_R^{(b)})_{b\in\{1,2\}}):}$
$\mathsf{aux}_Z^{(1)} \leftarrow \mathcal{Z}_1(\mathsf{ck}, R_1, \mathsf{crs}_1, \mathsf{aux}_R^{(1)})$
$\mathsf{aux}_Z^{(2)} \leftarrow \mathcal{Z}_2(\mathsf{ck}, R_2, \mathsf{crs}_2, \mathsf{aux}_R^{(2)})$
return $(\mathsf{aux}_Z^{(b)})_{b\in\{1,2\}}$

$\underline{\mathcal{Z}^*((\mathsf{ck}, R^\wedge_{R_1,R_2}), (\mathsf{ek}^*, \mathsf{vk}^*), (\mathsf{aux}_R, \mathsf{aux}'_R)):}$
$(\mathsf{aux}_Z^{(b)})_{b\in\{1,2\}}$
$\leftarrow \mathsf{Aux}^{\mathcal{Z}}(\mathsf{ck}, (\mathsf{crs}_b, R_b, \mathsf{aux}_R^{(b)})_{b\in\{1,2\}})$
return $(\mathsf{aux}_Z^{(b)})_{b\in\{1,2\}}$

$\underline{\overline{\mathcal{Z}}_b(\mathsf{ck}, R_b, \mathsf{crs}_b, \overline{\mathsf{aux}}_R^{(b)}):}$
Parse $\overline{\mathsf{aux}}_R$ as $(R_{3-b}, (\mathsf{aux}_R^{(b)})_{b\in\{1,2\}})$
$\mathsf{crs}_{3-b} \leftarrow \mathsf{CP}_{3-b}.\mathsf{Setup}(\mathsf{ck}, R_{3-b})$
$(\mathsf{aux}_Z^{(b)})_{b\in\{1,2\}} \leftarrow \mathsf{Aux}^{\mathcal{Z}}(\mathsf{ck}, \ldots$
$\ldots, (\mathsf{crs}_b, R_b, \mathsf{aux}_R^{(b)})_{b\in\{1,2\}})$
$\overline{\mathsf{aux}}_Z^{(b)} := (\mathsf{crs}_{3-b}, (\mathsf{aux}_Z^{(b)})_{b\in\{1,2\}})$
return $\overline{\mathsf{aux}}_Z^{(b)}$

Figure 4.1: Relation and Auxiliary Input Generators for AND Composition Construction

The following lemma shows how we can compose CP-NIZKs even when one of them is fully extractable but the other is not. We are interested in the conjunction $R^\wedge_{asym}$ of relations of type $R_1(x_1, (u_0, u_1, u_3), \omega_1)$ and $R_2(x_2, (u_2, u_3), \omega_2)$ where

$$R^\wedge_{asym}(x_1, x_2, (u_0, u_1, u_2, u_3), \omega_1, \omega_2) := R_1(x_1, (u_0, u_1, u_3), \omega_1) \wedge R_2(x_2, (u_2, u_3), \omega_2)$$

Lemma 3 (Composing Conjunctions (with asymmetric extractability)**.** *Let* C *be a computationally binding commitment scheme.If* CP_1 *is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}, \overline{\mathcal{RG}}_1, \overline{\mathcal{Z}}_1)$ *and* CP_2 *is* $\mathsf{KSND}(\mathsf{C}, \overline{\mathcal{RG}}_2, \overline{\mathcal{Z}}_2)$ *(where* $\overline{\mathcal{RG}}_b, \overline{\mathcal{Z}}_b$ *are defined in terms of* $\mathcal{RG}_b, \mathcal{Z}_b$ *in Figure 4.1 for* $b \in \{1, 2\}$*), then the scheme* $\mathsf{CP}^\wedge_{asym}$ *in Figure 4.2 is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}, \mathcal{RG}^*, \mathcal{Z}^*)$ *where* $\mathcal{RG}^*, \mathcal{Z}^*$ *are as defined in Figure 4.1.*

$\underline{\mathsf{CP}^{\wedge}_{asym}.\mathsf{Setup}(\mathsf{pp}, R^{\wedge}_{R_1,R_2}) :}$

$(\mathsf{ek}_1, \mathsf{vk}_1) \leftarrow \mathsf{CP}_1.\mathsf{Setup}(\mathsf{ck}, R_1)$

$(\mathsf{ek}_2, \mathsf{vk}_2) \leftarrow \mathsf{CP}_2.\mathsf{Setup}(\mathsf{ck}, R_2)$

$\mathsf{ek}^* := (\mathsf{ek}_b)_{b\in\{1,2\}}$

$\mathsf{vk}^* := (\mathsf{vk}_b)_{b\in\{1,2\}}$

return $(\mathsf{ek}^*, \mathsf{vk}^*)$

$\underline{\mathsf{CP}^{\wedge}_{asym}.\mathsf{VerProof}(\mathsf{vk}^*, \mathbb{x}_1, \mathbb{x}_2, (c_j)_{j\in[0,3]}, \pi^*) :}$

$b^{(1)}_{\mathsf{ok}} \leftarrow \mathsf{CP}_1.\mathsf{VerProof}(\mathsf{vk}_1, \mathbb{x}_1, (c_0, c_1, c_3), \pi_1)$

$b^{(2)}_{\mathsf{ok}} \leftarrow \mathsf{CP}_2.\mathsf{VerProof}(\mathsf{vk}_2, \mathbb{x}_2, (c_2, c_3), \pi_2)$

return $b^{(1)}_{\mathsf{ok}} \wedge b^{(2)}_{\mathsf{ok}}$

$\underline{\mathsf{CP}^{\wedge}_{asym}.\mathsf{Prove}(\mathsf{ek}^*, \mathbb{x}_1, \mathbb{x}_2, (c_j)_{j\in[0,3]}, (u_j)_{j\in[0,3]}, (o_j)_{j\in[0,3]}, \omega_1, \omega_2) :}$

$\pi_1 \leftarrow \mathsf{CP}_1.\mathsf{Prove}(\mathsf{ek}_1, \mathbb{x}_1, (c_0, c_1, c_3), (u_0, u_1, u_3), (o_0, o_1, o_3), \omega_1)$

$\pi_2 \leftarrow \mathsf{CP}_2.\mathsf{Prove}(\mathsf{ek}_2, \mathbb{x}_2, (c_2, c_3), (u_2, u_3), (o_2, o_3), \omega_2)$

return $\pi^* := (\pi_b)_{b\in\{1,2\}}$

Figure 4.2: CP-NIZK construction for AND composition (asymmetric case)

$\underline{\mathsf{CP}^{\wedge}_{sym}.\mathsf{Setup}(\mathsf{pp}, R^{\wedge}_{R_1,R_2}) :}$

$(\mathsf{ek}_1, \mathsf{vk}_1) \leftarrow \mathsf{CP}_1.\mathsf{Setup}(\mathsf{ck}, R_1)$

$(\mathsf{ek}_2, \mathsf{vk}_2) \leftarrow \mathsf{CP}_2.\mathsf{Setup}(\mathsf{ck}, R_2)$

$\mathsf{ek}^* := (\mathsf{ek}_b)_{b\in\{1,2\}}$

$\mathsf{vk}^* := (\mathsf{vk}_b)_{b\in\{1,2\}}$

return $(\mathsf{ek}^*, \mathsf{vk}^*)$

$\underline{\mathsf{CP}^{\wedge}_{sym}.\mathsf{VerProof}(\mathsf{vk}^*, \mathbb{x}_1, \mathbb{x}_2, (c_j)_{j\in[0,3]}, \pi^*) :}$

$b^{(1)}_{\mathsf{ok}} \leftarrow \mathsf{CP}_1.\mathsf{VerProof}(\mathsf{vk}_1, \mathbb{x}_1, (c_0, c_1, c_3), \pi_1)$

$b^{(2)}_{\mathsf{ok}} \leftarrow \mathsf{CP}_2.\mathsf{VerProof}(\mathsf{vk}_2, \mathbb{x}_2, (c_0, c_2, c_3), \pi_2)$

return $b^{(1)}_{\mathsf{ok}} \wedge b^{(2)}_{\mathsf{ok}}$

$\underline{\mathsf{CP}^{\wedge}_{sym}.\mathsf{Prove}(\mathsf{ek}^*, \mathbb{x}_1, \mathbb{x}_2, (c_j)_{j\in[0,3]}, (u_j)_{j\in[0,3]}, (o_j)_{j\in[0,3]}, \omega_1, \omega_2) :}$

$\pi_1 \leftarrow \mathsf{CP}_1.\mathsf{Prove}(\mathsf{ek}_1, \mathbb{x}_1, (c_0, c_1, c_3), (u_0, u_1, u_3), (o_0, o_1, o_3), \omega_1)$

$\pi_2 \leftarrow \mathsf{CP}_2.\mathsf{Prove}(\mathsf{ek}_2, \mathbb{x}_2, (c_0, c_2, c_3), (u_0, u_2, u_3), (o_0, o_2, o_3), \omega_2)$

return $\pi^* := (\pi_b)_{b\in\{1,2\}}$

Figure 4.3: CP-NIZK construction for AND composition (symmetric case)

The following lemma is a symmetric variant of Lemma 3, i.e. the CP-NIZKs we are composing are both secure over the same commitment scheme and support partial opening, that is they both handle relations with and adversarially open input u_0. This time we are interested in the conjunction $R^{\wedge}_{sym}$ of relations of type $R_1(x_1, (u_0, u_1, u_3), \omega_1)$ and $R_2(x_2, (u_0, u_2, u_3), \omega_2)$ where

$$R^{\wedge}_{sym}(x_1, x_2, (u_0, u_1, u_2, u_3), \omega_1, \omega_2) := R_1(x_1, (u_0, u_1, u_3), \omega_1) \wedge R_2(x_2, (u_0, u_2, u_3), \omega_2)$$

Lemma 4 (Composing Conjunctions (symmetric case))**.** *Let* C *be a (type-based) computationally binding commitment scheme. If* CP_b *is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}, \overline{\mathcal{RG}}_b, \overline{\mathcal{Z}}_b)$ *(where* $\overline{\mathcal{RG}}_b, \overline{\mathcal{Z}}_b$ *are defined in terms of* $\mathcal{RG}_b, \mathcal{Z}_b$ *in Figure 4.1) for* $b \in \{1, 2\}$*, then the scheme* $\mathsf{CP}^{\wedge}_{sym}$ *in Figure 4.3 is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}, \mathcal{RG}^*, \mathcal{Z}^*)$ *where* $\mathcal{RG}^*, \mathcal{Z}^*$ *are as defined in Figure 4.1.*

4.3 CP-SNARKs for Set Membership (and non-Membership)

In this section we discuss a specialization of CP-SNARKs for the specific NP relation that models membership (resp. non-membership) of an element in a set, formally defined below.

Set membership relations. Let $\mathcal{D}_{\mathsf{elm}}$ be some domain for set elements, and let $\mathcal{D}_{\mathsf{set}} \subseteq 2^{\mathcal{D}_{\mathsf{elm}}}$ be a set of possible sets over $\mathcal{D}_u$. We define the set membership relation $R_{\mathsf{mem}} : \mathcal{D}_{\mathsf{elm}} \times \mathcal{D}_{\mathsf{set}}$ as

$$R_{\mathsf{mem}}(S, u) = 1 \Leftrightarrow u \in S$$

This is the fundamental relation that we deal with in the rest of this work.
The non-membership relation $R_{\mathsf{nmem}} : \mathcal{D}_{\mathsf{elm}} \times \mathcal{D}_{\mathsf{set}}$ can be defined analogously as

$$R_{\mathsf{nmem}}(S, u) = 1 \Leftrightarrow u \notin S$$

CP-SNARKs for set membership. Intuitively, a commit-and-prove SNARK for set membership allows one to commit to a set S and to an element u, and then to prove in zero-knowledge that $R_{\mathsf{mem}}(S, u) = 1$. More formally, let $R_{\mathsf{mem}} : \mathcal{D}_{\mathsf{elm}} \times \mathcal{D}_{\mathsf{set}}$ be a set membership relation as defined above where $\mathbb{T}[\mathcal{D}_{\mathsf{elm}}] = \mathsf{t}_{\mathsf{elm}}$ and $\mathbb{T}[\mathcal{D}_{\mathsf{set}}] = \mathsf{t}_{\mathsf{set}}$, and let $\mathsf{Com}_{\mathsf{S}\cup\mathsf{elm}}$ be a type-based commitment scheme for $\mathcal{T}$ such that $\mathsf{t}_{\mathsf{set}}, \mathsf{t}_{\mathsf{elm}} \in \mathcal{T}$. Basically, $\mathsf{Com}_{\mathsf{S}\cup\mathsf{elm}}$ allows one to either commit an element of $\mathcal{D}_{\mathsf{elm}}$ or to a set of values of $\mathcal{D}_{\mathsf{elm}}$. Then a CP-SNARK for set membership is a CP-SNARK for the family of relations $\{\mathcal{R}_\lambda^{\mathsf{mem}}\}$ and a type-based commitment scheme $\mathsf{Com}_{\mathsf{S}\cup\mathsf{elm}}$. It is deduced from definition 24 that this is a zkSNARK for the relation:

$$\mathbf{R} = (ck, R_{\mathsf{mem}}) \text{ over}$$
$$(\boldsymbol{x}, \boldsymbol{w}) = ((x, c), (u, o, \omega)) := (\,(\,\varnothing\,,\,(c_S, c_u)\,)\,,\,(\,(S, u)\,,\,(o_S, o_u)\,,\,\varnothing\,)\,)$$

such that $\mathbf{R}$ holds iff:

$$R_{\mathsf{mem}}(S, u) = 1 \wedge \mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_{\mathsf{set}}, c_S, S, o_S) = 1 \wedge \mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_{\mathsf{elm}}, c_u, u, o_u) = 1$$

A commit-and-prove version of R_{nmem} can be defined as a natural variant of the relation above.

Notice that for the relation R_{mem} it is relevant for the proof system to be succinct so that proofs can be at most polylogarithmic (or constant) in the the size of the set (that is part of the witness). This is why for set membership we are mostly interested in designing CP-SNARKs.

4.3.1 Proving arbitrary relations involving set (non-)membership.

As discussed in the introduction, a primary motivation of proving set membership in zero-knowledge is to prove additional properties about an alleged set member. In order to make our CP-SNARK for set membership a reusable gadget, we discuss a generic and simple method for composing CP-SNARKs for set membership (with partial opening) with other CP-SNARKs (with full extractability) for arbitrary relations. More formally, let R_{mem} be the set membership relation over pairs $(S, u) \in \mathbb{X} \times \mathcal{D}_u$ as R be an arbitrary relation over pairs (u, ω), then we define as R^* the relation:

$$R^*(S, u, \omega) := R_{\mathsf{mem}}(S, u) \wedge R(u, \omega)$$

The next corollary (direct consequence of Lemmas 2, 3) states we can straightforwardly compose a CP-SNARK for set membership with a CP-SNARK for an arbitrary relation on elements of the set.

Corollary 1 (Extending Relations with Set Membership)**.** *Let* $\mathsf{C}_\mathsf{S}, \mathsf{C}_u$ *be two computationally binding commitment schemes defined over disjoint type sets* $\mathcal{T}_\mathsf{S}$ *and* $\mathcal{T}_u$*. Let* $\mathsf{CP}_{\mathsf{mem}}, \mathsf{CP}_u$ *be two CP-SNARKs and* $R_{\mathsf{mem}}, \mathcal{RG}_u$ *(resp.* $\mathcal{Z}_{\mathsf{mem}}, \mathcal{Z}_u$*) be two relation (resp. auxiliary input) generators. If* $\mathsf{CP}_{\mathsf{mem}}$ *is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}_\mathsf{S} \bullet \mathsf{C}_u, R_{\mathsf{mem}}, \mathcal{Z}_{\mathsf{mem}})$ *and* CP_u *is* $\mathsf{KSND}(\mathsf{C}_u, \mathcal{RG}_u, \mathcal{Z}_u)$ *then there exists a* CP^* *that is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}_\mathsf{S} \bullet \mathsf{C}_u, \mathcal{RG}^*, \mathcal{Z}^*)$ *where* $\mathcal{RG}^*, \mathcal{Z}^*$ *are as defined in Figure 4.1.*

In a similar fashion, we can combine an arbitrary relation R with the relation for non-membership obtaining relation $\bar{R}^*$ defined as:

$$\bar{R}^*(S, u, \omega) := R_{\mathsf{nmem}}(S, u) \wedge R(u, \omega)$$

The next corollary states we can straightforwardly compose a CP-SNARK for set non-membership with a CP-SNARK for an arbitrary relation on elements in the universe of the set.

Corollary 2 (Extending Relations with Set non-Membership)**.** *Let* $\mathsf{C}_\mathsf{S}, \mathsf{C}_u$ *be two computationally binding commitment schemes defined over disjoint type sets* $\mathcal{T}_\mathsf{S}$ *and* $\mathcal{T}_u$*. Let* $\mathsf{CP}_{\mathsf{nmem}}, \mathsf{CP}_u$ *be two CP-SNARKs and* $R_{\mathsf{nmem}}, \mathcal{RG}_u$ *(resp.* $\mathcal{Z}_{\mathsf{nmem}}, \mathcal{Z}_u$*) be two relation (resp. auxiliary input) generators. If* $\mathsf{CP}_{\mathsf{nmem}}$ *is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}_\mathsf{S} \bullet \mathsf{C}_u, R_{\mathsf{nmem}}, \mathcal{Z}_{\mathsf{nmem}})$ *and* CP_u *is* $\mathsf{KSND}(\mathsf{C}_u, \mathcal{RG}_u, \mathcal{Z}_u)$ *then there exists a* CP^* *that is* $\mathsf{CP\text{-}poKSND}(\mathsf{C}_\mathsf{S} \bullet \mathsf{C}_u, \mathcal{RG}^*, \mathcal{Z}^*)$ *where* $\mathcal{RG}^*, \mathcal{Z}^*$ *are as defined in Figure 4.1.*

CP-SNARKs for set membership from accumulators with proofs of knowledge. As discussed in the introduction, CP-SNARKs for set membership are simply a different lens through which we can approach accumulators that have a protocol for proving in zero-knowledge that a committed value is in the accumulator (i.e., it is in the set succinctly represented by the accumulator).

4.4 A CP-SNARK for Set Membership with Short Parameters

In this section we describe CP-SNARKs for set membership in which the elements of the sets can be committed using a Pedersen commitment scheme defined in a prime order group, and the sets are committed using an RSA accumulator. The advantage of having elements committed with Pedersen in a prime order group is that our CP-SNARKs can be composed with any other CP-SNARK for Pedersen commitments and relations R that take set elements as inputs. The advantage of committing to sets using RSA accumulators is instead that the public parameters (i.e., the CRS) of the CP-SNARKs presented in this section are *short*, virtually independent of the size of the sets. Since RSA accumulators are not extractable commitments, the CP-SNARKs presented here are secure in a model where the commitment to the set is assumed to be checked at least once, namely they are knowledge-sound with partial opening of the set commitment.

A bit more in detail, we propose two CP-SNARKs. Our first scheme, called $\mathsf{MemCP}_{\mathsf{RSA}}$, works for set elements that are arbitrary strings of length η, i.e., $\mathcal{D}_{\mathsf{elm}} = \{0,1\}^\eta$, and for sets that are any subset of $\mathcal{D}_{\mathsf{elm}}$, i.e., $\mathcal{D}_{\mathsf{set}} = 2^{\mathcal{D}_{\mathsf{elm}}}$. Our second scheme, $\mathsf{MemCP}_{\mathsf{RSAPrm}}$, instead works for set elements that are prime numbers of exactly μ bits, and for sets that are any subset of such prime numbers. This second scheme is a simplified variant of the first one that requires more

structure on the set elements (they must be prime numbers) but in exchange of that offers better efficiency. So it is preferable in those applications that can work with prime representatives.

An High-Level Overview of Our Constructions. We provide the main idea behind our scheme, and to this end we use the simpler scheme $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ in which set elements are prime numbers in $\mathbb{P}(\mu) := \left(2^{\mu-1}, 2^{\mu}\right)$. The commitment to the set $P = \{e_1, \ldots, e_n\}$ is an RSA accumulator [29, 18] that is defined as $\mathsf{Acc} = G^{\prod_{e_i \in P} e_i}$ for a random quadratic residue $G \in \mathsf{QR}_N$. The commitment to a set element e is instead a Pedersen commitment $c_e = g^e h^{r_q}$ in a group $\mathbb{G}_q$ of prime order q, where q is of ν bits and $\mu < \nu$. For public commitments Acc and c_e, our scheme allows to prove in zero-knowledge the knowledge of e committed in c_e such that $e \in P$ and $\mathsf{Acc} = G^{\prod_{e_i \in P} e_i}$. A public coin protocol for this problem was proposed by Camenisch and Lysyanskaya [57]. Their protocol however requires various restrictions. For instance, the accumulator must work with at least 2λ-bit long primes, which slows down accumulation time, and the prime order group must be more than 4λ-bits (e.g., of 512 bits), which is undesirable for efficiency reasons, especially if this prime order group is used to instantiate more proof systems to create other proofs about the committed element. In our scheme the goal is instead to keep the prime order group of "normal" size (say, 2λ bits), so that it can be for example a prime order group in which we can efficiently instantiate another CP-SNARK that could be composed with our $\mathsf{MemCP}_{\mathsf{RSAPrm}}$. And we can also allow flexible choices of the primes size that can be tuned to the application so that applications that work with moderately large sets can benefit in efficiency. In order to achieve these goals, our idea to create a membership proof is to compute the following:

- An accumulator membership witness $W = G^{\prod_{e_i \in P \setminus \{e\}} e_i}$, and an integer commitment to e in the RSA group, $C_e = G^e H^r$, where $H \in \mathsf{QR}_N$.
- A ZK proof of knowledge $\mathsf{CP}_{\mathsf{Root}}$ of a committed root for Acc, i.e. a proof of knowledge of e and W such that $W^e = \mathsf{Acc}$ and $C_e = G^e H^r$. Intuitively, this gives that C_e commits to an integer that is accumulated in Acc (at this point, however, the integer may be a trivial root, i.e., 1).
- A ZK proof $\mathsf{CP}_{\mathsf{modEq}}$ that C_e and c_e commit to the same value modulo q.
- A ZK proof $\mathsf{CP}_{\mathsf{range}}$ that c_e commits to an integer in the range $\left(2^{\mu-1}, 2^{\mu}\right)$.

From the combination of the above proofs we would like to conclude that the integer committed in c_e is in P. Without further restrictions, however, this may not be the case; in particular, since for the value committed in C_e we do not have a strict bound it may be that the integer committed in c_e is another e_q such $e = e_q \pmod q$ but $e \neq e_q$ over the integers. In fact, the proof $\mathsf{CP}_{\mathsf{Root}}$ does not guarantee us that C_e commits to a single prime number e, but only that e divides $\prod_{e_i \in P} e_i$, namely e might be a product of a few primes in P or the corresponding negative value, while its residue modulo q may be some value that is not in the set—what we call a "collision". We solve this problem by taking in consideration that e_q is guaranteed by $\mathsf{CP}_{\mathsf{range}}$ to be in $\left(2^{\mu-1}, 2^{\mu}\right)$ and by enhancing $\mathsf{CP}_{\mathsf{Root}}$ to also prove a bound on e: roughly speaking $|e| < 2^{2\lambda_s+\mu}$ for a statistical security parameter λ_s. Using this information we develop a careful analysis that bounds the probability that such collisions can happen for a malicious e (see Section 4.4.2 for more intuition).

In the following section we formally describe the type-based commitment scheme supported by our CP-SNARK, and a collection of building blocks. Then we present the $\mathsf{MemCP}_{\mathsf{RSA}}$

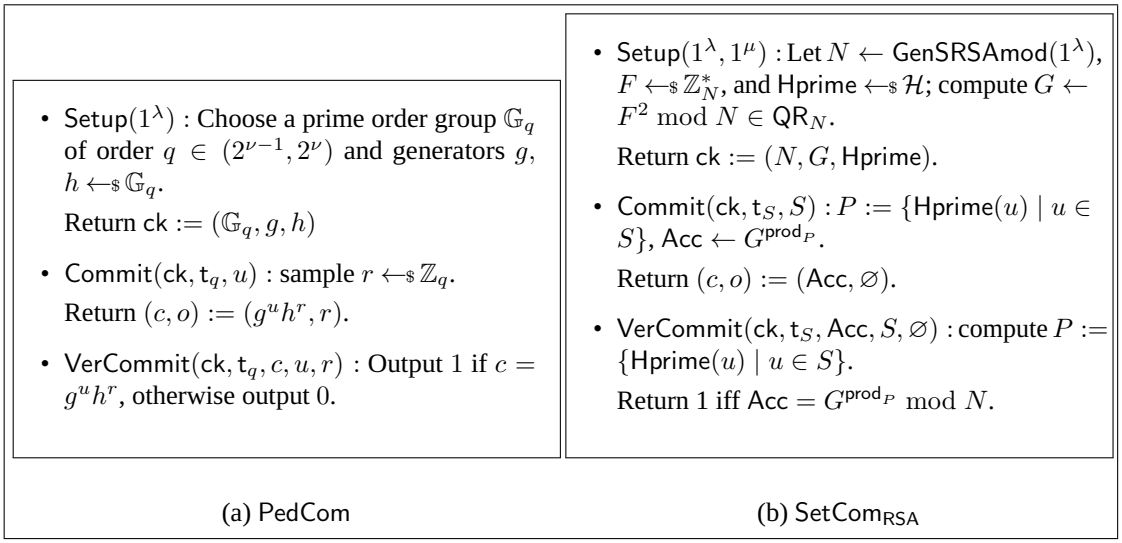


Figure 4.4: RSA Accumulator and Pedersen commitment schemes for RSAHashmem.

and $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ CP-SNARKs in Sections 4.4.2 and 4.4.3 respectively, and finally we give instantiations for some of our building blocks in Section 4.4.4.

Remark 8. *Although we specifically describe our protocols for RSA groups, they generalize to work over any Hidden Order Group with slight modifications. See appendix 4.8 for details.*

4.4.1 Preliminaries and Building Blocks

Notation. Given a set $S = \{u_1, \ldots, u_n\} \subset \mathbb{Z}$ of cardinality n we denote compactly with $\mathsf{prod}_S := \prod_{i=1}^n u_i$ the product of all its elements. We use capital letters for elements in an RSA group $\mathbb{Z}_N^*$, e.g., $G, H \in \mathbb{Z}_N^*$. Conversely, we use small letters for elements in a prime order group $\mathbb{G}_q$, e.g., $g, h \in \mathbb{G}_q$. Following this notation, we denote a commitment in a prime order group as $c \in \mathbb{G}_q$, while a commitment in an RSA group as $C \in \mathbb{Z}_N^*$.

Commitment Schemes. Our first CP-SNARK, called $\mathsf{MemCP}_{\mathsf{RSA}}$, is for a family of relations $R_{\mathsf{mem}} : \mathcal{D}_{\mathsf{elm}} \times \mathcal{D}_{\mathsf{set}}$ such that $\mathcal{D}_{\mathsf{elm}} = \{0,1\}^\eta$, $\mathcal{D}_{\mathsf{set}} = 2^{\mathcal{D}_{\mathsf{elm}}}$, and for a type-based commitment scheme that is the canonical composition $\mathsf{SetCom}_{\mathsf{RSA}} \bullet \mathsf{PedCom}$ of the two commitment schemes given in Fig. 4.4. PedCom is essentially a classical Pedersen commitment scheme in a group $\mathbb{G}_q$ of prime order q such that $q \in (2^{\nu-1}, 2^\nu)$ and $\eta < \nu$. PedCom is used to commit to set elements and its type is t_q. $\mathsf{SetCom}_{\mathsf{RSA}}$ is a (non-hiding) commitment scheme for sets of η-bit strings, that is built as an RSA accumulator [29, 18] to a set of μ-bit primes, each derived from an η-bit string by a deterministic hash function $\mathsf{Hprime} : \{0,1\}^\eta \to \mathbb{P}\left(2^{\mu-1}, 2^\mu\right)$. $\mathsf{SetCom}_{\mathsf{RSA}}$ is computationally binding under the factoring assumption[19] and the collision resistance of Hprime. Its type for sets is t_S.

Hashing to primes. The problem of mapping arbitrary values to primes in a collision-resistant manner has been studied in the past, see e.g., [115, 55, 84], and in [108] a method to generate

[19] Here is why: finding two different sets of primes $P, P', P \neq P'$ such that $G^{\mathsf{prod}_P} = \mathsf{Acc} = G^{\mathsf{prod}_{P'}}$ implies finding an integer $\alpha = \mathsf{prod}_P - \mathsf{prod}_{P'} \neq 0$ such that $G^\alpha = 1$. This is known to lead to an efficient algorithm for factoring N.

random primes is presented. Although the main idea of our scheme would work with any instantiation of Hprime, for the goal of significantly improving efficiency, our construction considers a specific class of Hprime functions that work as follows. Let $\mathsf{H} : \{0,1\}^{\eta} \times \{0,1\}^{\iota} \to \{0,1\}^{\mu-1}$ be a collision-resistant function, and define $\mathsf{Hprime}(u)$ as the function that starting with $j = 0$, looks for the first $j \in [0, 2^{\iota} - 1]$ such that the integer represented by the binary string $1|\mathsf{H}(u, j)$ is prime. In case it reaches $j = 2^{\iota} - 1$ it failed to find a prime and outputs $\perp$ [20]. We consider two main candidates of such function H (and thus Hprime):

- Pseudorandom function. Namely $\mathsf{H}(u, j) := \mathsf{F}_{\kappa}(u, j)$ where $\mathsf{F}_{\kappa} : \{0,1\}^{\eta+\iota}$ is a PRF with public seed κ and $\iota = \lceil \log \mu\lambda \rceil$. Due to the density of primes, the corresponding Hprime runs in the expected running time $O(\mu)$ and $\perp$ is returned with probability $\leq \mathsf{exp}(-\lambda) = \mathsf{negl}(\lambda)$.[21] Under the random oracle heuristic, F can be instantiated with a hash function like SHA256.
- Deterministic map. $\mathsf{H}(u, j) := f(u) + j$ with $u > 2^{\eta-1}$ and $j \in (f(u), f(u+1))$, where $f(u) := 2(u+2)\log_2(u+1)^2$. The corresponding function $\mathsf{Hprime}(u)$ is essentially the function that maps to the next prime after $f(u)$. This function is collision-free (indeed it requires to take $\mu > \eta$) and generates primes that can be smaller (in expectation) than the function above. Cramer's conjecture implies that the interval $(f(u), f(u+1))$ contains a prime when u is sufficiently large.

a**CP-NIZK for H computation and PedCom.** We use a CP-NIZK $\mathsf{CP}_{\mathsf{HashEq}}$ for the relation $R_{\mathsf{HashEq}} : \{0,1\}^{\mu} \times \{0,1\}^{\eta} \times \{0,1\}^{\iota}$ defined as

$$R_{\mathsf{HashEq}}(u_1, u_2, \omega) = 1 \Leftrightarrow u_1 = (1|\mathsf{H}(u_2, \omega))$$

and for the commitment scheme PedCom. Essentially, with this scheme one can prove that two commitments c_e and c_u in $\mathbb{G}_q$ are such that $c_e = g^e h^{r_q}$, $c_u = g^u h^{r_u}$ and there exists j such that $e = (1|\mathsf{H}(u, j))$. As it shall become clear in our security proof, we do not have to prove all the iterations of H until finding j such that $(1|\mathsf{H}(u, j)) = \mathsf{Hprime}(u)$ is prime, which saves significantly on the complexity of this CP-NIZK.

Integer Commitments. We use a scheme for committing to arbitrarily large integer values in RSA groups introduced by Fujisaki and Okamoto [111] and later improved in [85]. We briefly recall the commitment scheme. Let $\mathbb{Z}_N^*$ be an RSA group. The commitment key consists of two randomly chosen generators $G, H \in \mathbb{Z}_N^*$; to commit to any $x \in \mathbb{Z}$ one chooses randomly an $r \leftarrow_{\$} [1, N/2]$ and computes $C \leftarrow G^x H^r$; the verifier checks whether or not $C = \pm G^x H^r$. This commitment scheme is statistically hiding, as long as G and H lie in the subgroup of $\mathbb{Z}_N^*$. This can be achieved by setting $G \leftarrow F^2, H \leftarrow J^2 \in \mathsf{QR}(N)$, where F, J are randomly sampled from $\mathbb{Z}_N^*$. Moreover it's computationally binding under the assumption that factoring is hard in $\mathbb{Z}_N^*$. Furthermore, a proof of knowledge of an opening was presented in [85], its knowledge soundness was based on the strong RSA assumption, and later found to be reducible to the plain RSA assumption in [81]. We denote this commitment scheme as IntCom.

Strong-RSA Accumulators. As observed earlier, our commitment scheme for sets is an RSA accumulator Acc computed on the set of primes P derived from S through the map to primes, i.e., $P := \{\mathsf{Hprime}(s) | s \in S\}$. In our construction we use the accumulator's

[20] For specific instantiations of H, ι can be set so that $\perp$ is returned with negligible probability.

[21] We assume for simplicity that the function never outputs $\perp$, though it can happen with negligible probability.

feature for computing succinct membership witnesses, which we recall works as follows. Given $\mathsf{Acc} = G^{\prod_{e_i \in P} e_i} := G^{\mathsf{prod}_P}$, the membership witness for e_k is $W_k = G^{\prod_{e_i \in P \setminus \{e_k\}} e_i}$, which can be verified by checking if $W_k^{e_k} = \mathsf{Acc}$.

Argument of Knowledge of a Root. We make use of a zero-knowledge non-interactive argument of knowledge of a root of a public RSA group element $\mathsf{Acc} \in \mathsf{QR}_N$. This NIZK argument is called $\mathsf{CP}_{\mathsf{Root}}$. More precisely, it takes in an integer commitment to a $e \in \mathbb{Z}$ and then proves knowledge of an e-th root of Acc, i.e., of $W = \mathsf{Acc}^{\frac{1}{e}}$. More formally, $\mathsf{CP}_{\mathsf{Root}}$ is a NIZK for the relation $R_{\mathsf{Root}} : (\mathbb{Z}_N^* \times \mathsf{QR}_N \times \mathbb{N}) \times (\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}_N^*)$ defined as
$R_{\mathsf{Root}}((C_e, \mathsf{Acc}, \mu), (e, r, W)) = 1$ iff

$$C_e = \pm G^e H^r \bmod N \ \wedge \ W^e = \mathsf{Acc} \bmod N \ \wedge \ |e| < 2^{\lambda_z + \lambda_s + \mu + 2}$$

where λ_z and λ_s are the statistical zero-knowledge and soundness security parameters respectively of the protocol $\mathsf{CP}_{\mathsf{Root}}$. $\mathsf{CP}_{\mathsf{Root}}$ is obtained by applying the Fiat-Shamir transform to a public-coin protocol that we propose based on ideas from the protocol of Camenisch and Lysyanskaya for proving knowledge of an accumulated value [57]. In [57], the protocol ensures that the committed integer e is in a specific range, different from 1 and positive. In our $\mathsf{CP}_{\mathsf{Root}}$ protocol we instead removed these constraints and isolated the portion of the protocol that only proves knowledge of a root. We present the $\mathsf{CP}_{\mathsf{Root}}$ protocol in Section 4.4.4.1; its interactive public coin version is knowledge sound under the RSA assumption and statistical zero-knowledge. Finally, we notice that the relation R_{Root} is defined for statements where $\mathsf{Acc} \in \mathsf{QR}_N$, which may not be efficiently checkable given only N if Acc is adversarially chosen. Nevertheless $\mathsf{CP}_{\mathsf{Root}}$ can be used in larger cryptographic constructions that guarantee $\mathsf{Acc} \in \mathsf{QR}_N$ through some extra information, as is the case in our scheme.

Proof of Equality of Commitments in $\mathbb{Z}_N^*$ and $\mathbb{G}_q$. Our last building block, called $\mathsf{CP}_{\mathsf{modEq}}$, proves in zero-knowledge that two commitments, a Pedersen commitment in a prime order group and an integer commitment in an RSA group, open to the same value modulo the prime order $q = \mathsf{ord}(\mathbb{G})$. This is a conjunction of a classic Pedersen Σ-protocol and a proof of knowledge of opening of an integer commitment [85], i.e. for the relation

$$R_{\mathsf{modEq}}((C_e, c_e), (e, e_q, r, r_q)) = 1 \text{ iff}$$

$$e = e_q \bmod q \wedge C_e = \pm G^e H^r \bmod N \wedge c_e = g^{e_q \bmod q} h^{r_q \bmod q}$$

We present $\mathsf{CP}_{\mathsf{modEq}}$ in Section 4.4.4.2.

4.4.2 Our CP-SNARK $\mathsf{MemCP}_{\mathsf{RSA}}$

We are now ready to present our CP-SNARK $\mathsf{MemCP}_{\mathsf{RSA}}$ for set membership. The scheme is fully described in Figure 4.5 and makes use of the building blocks presented in the previous section.

The Setup algorithm takes as input the commitment key of Com_1 and a description of R_{mem} and does the following: it samples a random generator $H \leftarrow_{\$} \mathsf{QR}_N$ so that (G, H) define a key for the integer commitment, and generate a CRS $\mathsf{crs}_{\mathsf{HashEq}}$ of the $\mathsf{CP}_{\mathsf{HashEq}}$ CP-NIZK.

For generating a proof, the ideas are similar to the ones informally described at the beginning of Section 4.4 for the case when set elements are prime numbers. In order to support sets S of arbitrary strings the main differences are the following: (i) we use Hprime in order to derive a

set of primes P from S, (ii) given a commitment c_u to an element $u \in \{0,1\}^\eta$, we commit to $e = \mathsf{Hprime}(u)$ in c_e; (iii) we use the previously mentioned ideas to prove that c_e commits to an element in P (that is correctly accumulated), except that we replace the range proof π_{range} with a proof π_{HashEq} that c_u and c_e commits to u and e respectively, such that $\exists j : e = (1|\mathsf{H}(u,j))$.

Remark 9 (On the support of larger η.)**.** *In order to commit to a set element $u \in \{0,1\}^\eta$ with the* PedCom *scheme we require $\eta < \nu$. This condition is actually used for ease of presentation. It is straightforward to extend our construction to the case $\eta \geq \nu$, in which case every u should be split in blocks of less than ν bits that can be committed using the vector Pedersen commitment.*

- $\mathsf{KeyGen}(\mathsf{ck}, R^{\in})$: Parse $\mathsf{ck} := ((N, G, \mathsf{Hprime}), (\mathbb{G}_q, g, h))$ as the commitment keys of $\mathsf{SetCom}_{\mathsf{RSA}}$ and PedCom respectively. Sample a random generator H.

  Generate $\mathsf{crs}_{\mathsf{HashEq}} \leftarrow\!\!\$\ \mathsf{CP}_{\mathsf{HashEq}}.\mathsf{Setup}((\mathbb{G}_q, g, h), R_{\mathsf{HashEq}})$, a crs for $\mathsf{CP}_{\mathsf{HashEq}}$.

  Return $\mathsf{crs} := (N, G, H, \mathsf{Hprime}, \mathbb{G}_q, g, h, \mathsf{crs}_{\mathsf{HashEq}})$.

  Given crs, one can define $\mathsf{crs}_{\mathsf{Root}} := (N, G, H)$, $\mathsf{crs}_{\mathsf{modEq}} := (N, G, H, \mathbb{G}_q, g, h)$.

- $\mathsf{Prove}(\mathsf{crs}, (C_S, c_u), (S, u), (\varnothing, r_u))$: Compute $e \leftarrow \mathsf{Hprime}(u) = (1|\mathsf{H}(u,j))$, $(c_e, r_q) \leftarrow \mathsf{Com}_1.\mathsf{Commit}(\mathsf{ck}, \mathsf{t}_q, e)$.

  $(C_e, r) \leftarrow \mathsf{IntCom.Commit}((G,H), e)$; $P \leftarrow \{\mathsf{Hprime}(u) : u \in S\}$, $W = G^{\prod_{e_i \in P \setminus \{e\}} e_i}$.

  $\pi_{\mathsf{Root}} \leftarrow \mathsf{CP}_{\mathsf{Root}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{Root}}, (C_e, C_S, \mu), (e, r, W))$

  $\pi_{\mathsf{modEq}} \leftarrow \mathsf{CP}_{\mathsf{modEq}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{modEq}}, (C_e, c_e), (e, e, r, r_q))$

  $\pi_{\mathsf{HashEq}} \leftarrow \mathsf{CP}_{\mathsf{HashEq}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{HashEq}}, (c_e, c_u), (e, u), (r_q, r_u), j)$

  Return $\pi := (C_e, c_e, \pi_{\mathsf{Root}}, \pi_{\mathsf{modEq}}, \pi_{\mathsf{HashEq}})$.

- $\mathsf{VerProof}(\mathsf{crs}, (C_S, c_u), \pi)$: Return 1 iff
  $\mathsf{CP}_{\mathsf{Root}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{Root}}, (C_e, C_S, \mu), \pi_{\mathsf{Root}}) = 1 \wedge$
  $\mathsf{CP}_{\mathsf{modEq}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{modEq}}, (C_e, c_e), \pi_{\mathsf{modEq}}) = 1 \wedge$
  $\mathsf{CP}_{\mathsf{HashEq}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{HashEq}}, (c_e, c_u), \pi_{\mathsf{HashEq}}) = 1$

Figure 4.5: $\mathsf{MemCP}_{\mathsf{RSA}}$ CP-SNARK for set membership

The correctness of $\mathsf{MemCP}_{\mathsf{RSA}}$ can be checked by inspection: essentially, it follows from the correctness of all the building blocks and the condition that $\eta, \mu < \nu$. For succinctness, we observe that the commitments C_S, c_u and all the three proofs have size that does not depend on the cardinality of the set S, which is the only portion of the witness whose size is not a-priori fixed.

Proof of Security. Recall that the goal is to prove in ZK that c_u is a commitment to an element $u \in \{0,1\}^\eta$ that is in a set S committed in C_S. Intuitively, we obtain the security of our scheme from the conjunction of proofs for relations $R_{\mathsf{Root}}, R_{\mathsf{modEq}}$ and R_{HashEq}: (i) π_{HashEq} gives us that c_e commits to $e_q = (1|\mathsf{H}(u,j))$ for some j and for u committed in c_u. (ii) π_{modEq} gives that C_e commits to an integer e such that $e \mod q = e_q$ is committed in c_e. (iii) π_{Root} gives us that the integer e committed in C_e divides prod_P, where $C_S = G^{\mathsf{prod}_P}$ with $P = \{\mathsf{Hprime}(u_i) : u_i \in S\}$.

By combining these three facts we would like to conclude that $e_q \in P$ that, together with π_{HashEq}, should also guarantee $u \in S$. A first problem to analyze, however, is that for e we do not have guarantees of a strict bound in $\left(2^{\mu-1}, 2^{\mu}\right)$; so it may in principle occur that $e = e_q \pmod q$ but $e \neq e_q$ over the integers. Indeed, the relation R_{Root} does not guarantee us that e is a single prime number, but only that e divides the product of primes accumulated in C_S. Assuming the hardness of Strong RSA we may still have that e is the product of a few primes in P or even is a negative integer. We expose a simple attack that could arise from this: an adversary can find a product of primes from the set P, let it call e, such that $e = e_q \pmod q$ but $e \neq e_q$ over the integers. Since e is a legitimate product of members of P, the adversary can efficiently compute the e-th root of C_S and provide a valid π_{Root} proof. This is what we informally call a "collision". Another simple attack would be that an adversary takes a single prime e and then commits to its opposite $e_q \leftarrow -e \mod q$ in the prime order group. Again, since $e \in P$ the adversary can efficiently compute the e-th root of C_S, $W^e = C_S$, and then the corresponding $-e$-th root of C_S, $\left(W^{-1}\right)^{-e} = C_S$. This is a second type of attack to achieve what we called "collision". With a careful analysis we show that with appropriate parameters the probability that such collisions occur can be either 0 or negligible.

One key observation is that R_{Root} does guarantee a lower and an upper bound, $-2^{\lambda_z+\lambda_s+\mu+2}$ and $2^{\lambda_z+\lambda_s+\mu+2}$ respectively, for e committed in C_e. From these bounds (and that $e \mid \mathsf{prod}_P$) we get that an adversarial e can be the product of *at most* $d = 1 + \lfloor \frac{\lambda_z+\lambda_s+2}{\mu} \rfloor$ primes in P (or their corresponding negative product). Then, if $2^{d\mu} \leq 2^{\nu-2} < q$, or $d\mu + 2 \leq \nu$, we get that $e < 2^{d\mu} < q$. In case $e > 0$ and since q is prime, $e = e_q \bmod q \wedge e < q$ implies that $e = e_q$ over $\mathbb{Z}$, namely no collision can occur at all. In the other case $e < 0$ we have $e > -2^{d\mu}$ and $e = e_q \pmod q$ implies $e = -q + e_q < -2^{\nu-1} + 2^{\mu} < -2^{\nu-1} + 2^{\nu-2} = -2^{\nu-2}$. Therefore, $-2^{d\mu} < -2^{\nu-2}$, which is a contradiction since we assumed $d\mu + 2 \leq \nu$. So this type of collision cannot happen.

If on the other hand we are in a parameters setting where $d\mu > \nu - 2$, we give a concrete bound on the probability that such collisions occur. More precisely, for this case we need to assume that the integers returned by H are random, i.e., H is a random oracle, and we also use the implicit fact that R_{HashEq} guarantees that $e_q \in \left(2^{\mu-1}, 2^{\mu}\right)$. Then we give a concrete bound on the probability that the product of d out of $\mathsf{poly}(\lambda)$ random primes lies in a specific range $\left(2^{\mu-1}, 2^{\mu}\right)$, which turns out to be negligible when d is constant and $2^{\mu-\nu}$ is negligible.

Since the requirements of security are slightly different according to the setting of parameters mentioned above, we state two separate theorems, one for each case.

Theorem 1. *Let* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}}$ *and* IntCom *be computationally binding commitments,* $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{HashEq}}$ *be knowledge-sound NIZK arguments, and assume that the Strong RSA assumption holds, and that* H *is collision resistant. If* $d\mu + 2 \leq \nu$*, then* $\mathsf{MemCP}_{\mathsf{RSA}}$ *is knowledge-sound with partial opening of the set commitments* C_S*.*

Theorem 2. *Let* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}}$ *and* IntCom *be computationally binding commitments,* $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{HashEq}}$ *be knowledge-sound NIZK arguments, and assume that the Strong RSA assumption hold, and that* H *is collision resistant. If* $d\mu + 2 > \nu$*,* $d = O(1)$ *is a small constant,* $2^{\mu-\nu} \in \mathsf{negl}(\lambda)$ *and* H *is modeled as a random oracle, then* $\mathsf{MemCP}_{\mathsf{RSA}}$ *is knowledge-sound with partial opening of the set commitments* C_S*.*

Remark 10. *It is worth noting that Theorem 2 where we assume* H *to be a random oracle requires a random oracle assumption stronger than usual; this has to do with the fact that while*

we assume H *to be a random oracle we also assume that* $\mathsf{CP}_{\mathsf{modEq}}$ *can create proof about correct computations of* H. *Similar assumptions have been considered in previous works, see, e.g, [208, Remark 2].*

Finally, we state the theorem about the zero-knowledge of $\mathsf{MemCP}_{\mathsf{RSA}}$.

Theorem 3. *Let* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}}$ *and* IntCom *be statistically hiding commitments,* $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{HashEq}}$ *be zero-knowledge arguments. Then* $\mathsf{MemCP}_{\mathsf{RSA}}$ *is zero-knowledge.*

sketch. The proof is rather straightforward, so we only provide a sketch. We define the simulator $\mathcal{S}$ that takes as input (crs, C_S, c_u) and does the following:

- Parses $\mathsf{crs} := (N, G, H, \mathsf{Hprime}, \mathbb{G}_q, g, h, \mathsf{crs}_{\mathsf{HashEq}})$, from which it computes the corresponding $\mathsf{crs}_{\mathsf{Root}} := (N, G, H)$ and $\mathsf{crs}_{\mathsf{modEq}} := (N, G, H, \mathbb{G}_q, g, h)$.
- Samples at random $C_e^* \leftarrow_{\$} \mathbb{Z}_N^*$ and $c_e^* \leftarrow_{\$} \mathbb{G}_q$.
- Invokes $\mathcal{S}_{\mathsf{Root}}(\mathsf{crs}_{\mathsf{Root}}, C_e^*, C_S)$, $\mathcal{S}_{\mathsf{modEq}}(\mathsf{crs}_{\mathsf{modEq}}, C_e^*, c_e^*)$ and $\mathcal{S}_{\mathsf{HashEq}}(\mathsf{crs}_{\mathsf{HashEq}}, c_e^*, c_u)$ the corresponding simulators of $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ and $\mathsf{CP}_{\mathsf{HashEq}}$ respectively. They output simulated proof π^*_{Root}, π^*_{modEq} and π^*_{HashEq} respectively.
- $\mathcal{S}$ outputs $(C_e^*, c_e^*, \pi^*_{\mathsf{Root}}, \pi^*_{\mathsf{modEq}}, \pi^*_{\mathsf{HashEq}})$.

Let $\pi := (C_e, c_e, \pi_{\mathsf{Root}}, \pi_{\mathsf{modEq}}, \pi_{\mathsf{HashEq}}) \leftarrow \mathsf{Prove}(\mathsf{crs}, (C_S, c_u), (S, u), (\varnothing, r_u))$ be the output of a real proof. Since IntCom and PedCom are statistically hiding C_e^* and c_e^* are indistinguishable from C_e and c_e resp. Finally, since $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ and $\mathsf{CP}_{\mathsf{HashEq}}$ are zero knowledge arguments π^*_{Root}, π^*_{modEq} and π^*_{HashEq} are indistinguishable from π_{Root}, π_{modEq} and π_{HashEq} resp. □

Notation. We introduce some notation that eases our proofs exposition. Let $S = \{u_1, \ldots, u_n\} \subset \mathbb{Z}$ be a set of cardinality n. We denote as prod a product of (an arbitrary number of) elements of S, $\mathsf{prod} = \prod_{i \in I} u_i$, for some $I \subseteq [n]$. Furthermore, $\Pi_S = \{\mathsf{prod}_1, \ldots, \mathsf{prod}_{2^n-1}\}$ is the set of all possible products and more specifically $\Pi_{S,d} \subseteq \Pi_S$ denotes the set of possible products of exactly d elements of S, $|I| = d$, while for the degenerate case of $d > n$ we define $\Pi_{S,d} = \emptyset$. We note that $|\Pi_{S,d}| = \binom{n}{d}$ (except for the degenerate case where $|\Pi_{S,d}| = 0$). For convenience, in the special case of $\mathsf{prod} \in \Pi_{S,|S|}$, i.e. the (unique) product of all elements of S, we will simply write prod_S. Finally, for a $J \subseteq [n]$ we let $\Pi_{S,J} = \cup_{j \in J} \Pi_{S,j}$; for example $\Pi_{S,[1,\ldots,d]} = \cup_{j=1}^{d} \Pi_{S,j}$ is the set of all possible products of up to d elements of S. For all of the above we also denote with "$-$" the corresponding set of the opposite element, e.g. $-\Pi_S = \{-\mathsf{prod}_1, \ldots, -\mathsf{prod}_{2^n-1}\}$

of Theorem 1. Let a malicious prover $\mathcal{P}^*$, a PPT adversary of Knowledge Soundness with Partial Opening (see the definition in section 4.2.2) that on input $(\mathsf{ck}, R_{\mathsf{mem}}, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$ outputs (C_S, c_u, S, π) such that the verifier $\mathcal{V}$ accepts, i.e. $\mathsf{VerProof}(\mathsf{crs}, C_S, c_u), \pi) = 1$ and $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_S, C_S, S, \varnothing) = 1$ with non-negligible probability ϵ. We will construct a PPT extractor $\mathcal{E}$ that on the same input outputs a partial witness (u, r_q) such that $R_{\mathsf{mem}}(S, u) = 1 \wedge \mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_u, u, r_q) = 1$.

For this we rely on the Knowledge Soundness of $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ and $\mathsf{CP}_{\mathsf{HashEq}}$ protocols. $\mathcal{E}$ parses $\pi := (C_e, c_e, \pi_{\mathsf{Root}}, \pi_{\mathsf{modEq}}, \pi_{\mathsf{HashEq}})$ and $\mathsf{crs} := (N, G, H, \mathsf{Hprime}, \mathbb{G}_q, g, h, \mathsf{crs}_{\mathsf{HashEq}})$, from which it computes the corresponding $\mathsf{crs}_{\mathsf{Root}} := (N, G, H)$ and $\mathsf{crs}_{\mathsf{modEq}} := (N, G,$

H, $\mathbb{G}_q$, g, h). Then constructs an adversary $\mathcal{A}_{\mathsf{Root}}$ for $\mathsf{CP}_{\mathsf{Root}}$ Knowledge Soundness that outputs $(C_e, C_S, \mu, \pi_{\mathsf{Root}})$. It is obvious that since $\mathcal{V}$ accepts π then it also accepts π_{Root}, i.e., $\mathsf{CP}_{\mathsf{Root}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{Root}}, (C_e, C_S, \mu), \pi_{\mathsf{Root}}) = 1$. From Knowledge Soundness of $\mathsf{CP}_{\mathsf{Root}}$ we know that there is an extractor $\mathcal{E}_{\mathsf{Root}}$ that outputs (e, r, W) such that $C_e = \pm G^e H^r \pmod N \wedge W^e = C_S \pmod N \wedge |e| < 2^{\lambda_z+\lambda_s+\mu+2}$. Similarly, $\mathcal{E}$ constructs adversaries $\mathcal{A}_{\mathsf{modEq}}$ and $\mathcal{A}_{\mathsf{HashEq}}$ of protocols $\mathsf{CP}_{\mathsf{modEq}}$ and $\mathsf{CP}_{\mathsf{HashEq}}$ respectively. And similarly there are extractors $\mathcal{E}_{\mathsf{modEq}}$ and $\mathcal{E}_{\mathsf{HashEq}}$ that output (e', e_q, r', r_q) such that $e' = e_q \pmod q \wedge C_{e'} = \pm G^{e'} H^{r'} \pmod N \wedge c_{e_q} = g^{e_q \bmod q} h^{r_q \bmod q}$ and (e'_q, u, r'_q, r_u, j) such that $c_e = g^{e'_q} h^{r'_q} \wedge e'_q = (1|\mathsf{H}(u, j))$ respectively.

From the Binding property of the integer commitment scheme we get that $e = e'$ and $r = r'$ (over the integers), unless with a negligible probability. Similarly, from the Binding property of the Pedersen commitment scheme we get that $e_q = e'_q \pmod q$ and $r_q = r'_q \pmod q$, unless with a negligible probability. So if we put everything together the extracted values are $(e, r, W, e_q, r_q, u, r_u, j)$ such that:

$$W^e = C_S \pmod N \wedge |e| < 2^{\lambda_z+\lambda_s+\mu+2} \wedge e = e_q \pmod q \wedge e_q = (1|\mathsf{H}(u, j))$$

and additionally

$$C_e = \pm G^e H^r \wedge c_e = g^{e_q \bmod q} h^{r_q \bmod q} \wedge \mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_u, u, r_u) = 1$$

From $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_S, C_S, S, \varnothing) = 1$ we infer that $C_S = G^{\mathsf{prod}_P}$, where $P := \{\mathsf{Hprime}(u) \mid u \in S\}$. From the strong RSA assumption since $W^e = C_S = G^{\mathsf{prod}_P} \pmod N$ we get $e \in \Pi_P$ or $e \in -\Pi_P$, unless with a negligible probability.

Since, all the elements of P are outputs of Hprime they have exactly bitlength μ, that is $2^{\mu-1} < e_i < 2^\mu$ for each $e_i \in P$. This means that e is a ($\pm$) product of μ-sized primes. Let $|e|$ be a product of ℓ primes, meaning that $2^{\ell(\mu-1)} < |e| < 2^{\ell\mu}$, and $d := \lfloor \frac{\lambda_z+\lambda_s+\mu+2}{\mu} \rfloor$. From $|e| < 2^{\lambda_z+\lambda_s+\mu+2}$ we get that $2^{\ell\mu} < 2^{\lambda_z+\lambda_s+\mu+2} \Rightarrow \ell < d$ which means that $e \in \Pi_{P,[1,\ldots,d]}$ or $e \in -\Pi_{P,[1,\ldots,d]}$ (i.e. e is a ($\pm$) product of at most d primes).

First we show that $e \in \Pi_P$, i.e., that e cannot be negative. Let $e \in -\Pi_{P,[1,\ldots,d]}$. We use the fact that $e = e_q \pmod q$, so $e \leq -q + e_q < -2^{\nu-1} + 2^\mu < -2^{\nu-1} + 2^{\nu-2} = -2^{\nu-2}$. Since $-2^{d\mu} < e$ this leads to $-2^{d\mu} < -2^{\nu-2}$ which contradicts the assumption $d\mu + 2 \leq \nu$ (we used the fact that $e_q = (1|\mathsf{H}(u, j))$ to conclude that $2^{\mu-1} < e_q < 2^\mu$, which comes from the definition of H). So $e > 0$ or $e \in \Pi_{P,[1,\ldots,d]}$.

Recall that $e < 2^{d\mu}$. From the assumption $d\mu + 2 \leq \nu$ which means that $e < 2^{d\mu} < 2^{\nu-2} < q \Rightarrow e < q$. Since $e = e_q \pmod q$ and $e < q$ this means that $e = e_q$ over the integers. Again we are using the fact that $e_q = (1|\mathsf{H}(u, j))$ to conclude that $2^{\mu-1} < e_q < 2^\mu$, which comes from the definition of H, and combined with $e = e_q$ we get that $2^{\mu-1} < e < 2^\mu$. The last fact means that $e \in \Pi_{P,\{1\}}$ (i.e. e is exactly one prime from P) otherwise it would exceed 2^μ, so $e \in P$.

Finally, $e = e_q = (1|\mathsf{H}(u, j)) = \mathsf{Hprime}(u) \in P = \{\mathsf{Hprime}(u_1), \ldots, \mathsf{Hprime}(u_n)\}$, where $S := \{u_1, \ldots, u_n\}$. This means that there is an i such that $\mathsf{Hprime}(u) = \mathsf{Hprime}(u_i)$. From collision resistance of Hprime we infer that $u = u_i$. So we conclude that $u \in S$ or $R_{\mathsf{mem}}(S, u) = 1$ and as shown above $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_u, u, r_u) = 1$. □

4.4.2.1 Collision Finding Analysis

For the second theorem we cannot count on the formula $d\mu + 2 \leq \nu$ that ensures that the extracted integer e lies inside $[0, q-1]$. As explained above, we can only rely on the randomness

of each prime to avoid the described "collisions". First, we formally define what a "collision" is through a probabilistic experiment, CollisionFinding, and then we compute a concrete bound for the probability that this event happens, i.e. the experiment outputs 1. Finally, we state a theorem that shows this probability is asymptotically negligible under the assumption that $2^{\mu-\nu}$ is a negligible value (and d is a constant).

$\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_q, n)$

Let $P \subseteq \mathbb{P}\left(2^{\mu-1}, 2^{\mu}\right)$ be a randomly chosen set of cardinality n, i.e. each $e \in P$ is chosen uniformly at random, $e_i \leftarrow\!\!\$\, \mathbb{P}\left(2^{\mu-1}, 2^{\mu}\right)$ meaning that:

1. e_i is prime.
2. $2^{\mu-1} \leq e_i \leq 2^{\mu}$
3. $Pr[e_i = x] = \frac{\mu}{2^{\mu}} + \mathsf{negl}(\lambda)$ for each $x \in \mathbb{P}\left(2^{\mu-1}, 2^{\mu}\right)$

The output of the experiment is 1 iff there exists $\mathsf{prod} \in \left(\Pi_{P,[2,d]} \cup -\Pi_{P,[2,d]}\right)$ such that $(\mathsf{prod} \mod q) \in \left(2^{\mu-1}, 2^{\mu}\right)$

Lemma 5. *Let $\mathbb{G}_q$ be a prime order group of order $q \in \left(2^{\nu-1}, 2^{\nu}\right)$ and μ such that $\mu < \nu$ then* $Pr[\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_e, n) = 1] \leq 2 \cdot \sum_{j=2}^{d} \frac{\binom{n}{j} 2^{(j+1)\mu-j-\nu}(2^j-1)}{\frac{2^{j\mu-j}}{(\mu-1)^j} - \binom{n}{j}}$

Proof. First we will prove it for positive products, that is we bound the probability $Pr[\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_e, n) = 1 | \mathsf{prod} \in \Pi_{P,[2,d]}]$. Let $\mathsf{prod} = q_1 ... q_j$ be a product of exactly j primes for a $2 \leq j \leq d$. Since $q_i \in \left(2^{\mu-1}, 2^{\mu}\right)$ we get $\mathsf{prod} = q_1 ... q_j \in \left(2^{j\mu-j}, 2^{j\mu}\right)$. Also $\mathbb{Z}_q^*$ is cyclic so we know that at most

$$\left\lceil \frac{\left|\left(2^{j\mu-j}, 2^{j\mu}\right)\right|}{q} \right\rceil = \left\lceil \frac{2^{j\mu} - 2^{j\mu-j}}{q} \right\rceil = \left\lceil \frac{2^{j\mu-j} \cdot (2^j - 1)}{q} \right\rceil \leq 2^{j\mu-j-\nu+1} \cdot (2^j - 1)$$

integers in $\left(2^{j\mu-j}, 2^{j\mu}\right)$ are equal to c modulo q, for any $c \in \{0, 1, ..., q-1\}$.

We are interested in the interval $\left(2^{\mu-1}, 2^{\mu}\right)$ modulo q. From the previous we get that at most $2^{j\mu-j-\nu+1} \cdot (2^j - 1) \cdot \left|\left(2^{\mu-1}, 2^{\mu}\right)\right| = 2^{j\mu-j-\nu+1} \cdot (2^j - 1) \cdot 2^{\mu-1} = 2^{(j+1)\mu-j-\nu}(2^j - 1)$ integers in the range of $\left(2^{j\mu-j}, 2^{j\mu}\right)$ are "winning" integers for the adversary, meaning that after modulo q they are mapped to the winning interval $\left(2^{\mu-1}, 2^{\mu}\right)$.

From the distribution of primes we know that the number of primes in $\left(2^{\mu-1}, 2^{\mu}\right)$ is approximately $\frac{2^{\mu-1}}{\mu-1}$. So there are (approximately) $\left(\frac{2^{\mu-1}}{\mu-1}\right)^j = \frac{2^{j\mu-j}}{(\mu-1)^j}$ different products of j primes from $\mathbb{P}\left(2^{\mu-1}, 2^{\mu}\right)$ in $\left(2^{j\mu-d}, 2^{j\mu}\right)$. This leads us to the combinatorial experiment of choice of $B = \frac{2^{j\mu-j}}{(\mu-1)^j}$ "balls", with $T = 2^{(j+1)\mu-j-\nu}(2^j - 1)$ "targets" and $X = \binom{n}{j}$ "tries" without replacement, where "balls" are all possible products, "targets" are the ones that go to $\left(2^{\mu-1}, 2^{\mu}\right)$ modulo q (the winning ones) and tries are the number of products (for a constant j) that the adversary can try. The "without replacement" comes from the fact that all products are different. The final winning probability is:

$$\begin{aligned} Pr[\mathsf{prod} \mod q \in \left(2^{\mu-1}, 2^{\mu}\right) \wedge \mathsf{prod} \in \Pi_{P,j}] &\leq \frac{T}{B} + \frac{T}{B-1} + \frac{T}{B-2} + \ldots + \frac{T}{B-X} \\ &\leq X \cdot \frac{T}{B-X} \\ &= \frac{\binom{n}{j} 2^{(j+1)\mu-j-\nu}(2^j-1)}{\frac{2^{j\mu-j}}{(\mu-1)^j} - \binom{n}{j}} \end{aligned}$$

By applying the union bound for all j's we get:

$$Pr[\mathsf{prod} \mod q \in \left(2^{\mu-1}, 2^{\mu}\right) \wedge \mathsf{prod} \in \Pi_{P,[2,d]}] \leq \sum_{j=2}^{d} \frac{\binom{n}{j} 2^{(j+1)\mu-j-\nu}(2^j-1)}{\frac{2^{j\mu-j}}{(\mu-1)^j} - \binom{n}{j}}$$

By using the same arguments for negative products we would conclude that

$$Pr[\mathsf{prod} \mod q \in \left(2^{\mu-1}, 2^{\mu}\right) \wedge \mathsf{prod} \in -\Pi_{P,[2,d]}] \leq \sum_{j=2}^{d} \frac{\binom{n}{j} 2^{(j+1)\mu-j-\nu}(2^j-1)}{\frac{2^{j\mu-j}}{(\mu-1)^j} - \binom{n}{j}}$$

Therefore

$$\begin{aligned} Pr[\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_e, n) = 1] =& Pr[\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_e, n) = 1 \wedge \mathsf{prod} \in \Pi_{P,[2,d]}] + \\ &+ Pr[\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_e, n) = 1 \wedge \mathsf{prod} \in -\Pi_{P,[2,d]}] = \\ \leq& 2 \cdot \sum_{j=2}^{d} \frac{\binom{n}{j} 2^{(j+1)\mu-j-\nu}(2^j-1)}{\frac{2^{j\mu-j}}{(\mu-1)^j} - \binom{n}{j}} \end{aligned}$$

□

Theorem 4. *Let $\mathbb{G}_q$ be a prime order group of order $q \in \left(2^{\nu-1}, 2^{\nu}\right)$, μ such that $2^{\mu-\nu} \in \mathsf{negl}(\lambda)$, d constant and $n = \mathsf{poly}(\lambda)$ then $Pr[\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_q, n) = 1] \in \mathsf{negl}(\lambda)$*

Proof. Now $n = \mathsf{poly}(\lambda)$ so the set P is polynomially bounded. Due to lemma 5 it is straightforward that $Pr[\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_q, n) = 1] \leq \sum_{j=2}^{d} \frac{\binom{n}{j} 2^{(j+1)\mu-j-\nu}(2^j-1)}{\frac{2^{j\mu-j}}{(\mu-1)^j} - \binom{n}{j}}$. Since d is constant, for any $j \in [2, d]$ $\binom{n}{j} = O(n^j)$ and we get:

$$\begin{aligned} 2 \cdot \frac{\binom{n}{j} 2^{(j+1)\mu-j-\nu}(2^j-1)}{\frac{2^{j\mu-j}}{(\mu-1)^j} - \binom{n}{j}} &= 2 \cdot \frac{O(n^j) 2^{(j+1)\mu-j-\nu}(2^j-1)}{\frac{2^{j\mu-j}}{(\mu-1)^j} - O(n^j)} \\ &= 2 \cdot \frac{O(n^j)(2^j-1)(\mu-1)^j}{\frac{2^{j\mu-j}}{2^{(j+1)\mu-j-\nu}} - \frac{O(n^j)(\mu-1)^j}{2^{(j+1)\mu-j-\nu}}} \end{aligned}$$

$O(n^j)(2^j-1)(\mu-1)^j = \mathsf{poly}(\lambda)$ and $\frac{O(n^j)(\mu-1)^j}{2^{(j+1)\mu-j-\nu}} = \mathsf{negl}(\lambda)$. Also $\frac{2^{j\mu-j}}{2^{(j+1)\mu-j-\nu}} = 2^{\nu-\mu}$, therefore for j we get a probability bounded by $\frac{\mathsf{poly}(\lambda)2^{\mu-\nu}}{1-\mathsf{negl}(\lambda)2^{\mu-\nu}} = \mathsf{negl}(\lambda)$ by assumption.

Finally, $Pr[\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_q, n) = 1] \leq (d-1) \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$. □

- $\mathsf{Setup}(1^\lambda, 1^\mu)$: Sample an RSA modulus $N \leftarrow \mathsf{GenSRSAmod}(1^\lambda)$, a random group element $F \leftarrow\!\!\$\ \mathbb{Z}_N^*$, compute $G \leftarrow F^2 \bmod N \in \mathsf{QR}_N$. Return $\mathsf{ck} := (N, G)$.
- $\mathsf{Commit}(\mathsf{ck}, \mathsf{t}_S, S)$: compute $\mathsf{Acc} \leftarrow G^{\mathsf{prod}_P}$. Return $(c, o) := (\mathsf{Acc}, \varnothing)$.
- $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_S, \mathsf{Acc}, S, \varnothing)$: Return 1 if for all $e_i \in P$ $e_i \in \mathbb{P}\left(2^{\mu-1}, 2^\mu\right)$ and $\mathsf{Acc} = G^{\mathsf{prod}_P} \bmod N$, and 0 otherwise.

Figure 4.6: $\mathsf{SetCom}_{\mathsf{RSA}'}$ Commitment to Sets.

Remark 11. *For the sake of generality, in* CollisionFinding *we do not specify how the random primes are generated. In practice in our scheme they are outputs of the hash function* Hprime *that we model as a random oracle.*

Now we are ready to give the proof of theorem 2:

of theorem 2. The proof is almost the same as the one of Theorem 1 except for the next-to-last paragraph, i.e. the justification of $e \in \Pi_{P,\{1\}}$. Since $d\mu + 2 > \nu$ we cannot use the same arguments to conclude to it. However, still $e \in \left(\Pi_{P,[1,\ldots,d]} \cup -\Pi_{P,[1,\ldots,d]}\right)$.

Let $e \in \left(\Pi_{P,[1,\ldots,d]} \cup -\Pi_{P,[1,\ldots,d]}\right)$, it is straightforward to reduce this case to the the collision finding problem. Assume that the adversary $\mathcal{P}^*$ made q_H random oracle queries to H and let Q_H be the set of answers she received. Further assume that exactly q_Hprime of the them are primes and let Q_Hprime be the set of them. We note that $P \subseteq Q_\mathsf{Hprime}$, unless a collision happened in H.

Now let Q_Hprime be the set of the $\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_q, |Q_\mathsf{Hprime}|)$ experiment. It satisfies all three conditions since each $e_i \in Q_\mathsf{Hprime}$ is an output of Hprime. Therefore e_i is prime, $2^{\mu-1} < e_i < 2^\mu$ and since H is modeled as a random oracle the outputs of Hprime are uniformly distributed in $\mathbb{P}\left(2^{\mu-1}, 2^\mu\right)$. Then for the extracted e, we know that $e = e_q \pmod q \in \left(2^{\mu-1}, 2^\mu\right)$ and from the assumption $e \in \left(\Pi_{P,[1,\ldots,d]} \cup -\Pi_{P,[1,\ldots,d]}\right)$, which (as noted above) means that $e \in \left(\Pi_{Q_\mathsf{Hprime},[2,\ldots,d]} \cup -\Pi_{Q_\mathsf{Hprime},[2,\ldots,d]}\right)$. So $\mathsf{CollisionFinding}(\mu, d, \mathbb{G}_q, |Q_\mathsf{Hprime}|) = 1$. Since the adversary is PPT $|Q_\mathsf{Hprime}| = \mathsf{poly}(\lambda)$. Also, $d = O(1)$ and $2^{\mu-\nu} \in \mathsf{negl}(\lambda)$ (from the assumptions of the theorem) so the previous happens with a negligible probability according to theorem 4. So we conclude that, unless with a negligible probability, $e \in \Pi_{P,\{1\}}$. □

4.4.3 Our CP-SNARK for Set Membership for Primes Sets

In this section we show a CP-SNARK for set membership $\mathsf{MemCP}_\mathsf{RSAPrm}$ that supports set elements that are prime numbers of exactly μ bits, i.e., $\mathcal{D}_\mathsf{elm} = \mathbb{P}(2^{\mu-1}, 2^\mu)$, and $\mathcal{D}_\mathsf{set} = 2^{\mathcal{D}_\mathsf{elm}}$. $\mathsf{MemCP}_\mathsf{RSAPrm}$ works for a type-based commitment scheme Com_2 that is the canonical composition $\mathsf{SetCom}_{\mathsf{RSA}'} \bullet \mathsf{PedCom}$ where $\mathsf{SetCom}_{\mathsf{RSA}'}$ is in Fig. 4.6 (it is essentially a simplification of $\mathsf{SetCom}_\mathsf{RSA}$ since elements are already primes).

The scheme $\mathsf{MemCP}_\mathsf{RSAPrm}$ is described in figure 4.7. Its building blocks are the same as the ones for $\mathsf{MemCP}_\mathsf{RSA}$ except that instead of a CP-NIZK for proving correctness of a map-to-prime computation, we use a CP-NIZK for range proofs. Namely, we let $\mathsf{CP}_\mathsf{range}$ be a NIZK for the following relation on PedCom commitments c and two given integers $A < B$:

$$R_\mathsf{range}\left((c_e, A, B), (e, r_q)\right) = 1 \ \text{ iff } \ c = g^e h^{r_q} \ \wedge \ A < e_q < B$$

- $\mathsf{KeyGen}(\mathsf{ck}, R^{\in})$: parse $\mathsf{ck} := ((N, G), (\mathbb{G}_q, g, h))$ as the commitment keys of $\mathsf{SetCom}_{\mathsf{RSA}'}$ and PedCom respectively. Sample a random generator H.

  Generate $\mathsf{crs}_{\mathsf{range}} \leftarrow_{\$} \mathsf{CP}_{\mathsf{range}}.\mathsf{Setup}((\mathbb{G}_q, g, h), R_{\mathsf{range}})$, a crs for $\mathsf{CP}_{\mathsf{range}}$.

  Return $\mathsf{crs} := (N, G, H, \mathbb{G}_q, g, h, \mathsf{crs}_{\mathsf{range}})$.

  Given crs, one can define $\mathsf{crs}_{\mathsf{Root}} := (N, G, H)$, $\mathsf{crs}_{\mathsf{modEq}} := (N, G, H, \mathbb{G}_q, g, h)$.
- $\mathsf{Prove}(\mathsf{crs}, (C_P, c_e), (P, e), (\varnothing, r_q))$:

  $(C_e, r) \leftarrow \mathsf{IntCom.Commit}((G, H), e)$

  $W = G^{\prod_{e_i \in P \setminus \{e\}} e_i}$.

  $\pi_{\mathsf{Root}} \leftarrow \mathsf{CP}_{\mathsf{Root}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{Root}}, (C_e, C_P, \mu), (e, r, W))$

  $\pi_{\mathsf{modEq}} \leftarrow \mathsf{CP}_{\mathsf{modEq}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{modEq}}, (C_e, c_e), (e, e, r, r_q))$

  $\pi_{\mathsf{range}} \leftarrow \mathsf{CP}_{\mathsf{range}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{range}}, (2^{\mu-1}, 2^{\mu}), c_e, e, r_q)$

  Return $\pi := (C_e, \pi_{\mathsf{Root}}, \pi_{\mathsf{modEq}}, \pi_{\mathsf{range}})$.
- $\mathsf{VerProof}(\mathsf{crs}, (C_P, c_e), \pi)$: Return 1 iff
  $\mathsf{CP}_{\mathsf{Root}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{Root}}, (C_e, C_P, \mu), \pi_{\mathsf{Root}}) = 1 \wedge$
  $\mathsf{CP}_{\mathsf{modEq}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{modEq}}, (C_e, c_e), \pi_{\mathsf{modEq}}) = 1 \wedge$
  $\mathsf{CP}_{\mathsf{range}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{range}}, c_e, \pi_{\mathsf{range}}) = 1$.

Figure 4.7: $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ CP-SNARK for set membership

The idea behind the security of the scheme is similar to the one of the $\mathsf{MemCP}_{\mathsf{RSA}}$ scheme. The main difference is that here we rely on the range proof π_{range} in order to "connect" the Pedersen commitment c_e to the accumulator. In particular, in order to argue the absence of possible collisions here we assume that $d\mu + 2 \leq \nu$ holds, namely we argue security only for this setting of parameters. It is worth noting that in applications where $\mathcal{D}_{\mathsf{elm}}$ is randomly chosen subset of $\mathbb{P}\left(2^{\mu-1}, 2^{\mu}\right)$, we could argue security even when $d\mu + 2 > \nu$, in a way similar to Theorem 2. We omit the analysis of this case from the paper.

Theorem 5. *Let* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}'}$ *and* IntCom *be computationally binding commitments,* $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{range}}$ *be knowledge-sound NIZK arguments, and assume that the Strong RSA assumption hold. If* $d\mu + 2 \leq \nu$, *then* $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ *is knowledge-sound with partial opening of the set commitments* c_P. *Furthermore, if* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}'}$ *and* IntCom *are statistically hiding commitments, and* $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{range}}$ *be zero-knowledge, then* $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ *is zero-knowledge.*

of Theorem 5. Knowledge Soundness with Partial Opening of C_P: the proof is similar to the one of theorem 1 except for some minor parts.
Let a malicious prover $\mathcal{P}^*$, a PPT adversary of Knowledge Soundness with Partial Opening (see the definition in section 4.2.2) that on input $(\mathsf{ck}, R_{\mathsf{mem}}, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$ outputs (C_P, c_e, P, π) such that the verifier $\mathcal{V}$ accepts, i.e. $\mathsf{VerProof}(\mathsf{crs}, C_P, c_e), \pi) = 1$ and $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_S, C_P, P, \varnothing) = 1$ with non-negligible probability ϵ. We will construct a PPT extractor $\mathcal{E}$ that on the same input outputs a partial witness (e, r) such that $R_{\mathsf{mem}}(P, e) = 1 \wedge \mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_e, e, r) = 1$.

For this we rely on the Knowledge Soundness of $\mathsf{CP}_{\mathsf{Root}}$, $\mathsf{CP}_{\mathsf{modEq}}$ and $\mathsf{CP}_{\mathsf{range}}$ protocols. $\mathcal{E}$ parses $\pi := (C_e, \pi_{\mathsf{Root}}, \pi_{\mathsf{modEq}}, \pi_{\mathsf{range}})$ and $\mathsf{crs} := (N, G, H, \mathsf{Hprime}, \mathbb{G}_q, g, h, \mathsf{crs}_{\mathsf{range}})$, from which it computes the corresponding $\mathsf{crs}_{\mathsf{Root}} := (N, G, H)$ and $\mathsf{crs}_{\mathsf{modEq}} := (N, G, H, \mathbb{G}_q, g, h)$. Then constructs an adversary $\mathcal{A}_{\mathsf{Root}}$ for $\mathsf{CP}_{\mathsf{Root}}$ Knowledge Soundness that outputs $(C_e, C_P, \mu, \pi_{\mathsf{Root}})$. It is obvious that since $\mathcal{V}$ accepts π then it also accepts π_{Root}, i.e., $\mathsf{CP}_{\mathsf{Root}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{Root}}, (C_e, C_P, \mu), \pi_{\mathsf{Root}}) = 1$. From Knowledge Soundness of $\mathsf{CP}_{\mathsf{Root}}$ we know that there is an extractor $\mathcal{E}_{\mathsf{Root}}$ that outputs (e, r, W) such that $C_e = \pm G^e H^r \pmod N \wedge W^e = C_P \pmod N \wedge e < 2^{\lambda_z+\lambda_s+\mu+2}$. Similarly, $\mathcal{E}$ constructs adversaries $\mathcal{A}_{\mathsf{modEq}}$ and $\mathcal{A}_{\mathsf{range}}$ of protocols $\mathsf{CP}_{\mathsf{modEq}}$ and $\mathsf{CP}_{\mathsf{range}}$ respectively. And similarly there are extractors $\mathcal{E}_{\mathsf{modEq}}$ and $\mathcal{E}_{\mathsf{range}}$ that output (e', e_q, r', r_q) such that $e' = e_q \pmod q \wedge C_{e'} = \pm G^{e'} H^{r'} \pmod N \wedge c_{e_q} = g^{e_q \bmod q} h^{r_q \bmod q}$ and (e'_q, r'_q) such that $c_e = g^{e'_q} h^{r'_q} \wedge 2^{\mu-1} < e'_q < 2^{\mu}$ respectively.

From the Binding property of the integer commitment scheme we get that $e = e'$ and $r = r'$ (over the integers), unless with a negligible probability. Similarly, from the Binding property of the Pedersen commitment scheme we get that $e_q = e'_q \pmod q$ and $r_q = r'_q \pmod q$, unless with a negligible probability. So if we put everything together the extracted values are (e, r, W, e_q, r_q) such that:

$$W^e = C_P \pmod N \wedge e < 2^{\lambda_z+\lambda_s+\mu+2} \wedge e = e_q \pmod q \wedge 2^{\mu-1} < e_q < 2^{\mu}$$

and additionally

$$C_e = \pm G^e H^r \wedge c_e = g^{e_q \bmod q} h^{r_q \bmod q}$$

From $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_S, C_P, P, \varnothing) = 1$ we infer that $C_P = G^{\mathsf{prod}_P}$, where for each $e_i \in P$ it holds that $e \in \mathbb{P}\left(2^{\mu-1}, 2^{\mu}\right)$. From the strong RSA assumption since $W^e = C_P = G^{\mathsf{prod}_P} \pmod N$ we get $e \in \Pi_P$, unless with a negligible probability.

The rest of the analysis that justifies $e \in P$ is identical to the one of the proof of theorem 1. So $e \in P$ and as shown above $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_e, e_q, r_q) = 1$.

Zero Knowledge: For the Zero Knowledge Property we rely on similar techniques with the ones of the proof of theorem 3 except for the use of $\mathcal{S}_{\mathsf{HashEq}}$. Here we use instead the simulator of the $\mathsf{CP}_{\mathsf{range}}$ protocol, $\mathcal{S}_{\mathsf{range}}$. □

4.4.4 Proposed Instantiations of Protocols for R_{Root} and R_{modEq}

4.4.4.1 Protocol $\mathsf{CP}_{\mathsf{Root}}$.

We first give a protocol $\mathsf{CP}_{\mathsf{Root}'}$ for a simpler version of the Root relation in which the upper bound on e is removed; let us call $R_{\mathsf{Root}'}$ this relation.

Below is an interactive ZK protocol for $R_{\mathsf{Root}'}$:

1. Prover computes a W such that $W^e = Acc$ and $C_W = WH^{r_2}, C_r = G^{r_2} H^{r_3}$ and sends to the verifier:

   $\underline{\mathcal{P} \rightarrow \mathcal{V}} : C_W, C_r$

2. Prover and Verifier perform a protocol for the relation:

   $R((C_e, C_r, C_W, Acc), (e, r, r_2, r_3, \beta, \delta)) = 1$ iff

$$C_e = G^e H^r \wedge C_r = G^{r_2} H^{r_3} \wedge Acc = C_W^e \left(\frac{1}{H}\right)^{\beta} \wedge 1 = C_r^e \left(\frac{1}{H}\right)^{\delta} \left(\frac{1}{G}\right)^{\beta}$$

Root protocol

- On common reference string $\mathsf{crs} = (\mathbb{Z}_N^*, G, H)$
- $\mathsf{Prove}(\mathsf{crs}, (C_e, Acc), (e, r, w))$:
  1. samples $r_2, r_3 \leftarrow_\$ (-\lfloor N/4 \rfloor, \lfloor N/4 \rfloor)$ and computes $C_W \leftarrow WH^{r_2}, C_r \leftarrow G^{r_2}H^{r_3}$.
  2. Computes the non-interactive version of the above protocol
     $r_e \leftarrow_\$ \left(-2^{\lambda_z+\lambda_s+\mu}, 2^{\lambda_z+\lambda_s+\mu}\right), r_r, r_{r_2}, r_{r_3} \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right)$
     $r_\beta, r_\delta \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}\right)$
     $\alpha_1 \leftarrow G^{r_e}H^{r_r}, \alpha_2 \leftarrow G^{r_{r_2}}H^{r_{r_3}}, \alpha_3 \leftarrow C_W^{r_e}\left(\frac{1}{H}\right)^{r_\beta}, \alpha_4 \leftarrow C_r^{r_e}(\frac{1}{H})^{r_\delta}\left(\frac{1}{G}\right)^{r_\beta}$
     $c \leftarrow \mathsf{H}(\alpha_1, \alpha_2, \alpha_3, \alpha_4, C_e, Acc)$
     $s_e \leftarrow r_e - ce, s_r \leftarrow r_r - cr, s_{r_2} \leftarrow r_{r_2} - cr_2, s_{r_3} \leftarrow r_{r_3} - cr_{r_3}, s_\beta \leftarrow r_\beta - cer_2, s_\delta \leftarrow r_\delta - cer_3$

  Returns $\pi \leftarrow (C_W, C_r, \alpha_1, \alpha_2, \alpha_3, \alpha_4, s_e, s_r, s_{r_2}, s_{r_3}, s_\beta, s_\delta)$
- $\mathsf{VerProof}(\mathsf{crs}, (C_e, Acc), \pi)$: returns 1 iff $\alpha_1 = C_e^c G^{s_e} H^{s_r} \wedge \alpha_2 = C_r^c G^{s_{r_2}} H^{s_{r_3}} \wedge \alpha_3 = Acc^c C_W^{s_e}\left(\frac{1}{H}\right)^{s_\beta} \wedge \alpha_4 = C_r^{s_e}\left(\frac{1}{H}\right)^{s_\delta}\left(\frac{1}{G}\right)^{s_\beta} \wedge s_e \in \left[-2^{\lambda+\lambda_s+\mu+1}, 2^{\lambda+\lambda_s+\mu+1}\right]$

Figure 4.8: Our Root protocol instantiation.

Let λ_s be the size of the challenge space, λ_z be the statistical security parameter and μ the size of e.

- Prover samples:

$$r_e \leftarrow_\$ \left(-2^{\lambda_z+\lambda_s+\mu}, 2^{\lambda_z+\lambda_s+\mu}\right)$$
$$r_r, r_{r_2}, r_{r_3} \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right)$$
$$r_\beta, r_\delta \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}\right)$$

  and computes:

$$\alpha_1 = G^{r_e}H^{r_r}, \quad \alpha_2 = G^{r_{r_2}}H^{r_{r_3}}, \quad \alpha_3 = C_W^{r_e}\left(\frac{1}{H}\right)^{r_\beta}, \quad \alpha_4 = C_r^{r_e}(\frac{1}{H})^{r_\delta}\left(\frac{1}{G}\right)^{r_\beta}$$

  $\underline{\mathcal{P} \to \mathcal{V}} : (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$
- Verifier samples the challenge $c \leftarrow \{0,1\}^{\lambda_s}$
  $\underline{\mathcal{V} \to \mathcal{P}} : c$
- Prover computes the response:

$$s_e = r_e - ce$$
$$s_r = r_r - cr, \quad s_{r_2} = r_{r_2} - cr_2, \quad s_{r_3} = r_{r_3} - cr_{r_3}$$
$$s_\beta = r_\beta - cer_2, \quad s_\delta = r_\delta - cer_3$$

$\underline{\mathcal{P} \to \mathcal{V}} : (s_e, s_r, s_{r_2}, s_{r_3}, s_\beta, s_\delta)$

- Verifier checks if:

$$\alpha_1 \stackrel{?}{=} C_e^c G^{s_e} H^{s_r}, \alpha_2 \stackrel{?}{=} C_r^c G^{s_{r_2}} H^{s_{r_3}}, \alpha_3 \stackrel{?}{=} Acc^c C_W^{s_e} \left(\frac{1}{H}\right)^{s_\beta}, \alpha_4 \stackrel{?}{=} C_r^{s_e} \left(\frac{1}{H}\right)^{s_\delta} \left(\frac{1}{G}\right)^{s_\beta}$$

Theorem 6. *Let $\mathbb{Z}_N^*$ be an RSA group where strong-RSA assumption holds, then the above protocol is a correct, knowledge sound and honest-verifier zero knowledge protocol for $R_{\mathsf{Root}'}$.*

The proof of the above is similar to the one of [57] where the more specific protocol was introduced, but implicitly was including a protocol for $R_{\mathsf{Root}'}$. Before proceeding to the proof we recall some properties related to RSA groups. First we expose two standard arguments. The first is that obtaining a multiple of $\phi(N)$ is equivalent to factoring N. This directly allows us to argue that for any $G \in \mathbb{Z}_N^*$, if one is able to find an $x \in \mathbb{Z}$ such that $G^x = 1 \pmod N$ then under the factoring assumption $x = 0$, otherwise x is a multiple of $\phi(N)$. Secondly, finding any non-trivial solution of the equation $\mu^2 = 1 \pmod N$ in $\mathbb{Z}_N^*$ (non-trivial means $\mu \neq \pm 1$) is equivalent to factoring N.

Remark 12. *In 2017 Couteau et al. proved that in fact knowledge soundness for the protocol of opening an integer commitment can be reduced to (plain) RSA problem [81]. This could be inherited to our protocol too. However, the relation itself assumes strong RSA's hardness, otherwise finding a root would be computable in polynomial time. Additionally, in the reduction to (plain) RSA, the extractor's probability of success is cubic, while in the reduction to strong RSA linear, in the adversary's probability of success.*

Proposition 1. *Let $\mathbb{Z}_N^*$ be an RSA group with a modulus N and QR_N the corresponding group of quadratic residues modulo N.*

1. *Let $G, H \leftarrow_\$ \mathsf{QR}_N$ two random generators of QR_N and a PPT adversary $\mathcal{A}$ outputting $\alpha, \beta \in \mathbb{Z}_N^*$ such that $G^\alpha H^\beta = 1$ then under the assumption that DLOG problem is hard in QR_N it holds that $\alpha = \beta = 0$.*
2. *Let $A, B \in \mathbb{Z}_N^*$ and a PPT adversary $\mathcal{A}$ outputting $x, y \in \mathbb{Z}_N^*$ such that $A^y = B^x$ and $y \mid x$ then under the assumption that factoring of N is hard it holds that $A = \pm B^{\frac{x}{y}}$*

Proof. 1. Since $G, H \in \mathsf{QR}_N$ there is an $x \in \mathbb{Z}_N^*$ such that $G = H^x \pmod N$ which leads to $H^{x\alpha+\beta} = 1$. As we discussed above under the assumption that factoring of N is hard, $x\alpha + \beta = 0$. If $\alpha \neq 0$ then $x \leftarrow -\frac{\beta}{\alpha}$ is a discrete logarithm of H, so assuming that DLOG is hard $\alpha = 0$. Similarly, there is an $y \in \mathbb{Z}_N^*$ such that $G^y = H \pmod N$ and with a similar argument we can conclude that $\beta = 0$.

2. We discern two cases, $y = \rho$ is odd or $y = 2^v\rho$ is even (for an odd ρ). In case y is odd then it is co-prime with $\phi(N) = p'q'$ (otherwise if $y = p'$ or $y = q'$ we would be able to factor N), so $y^{-1} \pmod{\phi(N)}$ exists and $A = B^{\frac{x}{y}}$. If $y = 2^v\rho$ then $\left(A^{-1}B^{\frac{x}{y}}\right)^y = 1 \Rightarrow \left(A^{-1}B^{\frac{x}{y}}\right)^{2^v\rho} = 1 \Rightarrow \left(A^{-1}B^{\frac{x}{y}}\right)^{2^v} = 1$. From the second fact that we discussed above under the factoring assumption $\left(A^{-1}B^{\frac{x}{y}}\right)^{2^{v-1}} = \pm 1$. However for $v > 1$ the left part of the equation is a quadratic residue so it cannot be -1, therefore $\left(A^{-1}B^{\frac{x}{y}}\right)^{2^{v-1}} = 1$.

Using the same facts repeatedly we will eventually conclude that $\left(A^{-1}B^{\frac{x}{y}}\right)^2 = 1$, hence $A^{-1}B^{\frac{x}{y}} = \pm 1 \Rightarrow A = \pm B^{\frac{x}{y}}$.

□

proof of theorem 6. Correctness is straightforward. Honest-verifier zero knowledge can be shown with standard arguments used in Σ-protocols and the fact that the commitments to C_e, C_W, C_r are statistically hiding. That is the simulator $\mathcal{S}$ on input (C_e, Acc) samples $C^*_W \leftarrow_\$ \mathbb{Z}^*_N$ and $C^*_r \leftarrow_\$ \mathbb{Z}^*_N$. Then samples

$$s^*_e \leftarrow_\$ \left(-2^{\lambda_z+\lambda_s+\mu} - 2^{\lambda_z+\mu}, 2^{\lambda_z+\lambda_s+\mu} + 2^{\lambda_z+\mu}\right),$$

$$s^*_r, s^*_{r_2}, s^*_{r_3} \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s} - \lfloor N/4 \rfloor 2^{\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s} + \lfloor N/4 \rfloor 2^{\lambda_s}\right),$$

$$s^*_\beta, s^*_\delta \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu} - \lfloor N/4 \rfloor 2^{\lambda_s+\mu}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu} + \lfloor N/4 \rfloor 2^{\lambda_s+\mu}\right).$$

Finally it samples $c^* \leftarrow_\$ \{0,1\}^{\lambda_s}$. Then it sets $\alpha^*_1 \leftarrow C^c_e G^{s_e} H^{s_r}$, $\alpha^*_2 \leftarrow C^c_r G^{s_{r_2}} H^{s_{r_3}}$, $\alpha^*_3 \leftarrow \mathsf{Acc}^c C^{s_e}_W \left(\frac{1}{H}\right)^{s_\beta}$ and $\alpha^*_4 \stackrel{?}{=} C^{s_e}_r \left(\frac{1}{H}\right)^{s_\delta} \left(\frac{1}{G}\right)^{s_\beta}$. $\mathcal{S}$ outputs $\pi^* \leftarrow (C^*_W, C^*_r, \alpha^*_1, \alpha^*_2, \alpha^*_3, \alpha^*_4, c^*, s^*_e, s^*_r, s^*_{r_2}, s^*_{r_3}, s^*_\beta, s^*_\delta)$. The distribution of π^* is identical to the one of a real proof π.

For the knowledge soundness, let an adversary of the knowledge soundness $\mathcal{A}$ that is able to convince the verifier $\mathcal{V}$ with a probability at least ϵ. We will construct an extractor $\mathcal{E}$ that extracts the witness $(e, r, r_2, r_3, \beta, \delta)$. Using rewinding $\mathcal{E}$ gets two accepted transcripts

$$(C_W, C_r, \alpha_1, \alpha_2, \alpha_3, \alpha_4, c, s_e, s_r, s_{r_2}, s_{r_3}, s_\beta, s_\delta) \text{ and}$$
$$(C_W, C_r, \alpha_1, \alpha_2, \alpha_3, \alpha_4, c', s'_e, s'_r, s'_{r_2}, s'_{r_3}, s'_\beta, s'_\delta)$$

on two different challenges c and c'. $\mathcal{E}$ aborts if it cannot get two such transcripts (abort1).

We denote $\Delta c \coloneqq c' - c, \Delta s_e \coloneqq s_e - s'_e, \Delta s_r \coloneqq s_r - s'_r, \Delta s_{r_2} \coloneqq s_{r_2} - s'_{r_2}, \Delta s_{r_3} \coloneqq s_{r_3} - s'_{r_3}, \Delta s_\beta \coloneqq s_\beta - s'_\beta, \Delta s_\delta \coloneqq s_\delta - s'_\delta$ then

$$C^{\Delta c}_e = G^{\Delta s_e} H^{\Delta s_r}, \quad C^{\Delta c}_r = G^{\Delta s_{r_2}} H^{\Delta s_{r_3}},$$
$$\mathsf{Acc}^{\Delta c} = C^{\Delta s_e}_W \left(\frac{1}{H}\right)^{\Delta s_\beta}, \quad 1 = C^{\Delta s_e}_r \left(\frac{1}{H}\right)^{\Delta s_\delta} \left(\frac{1}{G}\right)^{\Delta s_\beta}$$

Define the (possibly rational) numbers $\hat{e} \coloneqq \frac{\Delta s_e}{\Delta c}, \hat{r} \coloneqq \frac{\Delta s_r}{\Delta c}, \hat{r_2} \coloneqq \frac{\Delta s_{r_2}}{\Delta c}, \hat{r_3} \coloneqq \frac{\Delta s_{r_3}}{\Delta c}$. In case Δc doesn't divide Δs_e and Δs_r, $\mathcal{E}$ aborts (abort 2a). Similarly, in case Δc doesn't divide Δs_{r_2} and Δs_{r_3}, $\mathcal{E}$ aborts (abort 2b). Therefore, since the above aborts didn't happen and according to second point of proposition 1, $C_e = \pm G^{\hat{e}} H^{\hat{r}}$ and $C_r = \pm G^{\hat{r_2}} H^{\hat{r_3}}$.

Now if we replace C_r in the fourth equation we get $1 = (\pm 1)^{\Delta s_e} G^{\hat{r_2}\Delta s_e} H^{\hat{r_3}\Delta s_e} \left(\frac{1}{H}\right)^{\Delta s_\delta} \left(\frac{1}{G}\right)^{\Delta s_\beta}$ or $(\pm 1)^{\Delta s_e} G^{\hat{r_2}\Delta s_e - \Delta s_\beta} H^{\hat{r_3}\Delta s_e - \Delta s_\delta} = 1$. However, $(\pm 1)^{\Delta s_e} = 1$ otherwise if $(\pm 1)^{\Delta s_e} = -1$ then $-G^{\hat{r_2}\Delta s_e - \Delta s_\beta} H^{\hat{r_3}\Delta s_e - \Delta s_\delta}$ would be a non-quadratic residue (since G, H are both in QR_N and QR_N is closed under multiplication) equal to 1 which is a quadratic residue and this would be a contradiction, hence $G^{\hat{r_2}\Delta s_e - \Delta s_\beta} H^{\hat{r_3}\Delta s_e - \Delta s_\delta} = 1$. According to the first point of proposition 1, under the factoring assumption $\hat{r_2}\Delta s_e - \Delta s_\beta = \hat{r_3}\Delta s_e - \Delta s_\delta = 0$, so $\hat{r_2}\Delta s_e = \Delta s_\beta$.

Finally we replace Δs_β in the third equation and we get $Acc^{\Delta c} = C^{\Delta s_e}_W \left(\frac{1}{H}\right)^{\hat{r_2}\Delta s_e} \Rightarrow Acc^{\Delta c} = \left(\frac{C_w}{H^{\hat{r_2}}}\right)^{\Delta s_e}$. As stated above Δc divides Δs_e so according to the second point of proposition 1 $\mathsf{Acc} = \pm \left(\frac{C_W}{H^{\hat{r_2}}}\right)^{\frac{\Delta s_e}{\Delta c}} = \pm \left(\frac{C_W}{H^{\hat{r_2}}}\right)^{\hat{e}}$. We discern three cases:

- $\mathsf{Acc} = +\left(\frac{C_W}{H^{\tilde{r}_2}}\right)^{\frac{\Delta s_e}{\Delta c}}$: Then $\mathcal{E}$ sets $\tilde{W} \leftarrow \frac{C_W}{H^{\tilde{r}_2}}$ and $\tilde{e} \leftarrow \hat{e} := \frac{\Delta s_e}{\Delta c}$ $\tilde{r} \leftarrow \hat{r} := \frac{\Delta s_r}{\Delta c}$ as above. It is clear that $\mathsf{Acc} = \tilde{W}^{\tilde{e}}$ and as stated above $C_e = G^{\tilde{e}} H^{\tilde{r}}$.
- $\mathsf{Acc} = -\left(\frac{C_W}{H^{\tilde{r}_2}}\right)^{\frac{\Delta s_e}{\Delta c}}$ and $\frac{\Delta s_e}{\Delta c}$ odd: Then $\mathcal{E}$ sets $\tilde{W} \leftarrow -\frac{C_W}{H^{\tilde{r}_2}}$ and $\tilde{e} \leftarrow \hat{e} := \frac{\Delta s_e}{\Delta c}$ $\tilde{r} \leftarrow \hat{r} := \frac{\Delta s_r}{\Delta c}$ as above. It is clear that $\mathsf{Acc} = \tilde{W}^{\tilde{e}}$ and as stated above $C_e = G^{\tilde{e}} H^{\tilde{r}}$.
- $\mathsf{Acc} = -\left(\frac{C_W}{H^{\tilde{r}_2}}\right)^{\frac{\Delta s_e}{\Delta c}}$ and $\frac{\Delta s_e}{\Delta c}$ even: this means that Acc is a non-quadratic residue, which is a contradiction since in the $R_{\mathsf{Root}'}$ relation we assume that $\mathsf{Acc} \in \mathsf{QR}_N$.

Finally the $\mathcal{E}$ outputs $(\tilde{e}, \tilde{r}, \tilde{W})$.

Now we show that the probability the extractor terminates with outputting a valid witness is $O(\epsilon)$. If the extractor does not abort then it clearly outputs a valid witness (under factoring assumption). For the first abort, with a standard argument it can be shown that the extractor is able to extract two accepting transcripts with probability $O(\epsilon)$ (for the probabilistic analysis we refer to [85]). Thus $Pr[\mathsf{abort1}] = 1 - \mathsf{O}(\epsilon)$. For the second type of aborts (abort 2a and abort 2b), they happen with negligible probability under the strong RSA assumption. For the details see lemma 6 below, which was proven in [85]. Putting them together the probability of success of $\mathcal{E}$ is at least $O(\epsilon) - \mathsf{negl}(\lambda_s)$.

Lemma 6 ([85])**.** *Given that* abort 2a *occurs a PPT adversary $\mathcal{B}$ can solve the strong RSA problem with probability at least $\frac{1}{2} - 2^{-\lambda_s}$.*

From the above we get $Pr[\mathcal{B} \text{ solves } sRSA] \geq \left(\frac{1}{2} - 2^{-\lambda_s}\right) Pr[\mathsf{abort\ 2a}]$, so we conclude to $Pr[\mathsf{abort\ 2a}] \leq \frac{1}{\frac{1}{2} - 2^{-\lambda_s}} \mathsf{Pr}[\mathcal{B} \text{ solves sRSA}] = \mathsf{negl}(\lambda_s)$. The same lemma holds for abort 2b. □

Notice in the above protocol that

$$-2^{\lambda_z+\lambda_s+\mu} - 2^{\lambda_s+\mu} \leq s_e \leq 2^{\lambda_z+\lambda_s+\mu} + 2^{\lambda_s+\mu} \Rightarrow$$
$$-2^{\lambda_z+\lambda_s+\mu+1} \leq s_e \leq 2^{\lambda_z+\lambda_s+\mu+1} \Rightarrow$$
$$-2^{\lambda_z+\lambda_s+\mu+2} \leq \Delta s_e \leq 2^{\lambda_z+\lambda_s+\mu+2} \Rightarrow$$
$$-2^{\lambda_z+\lambda_s+\mu+2} \leq \hat{e} \leq 2^{\lambda_z+\lambda_s+\mu+2}$$

so if we impose an additional verification check of honest s_e size, i.e., $s_e \in \left[-2^{\lambda_z+\lambda_s+\mu+1}, 2^{\lambda_z+\lambda_s+\mu+1}\right]$, we get that $|\hat{e}| \leq 2^{\lambda_z+\lambda_s+\mu+2}$. The verifier performs an extra range check $s_e \overset{?}{\in} \left[-2^{\lambda_z+\lambda_s+\mu+1}, 2^{\lambda_z+\lambda_s+\mu+1}\right]$ and the resulting protocol is the $\mathsf{CP}_{\mathsf{Root}}$ that except for proving of knowledge of an e-th root also provides a bound for the size of $|e|$:

$$R_{\mathsf{Root}}\left((C_e, Acc, \mu), (e, r, W)\right) = 1 \text{ iff}$$
$$C_e = \pm G^e H^r \pmod{N} \wedge W^e = Acc \pmod{N} \wedge |e| < 2^{\lambda_z+\lambda_s+\mu+2}$$

4.4.4.2 Protocol $\mathsf{CP}_{\mathsf{modEq}}$.

Below we describe the public-coin ZK protocol for R_{modEq}. In Figure 4.9 we summarize the corresponding NIZK obtained after applying the Fiat-Shamir transform to it.

1. Prover samples:
$$r_e \leftarrow \left(-2^{\lambda_z+\lambda_s+\mu}, 2^{\lambda_z+\lambda_s+\mu}\right)$$
$$r_r \leftarrow \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right)$$
$$r_{r_q} \leftarrow \mathbb{Z}_q$$
and computes:
$$\alpha_1 = G^{r_e} H^{r_r}, \quad \alpha_2 = g^{r_e} \pmod p h^{r_{r_q}}$$
$\underline{\mathcal{P} \rightarrow \mathcal{V}}$: (α_1, α_2)
2. Verifier samples the challenge $c \leftarrow \{0,1\}^{\lambda_s}$
$\underline{\mathcal{V} \rightarrow \mathcal{P}}$: c
3. Prover computes the response:
$$s_e = r_e - ce$$
$$s_r = r_r - cr$$
$$s_{r_q} = r_{r_q} - cr_q \pmod q$$
$\underline{\mathcal{P} \rightarrow \mathcal{V}}$: (s_e, s_r, s_{r_q})
4. Verifier checks if:
$$\alpha_1 \stackrel{?}{=} \pm C_e^c G^{s_e} H^{s_r} \pmod N, \alpha_2 \stackrel{?}{=} c_{e_q}^c g^{s_e} \pmod q h^{s_{r_q}}$$

Theorem 7. *Let $\mathbb{Z}_N^*$ be an RSA group where strong-RSA assumption holds and $\mathbb{G}$ be a prime order group where DLOG assumption holds then the above protocol is a correct, knowledge sound and honest-verifier zero knowledge protocol for R_{modEq}.*

The proof is quite simple and is omitted.

4.4.5 Instantiations

We discuss the possible instantiations of our schemes $\mathsf{MemCP}_{\mathsf{RSA}}$ and $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ that can be obtained by looking at applications' constraints and security parameters constraints.

Parameters for $d\mu + 2 \leq \nu$ and $\mu \leq \nu - 2$. First we analyze possible parameters that satisfy the conditions $d\mu + 2 \leq \nu \wedge \mu \leq \nu - 2$ that is used in Theorems 1 and 2; we recall $d = 1 + \lfloor \frac{\lambda_z+\lambda_s+2}{\mu} \rfloor$, where λ_z and λ_s are statistical security parameters for zero-knowledge and soundness respectively of $\mathsf{CP}_{\mathsf{Root}}$.

If the prime order group $\mathbb{G}_q$ is instantiated with (pairing-friendly) elliptic curves, then the bitsize ν of its order must be at least 2λ. And recall that for correctness we need $\mu < \nu$.

modEq protocol

- On common reference string $\mathsf{crs} = (\mathbb{Z}_N^*, G, H, \mathbb{G}, g, h)$
- $\mathsf{Prove}(\mathsf{crs}, (C_e, c_{e_q}), (e, e_q, r, r_q))$:
  $r_e \leftarrow_\$ \left(-2^{\lambda_z+\lambda_s+\mu}, 2^{\lambda_z+\lambda_s+\mu}\right), r_r \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right), r_{r'} \leftarrow_\$ \mathbb{Z}_q$
  $\alpha_1 \leftarrow G^{r_e} H^{r_r}, \alpha_2 \leftarrow g^{r_e \pmod q} h^{r_{r'}}$
  $c \leftarrow \mathsf{H}(\alpha_1, \alpha_2, C_e, c_{e_q})$
  $s_e \leftarrow r_e - ce, s_r \leftarrow r_r - cr, s_{r'} \leftarrow r_{r_q} - cr_q \pmod q$
  Returns $\pi \leftarrow (\alpha_1, \alpha_2, s_e, s_r, s_{r_q})$
- $\mathsf{VerProof}(\mathsf{crs}, (C_e, c_{e_q}), \pi)$: returns 1 iff $\alpha_1 = \pm C_e^c G^{s_e} H^{s_r} \pmod N \wedge \alpha_2 = c_{e_q}^c g^{s_e \pmod q} h^{s_{r_q}}$

Figure 4.9: Our modEq protocol instantiation.

Considering these constraints, one way to satisfy $d\mu + 2 \leq \nu$ is to choose μ such that $\nu - 1 > \mu > \lambda_z + \lambda_s + 2$. More specifically, a choice that maximizes security is $\nu = 2\lambda$, $\mu = 2\lambda - 2$ and $\lambda_z = \lambda - 3, \lambda_s = \lambda - 2$. For the case of the $\mathsf{MemCP}_{\mathsf{RSA}}$ scheme, this choice yields an instantiation with nearly λ bits of security and where the function H does not necessarily need to be a random oracle (yet it must be collision resistant).

Because of the constraint $\mu > \lambda_z + \lambda_s + 2$, we the choice above implies the use of large primes. This would be anyway the case if one instantiates the scheme with a collision-resistant hash function H (e.g., SHA256 or SHA3), e.g., because set elements are quite arbitrary. If on the other hand, one could support more specific set elements, one could use instead a deterministic map-to-primes or even use our scheme $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ in which set elements themselves are primes. In this case one may wonder if it is possible to choose values of μ smaller than 2λ; for example $\mu \approx 30, 60, 80$. The answer is positive although the characterization of such μ's require an involved analysis.

Let us fix $\nu = 2\lambda$, and say that the statistical security parameters λ_z, λ_s are such that $\lambda_z + \lambda_s + 2 = 2\lambda - 2 - c$ for some constant c (for example $c = 4$ if $\lambda_z = \lambda_s = \lambda - 4$). We are essentially looking for μ such that

$$\mu \leq 2\lambda - 2 - c \text{ and } \mu + \mu \left\lfloor \frac{2\lambda - 2}{\mu} - \frac{c}{\mu} \right\rfloor \leq 2\lambda - 2$$

$$\Leftrightarrow \mu \leq 2\lambda - 2 - c \text{ and } \left\lfloor \frac{2\lambda - 2}{\mu} - \frac{c}{\mu} \right\rfloor \leq \frac{2\lambda - 2}{\mu} - 1$$

From the fact $x \mod y = x - y\lfloor \frac{x}{y} \rfloor$, we can reduce the above inequality into

$$\mu \leq 2\lambda - 2 - c \text{ and } 2\lambda - 2 - c \bmod \mu \geq \mu - c$$

that can admit solutions for $c \geq 2$.

For instance, if $\lambda = 128$ and $c = 4$, then we get several options for μ, e.g., $\mu = 32, 42, 63, 84, 126, 127$.

Parameters for $d\mu + 2 > \nu$. This case concerns only $\mathsf{MemCP}_{\mathsf{RSA}}$ and Theorem 2 in particular. In this case, if one aims at maximizing security, say to get a scheme with λ-bits of security, then would have to set $\mu \approx 2\lambda$ for collision resistance, and consequently select the prime order group so that $\nu \geq 3\lambda$. This choice however is costly in terms of performance since the efficiency of all protocols that work in the prime order group degrades.

4.5 A CP-SNARK for Set Non-Membership with Short Parameters

Here we describe two CP-SNARKs for set non-membership that work in a setting identical to the one of section 4.4. Namely, the set is committed using an RSA accumulator, and the element (that one wants to prove not to belong to the set) is committed using a Pedersen commitment scheme. As in the previous section, we propose two protocols for non-membership, called $\mathsf{NonMemCP}_{\mathsf{RSA}}$ and $\mathsf{NonMemCP}_{\mathsf{RSAPrm}}$, in complete analogy to $\mathsf{MemCP}_{\mathsf{RSA}}$ and $\mathsf{MemCP}_{\mathsf{RSAPrm}}$. In the former, the elements of the set are arbitrary bit-strings of length η, $\mathcal{D}_u = \{0,1\}^\eta$, while in the latter the elements are primes of length μ. The schemes are fully described in figures 4.10 and 4.11.

An High-Level Overview of the Constructions. The main idea of $\mathsf{NonMemCP}_{\mathsf{RSA}}$ is similar to the one of the corresponding membership protocol, $\mathsf{MemCP}_{\mathsf{RSA}}$. It uses in the same modular way the modEq and HashEq protocols. The only difference lies in the third protocol: instead of using Root it uses a new protocol $\mathsf{Coprime}$. In a similar manner, $\mathsf{NonMemCP}_{\mathsf{RSAPrm}}$ uses modEq, range and $\mathsf{Coprime}$.

Let us explain the need of the $\mathsf{Coprime}$ protocol and what it does. First, recall how a non-membership proof is computed in RSA Accumulators [147]. Let P be a set of primes to be accumulated and prod the corresponding product. For any prime element $e \notin P$ it holds that $\gcd(e, \mathsf{prod}) = 1$, while for any member $e \in P$ it is $\gcd(e, \mathsf{prod}) = e \neq 1$. Thus, proving that $\gcd(e, \mathsf{prod}) = 1$ would exhibit non-membership of e in P. Recall, also, that using the extended Euclidean algorithm one can efficiently compute coefficients (a, b) such that $a \cdot e + b \cdot \mathsf{prod} = \gcd(e, \mathsf{prod})$. A non-membership proof for an element e w.r.t. an accumulator $\mathsf{Acc} = G^{\mathsf{prod}}$ consists of a pair $(D = G^a, b)$, where a, b are such that $a \cdot e + b \cdot \mathsf{prod} = 1$. The verification is $D^e \mathsf{Acc}^b = G$, which ensures that e and prod are coprime, i.e. $\gcd(e, \mathsf{prod}) = 1$. Therefore, the goal of the $\mathsf{Coprime}$ protocol is to prove knowledge of an element e committed in an integer commitment C_e that satisfies this relation. A more formal definition of $\mathsf{Coprime}$ is given below and an instantiation of this protocol is in Section 4.5.1.

Argument of Knowledge for a coprime element. We make use of a non-interactive argument of knowledge of a non-membership witness of an element such that the verification equation explained above holds. More formally $\mathsf{CP}_{\mathsf{Coprime}}$, is a NIZK for the relation: $R_{\mathsf{Coprime}} : (\mathbb{Z}_N^* \times \mathsf{QR}_N) \times (\mathbb{Z} \times \mathbb{Z} \times \mathsf{QR}_N \times \mathbb{Z})$ defined as
$R_{\mathsf{Coprime}}((C_e, \mathsf{Acc}), (e, r, D, b)) = 1$ iff

$$C_e = \pm G^e H^r \bmod N \ \wedge \ D^e \mathsf{Acc}^b = G \wedge |e| < 2^{\lambda_z + \lambda_s + \mu + 2}$$

We propose an instantiation of a protocol for the above relation in the Section 4.5.1.

Our Constructions of $\mathsf{NonMemCP}_{\mathsf{RSA}}$ and $\mathsf{NonMemCP}_{\mathsf{RSAPrm}}$. In Figures 4.10 and 4.11 we give a full description of the schemes.

$\mathsf{MemCP}_{\mathsf{RSA}}$ protocol

- $\mathsf{KeyGen}(\mathsf{ck}, R^{\in})$: parse $\mathsf{ck} := ((N, G, \mathsf{Hprime}), (\mathbb{G}_q, g, h))$ as the commitment keys of $\mathsf{SetCom}_{\mathsf{RSA}}$ and PedCom respectively. Sample a random generator H.

  Generate $\mathsf{crs}_{\mathsf{HashEq}} \leftarrow\!\!\$\ \mathsf{CP}_{\mathsf{HashEq}}.\mathsf{Setup}((\mathbb{G}_q, g, h), R_{\mathsf{HashEq}})$, a crs for $\mathsf{CP}_{\mathsf{HashEq}}$.

  Return $\mathsf{crs} := (N, G, H, \mathsf{Hprime}, \mathbb{G}_q, g, h, \mathsf{crs}_{\mathsf{HashEq}})$.

  Given crs, one can define $\mathsf{crs}_{\mathsf{Coprime}} := (N, G, H)$, $\mathsf{crs}_{\mathsf{modEq}} := (N, G, H, \mathbb{G}_q, g, h)$.

- $\mathsf{Prove}(\mathsf{crs}, (C_S, c_u), (S, u), (\varnothing, r_u))$: Compute $e \leftarrow \mathsf{Hprime}(u) = (1|\mathsf{H}(u, j))$, $(c_e, r_q) \leftarrow \mathsf{Com}_1.\mathsf{Commit}(\mathsf{ck}, \mathsf{t}_q, e)$.

  $(C_e, r) \leftarrow \mathsf{IntCom.Commit}((G, H), e)$; $P \leftarrow \{\mathsf{Hprime}(u) : u \in S\}$, compute a, b s.t. $a \cdot e + b \cdot \prod_{e_i \in P} e_i = 1$ and set $D = G^a$.

  $\pi_{\mathsf{Coprime}} \leftarrow \mathsf{CP}_{\mathsf{Coprime}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{Coprime}}, (C_e, C_S, \mu), (e, r, D, b))$

  $\pi_{\mathsf{modEq}} \leftarrow \mathsf{CP}_{\mathsf{modEq}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{modEq}}, (C_e, c_e), (e, e, r, r_q))$

  $\pi_{\mathsf{HashEq}} \leftarrow \mathsf{CP}_{\mathsf{HashEq}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{HashEq}}, (c_e, c_u), (e, u), (r_q, r_u), j)$

  Return $\pi := (C_e, c_e, \pi_{\mathsf{Root}}, \pi_{\mathsf{modEq}}, \pi_{\mathsf{HashEq}})$.

- $\mathsf{VerProof}(\mathsf{crs}, (C_S, c_u), \pi)$: Return 1 iff
  $\mathsf{CP}_{\mathsf{Root}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{Coprime}}, (C_e, C_S, \mu), \pi_{\mathsf{Coprime}}) = 1 \wedge$
  $\mathsf{CP}_{\mathsf{modEq}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{modEq}}, (C_e, c_e), \pi_{\mathsf{modEq}}) = 1 \wedge$
  $\mathsf{CP}_{\mathsf{HashEq}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{HashEq}}, (c_e, c_u), \pi_{\mathsf{HashEq}}) = 1$.

Figure 4.10: $\mathsf{NonMemCP}_{\mathsf{RSA}}$ CP-SNARK for set non-membership

The security of these schemes follow very closely the one of the corresponding membership schemes given in Section 4.4. Below we give the Theorems that state their security. The proofs are omitted since they are almost identical to the corresponding proofs for the membership schemes.

Theorem 8. *Let* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}}$ *and* IntCom *be computationally binding commitments,* $\mathsf{CP}_{\mathsf{Coprime}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{HashEq}}$ *be knowledge-sound NIZK arguments, and assume that the Strong RSA assumption hold, and that* H *is collision resistant. If* $d\mu + 2 \leq \nu$, $\lambda_s + 1 < \mu$ *and* $\lambda_s < \log(N)/2$ *then* $\mathsf{NonMemCP}_{\mathsf{RSA}}$ *is knowledge-sound with partial opening of the set commitments* C_S.

Theorem 9. *Let* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}}$ *and* IntCom *be computationally binding commitments,* $\mathsf{CP}_{\mathsf{Coprime}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{HashEq}}$ *be knowledge-sound NIZK arguments, and assume that the Strong RSA assumption hold, and that* H *is collision resistant. If* $d\mu + 2 > \nu$, $\lambda_s + 1 < \mu$, $\lambda_s < \log(N)/2$, $d = O(1)$ *is a small constant,* $2^{\mu-\nu} \in \mathsf{negl}(\lambda)$ *and* H *is modeled as a random oracle, then* $\mathsf{NonMemCP}_{\mathsf{RSA}}$ *is knowledge-sound with partial opening of the set commitments* C_S.

Theorem 10. *Let* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}'}$ *and* IntCom *be computationally binding commitments,* $\mathsf{CP}_{\mathsf{Coprime}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{range}}$ *be knowledge-sound NIZK arguments, and assume*

$\mathsf{MemCP}_{\mathsf{RSA}}$ protocol

- $\mathsf{KeyGen}(\mathsf{ck}, R^{\in})$: parse $\mathsf{ck} := ((N, G, \mathsf{Hprime}), (\mathbb{G}_q, g, h))$ as the commitment keys of $\mathsf{SetCom}_{\mathsf{RSA}'}$ and PedCom respectively. Sample a random generator H.

  Generate $\mathsf{crs}_{\mathsf{range}} \leftarrow\!\!\$\ \mathsf{CP}_{\mathsf{range}}.\mathsf{Setup}((\mathbb{G}_q, g, h), R_{\mathsf{range}})$, a crs for $\mathsf{CP}_{\mathsf{range}}$.

  Return $\mathsf{crs} := (N, G, H, \mathsf{Hprime}, \mathbb{G}_q, g, h, \mathsf{crs}_{\mathsf{range}})$.

  Given crs, one can define $\mathsf{crs}_{\mathsf{Coprime}} := (N, G, H)$, $\mathsf{crs}_{\mathsf{modEq}} := (N, G, H, \mathbb{G}_q, g, h)$.

- $\mathsf{Prove}(\mathsf{crs}, (C_P, c_e), (P, e), (\varnothing, r_e)) : (C_e, r) \leftarrow \mathsf{IntCom.Commit}((G, H), e)$; compute a, b s.t. $a \cdot e + b \cdot \prod_{e_i \in P} e_i = 1$ and set $D = G^a$.

  $\pi_{\mathsf{Coprime}} \leftarrow \mathsf{CP}_{\mathsf{Coprime}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{Coprime}}, (C_e, C_P, \mu), (e, r, D, b))$

  $\pi_{\mathsf{modEq}} \leftarrow \mathsf{CP}_{\mathsf{modEq}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{modEq}}, (C_e, c_e), (e, e, r, r_q))$

  $\pi_{\mathsf{range}} \leftarrow \mathsf{CP}_{\mathsf{range}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{range}}, (2^{\mu-1}, 2^{\mu}), c_e, e, r_q)$

  Return $\pi := (C_e, c_e, \pi_{\mathsf{Coprime}}, \pi_{\mathsf{modEq}}, \pi_{\mathsf{range}})$.

- $\mathsf{VerProof}(\mathsf{crs}, (C_P, c_e), \pi)$: Return 1 iff
  $\mathsf{CP}_{\mathsf{Coprime}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{Coprime}}, (C_e, C_P, \mu), \pi_{\mathsf{Coprime}}) = 1 \wedge$
  $\mathsf{CP}_{\mathsf{modEq}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{modEq}}, (C_e, c_e), \pi_{\mathsf{modEq}}) = 1 \wedge$
  $\mathsf{CP}_{\mathsf{range}}.\mathsf{VerProof}(\mathsf{crs}_{\mathsf{range}}, c_e, \pi_{\mathsf{range}}) = 1$.

Figure 4.11: $\mathsf{NonMemCP}_{\mathsf{RSAPrm}}$ CP-SNARK for set non-membership

that the Strong RSA assumption hold. If $d\mu + 2 \leq \nu$, $\lambda_s + 1 < \mu$ *and* $\lambda_s < \log(N)/2$ *then* $\mathsf{NonMemCP}_{\mathsf{RSAPrm}}$ *is knowledge-sound with partial opening of the set commitments* c_P*. Furthermore, if* PedCom, $\mathsf{SetCom}_{\mathsf{RSA}'}$ *and* IntCom *are statistically hiding commitments, and* $\mathsf{CP}_{\mathsf{Coprime}}$, $\mathsf{CP}_{\mathsf{modEq}}$ *and* $\mathsf{CP}_{\mathsf{range}}$ *be zero-knowledge, then* $\mathsf{NonMemCP}_{\mathsf{RSAPrm}}$ *is zero-knowledge.*

4.5.1 Proposed Instantiations of Protocol for R_{Coprime}

Below we propose an interactive ZK protocol for R_{Coprime}. As the relation indicates, we need to prove knowledge of (D, b) such that $D^e \mathsf{Acc}^b = G$, for a committed e. Proving opening of C_e to e is straightforward, so the main challenge is to prove the non-membership equation. For this the prover should send D and Acc^b to the verifier so that she can check that $D^e \mathsf{Acc}^b = G$ herself. Of course, there are two caveats. The first one is that D and Acc^b cannot be sent in the plain as we require zero-knowledge; we solve this by sending them in a hiding manner, i.e., $C_a = DH^{r_a}$ and $C_B = \mathsf{Acc}^b H^{\rho_B}$ for random values r_a, ρ_B. Consequently, the verification now should work with the hiding elements. Secondly, the verifier should be ensured that Acc^b is indeed an exponentiation of Acc with a known (to the prover) value b, otherwise soundness can be broken. More specifically we require extraction of b, ρ_B such that $C_B = \mathsf{Acc}^b H^{\rho_B}$. This is done using the partial opening of Acc to the set represented by prod, i.e., the protocol assumes that $\mathsf{Acc} = G^{\mathsf{prod}}$ is a common knowledge.

Below we present our protocol in full details.

1. Prover computes $C_a = DH^{r_a}, C_{r_a} = G^{r_a}H^{r'_a}, C_B = Acc^b H^{\rho_B}, C_{\rho_B} = G^{\rho_B}H^{\rho'_B}$ and sends to the verifier:

   $\underline{\mathcal{P} \to \mathcal{V}} : C_a, C_{r_a}, C_B, C_{\rho_B}$

2. Prover and Verifier perform a protocol for the relation:

   $R((\mathsf{Acc}, C_e, C_a, C_{r_a}, C_B, C_{\rho_B}), (e, b, r, r_a, r'_a, \rho_B, \rho'_B, \beta, \delta)) = 1$ iff

   $$C_B = Acc^b H^{\rho_B} \wedge C_e = G^e H^r \wedge C_{r_a} = G^{r_a} H^{r'_a}$$
   $$\wedge C_{\rho_B} = G^{\rho_B} H^{\rho'_B} \wedge C_a^e C_B = GH^{\beta} \wedge C_{r_a}^e C_{\rho_B} = G^{\beta} H^{\delta}$$

   Let λ_s be the size of the challenge space, λ_z be the statistical security parameter and μ the size of e.

   - Prover samples:

     $$r_b, r_e \leftarrow\!\!\$ \left(-2^{\lambda_z+\lambda_s+\mu}, 2^{\lambda_z+\lambda_s+\mu}\right)$$
     $$r_{\rho_B}, r_r, r_{r_a}, r_{r'_a}, r_{\rho'_B} \leftarrow\!\!\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right)$$
     $$r_\beta, r_\delta \leftarrow\!\!\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}\right)$$

     and computes:

     $$\alpha_2 = \mathsf{Acc}^{r_b} H^{r_{\rho_B}}, \quad \alpha_3 = G^{r_e} H^{r_r}, \quad \alpha_4 = G^{r_{r_a}} H^{r_{r'_a}},$$
     $$\alpha_5 = C_a^{r_e} H^{r_\beta}, \quad \alpha_6 = C_{r_a}^{r_e} G^{r_\beta} H^{r_\delta}, \quad \alpha_7 = G^{r_{\rho_B}} H^{r_{\rho'_B}}$$

     $\underline{\mathcal{P} \to \mathcal{V}} : (\alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7)$

   - Verifier samples the challenge $c \leftarrow \{0,1\}^{\lambda_s}$
     $\underline{\mathcal{V} \to \mathcal{P}} : c$
   - Prover computes the response:

     $$s_b = r_b - cb, s_e = r_e - ce$$
     $$s_{\rho_B} = r_{\rho_B} - c\rho_B, s_r = r_r - cr, s_{r_a} = r_{r_a} - cr_a, s_{r'_a} = r_{r'_a} - cr'_a, s_{\rho'_B} = r_{\rho'_B} - c\rho'_B$$
     $$s_\beta = r_\beta + c(er_a + \rho_B), \quad s_\delta = r_\delta + c(er'_a + \rho'_B)$$

     $\underline{\mathcal{P} \to \mathcal{V}} : (s_b, s_e, s_{\rho_B}, s_r, s_{r_a}, s_{r'_a}, s_{\rho'_B}, s_\beta, s_\delta)$
   - Verifier checks if:

     $$\alpha_2 \stackrel{?}{=} C_B^c \mathsf{Acc}^{s_b} H^{s_{\rho_B}}, \quad \alpha_3 \stackrel{?}{=} C_e^c G^{s_e} H^{s_r}, \quad \alpha_4 \stackrel{?}{=} C_{r_a}^c G^{s_{r_a}} H^{s_{r'_a}},$$
     $$\alpha_5 \stackrel{?}{=} C_a^{s_e} H^{s_\beta} G^c C_B^{-c}, \quad \alpha_6 \stackrel{?}{=} C_{r_a}^{s_e} H^{s_\delta} G^{s_\beta} C_{\rho_B}^{-c}, \quad \alpha_7 \stackrel{?}{=} C_{\rho_B}^c G^{s_{\rho_B}} H^{s_{\rho'_B}},$$
     $$s_e \stackrel{?}{\in} \left[-2^{\lambda_z+\lambda_s+\mu+1}, 2^{\lambda_z+\lambda_s+\mu+1}\right]$$

Coprime protocol

- On common reference string $\mathsf{crs} = (\mathbb{Z}_N^*, G, H)$
- $\mathsf{Prove}(\mathsf{crs}, (C_e, \mathsf{Acc}), (e, r, (D, b)))$:
  1. samples $r_a, r_{a'}, \rho_B, \rho_{B'} \leftarrow\!\!\$ \left(-\lfloor N/4 \rfloor, \lfloor N/4 \rfloor\right)$ and computes $C_a = DH^{r_a}, C_{r_a} = G^{r_a}H^{r'_a}, C_B = Acc^b H^{\rho_B}, C_{\rho_B} = G^{\rho_B}H^{\rho'_B}$.
  2. Computes the non-interactive version of the above protocol
     $r_b, r_e \leftarrow\!\!\$ \left(-2^{\lambda_z+\lambda_s+\mu}, 2^{\lambda_z+\lambda_s+\mu}\right)$
     $r_{\rho_B}, r_r, r_{r_a}, r_{r'_a}, r_{\rho'_B} \leftarrow\!\!\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right)$
     $r_\beta, r_\delta \leftarrow\!\!\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}\right)$
     $\alpha_2 = \mathsf{Acc}^{r_b} H^{r_{\rho_B}}, \alpha_3 = G^{r_e} H^{r_r}, \alpha_4 = G^{r_{r_a}} H^{r_{r'_a}}, \alpha_5 = C_a^{r_e} H^{r_\beta}, \alpha_6 = C_{r_a}^{r_e} G^{r_\beta} H^{r_\delta}, \alpha_7 = G^{r_{\rho_B}} H^{r_{\rho'_B}}$
     $c \leftarrow \mathsf{H}(\alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, C_e, Acc)$
     $s_b = r_b - cb, s_e = r_e - ce, s_{\rho_B} = r_{\rho_B} - c\rho_B, s_r = r_r - cr, s_{r_a} = r_{r_a} - cr_a, s_{r'_a} = r_{r'_a} - cr_{r_a}, s_{\rho'_B} = r_{\rho'_B} - c\rho'_B, s_\beta = r_\beta + c(er_a + \rho_B), s_\delta = r_\delta + c(er'_a + \rho'_B)$

  Return
  $\pi \leftarrow (C_a, C_{r_a}, C_B, C_{\rho_B}, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, s_b, s_e, s_{r_b}, s_{\rho_B}, s_r, s_{r_a}, s_{r'_a}, s_{\rho'_B}, s_\beta, s_\delta)$
- $\mathsf{VerProof}(\mathsf{crs}, (C_e, \mathsf{Acc}), \pi)$: Return 1 iff $\alpha_2 = C_B^c \mathsf{Acc}^{s_b} H^{s_{\rho_B}} \wedge \alpha_3 = C_e^c G^{s_e} H^{s_r} \wedge \alpha_4 = C_{r_a}^c G^{s_{r_a}} H^{s_{r'_a}} \wedge \alpha_5 = C_a^{s_e} H^{s_\beta} G^c C_B^{-c} \wedge \alpha_6 = C_{r_a}^{s_e} H^{s_\delta} G^{s_\beta} C_{\rho_B}^{-c} \wedge \alpha_7 = C_{\rho_B}^c G^{s_{\rho_B}} H^{s_{\rho'_B}} \wedge \wedge s_e \in \left[-2^{\lambda+\lambda_s+\mu+1}, 2^{\lambda+\lambda_s+\mu+1}\right]$

Figure 4.12: Our first Coprime protocol instantiation.

Correctness. Here we show the correctness of the protocol.

$$
\begin{aligned}
\alpha_2 &= \mathsf{Acc}^{r_b} H^{r_{\rho_B}} = \mathsf{Acc}^{s_b+cb} H^{s_{\rho_B}+c\rho_B} = \mathsf{Acc}^{s_{r_b}} H^{s_{\rho_B}} (\mathsf{Acc}^b H^{\rho_B})^c \\
&= \mathsf{Acc}^{s_b} H^{s_{\rho_B}} C_B^c \\
\alpha_3 &= G^{r_e} H^{r_r} = G^{s_e+ce} H^{s_r+cr} = G^{s_e} H^{s_r} (G^e H^r)^c \\
&= G^{s_e} H^{s_r} C_e^c \\
\alpha_4 &= G^{r_{r_a}} H^{r_{r'_a}} = G^{s_{r_a}+cr_a} H^{s_{r'_a}+cr'_a} = G^{s_{r_a}} H^{s_{r'_a}} (G^{r_a} H^{r'_a})^c \\
&= G^{s_{r_a}} H^{s_{r'_a}} C_{r_a}^c \\
\alpha_5 &= C_a^{r_e} H^{r_\beta} = C_a^{s_e+ce} H^{s_\beta - c(er_a+\rho_B)} = C_a^{s_e} H^{s_\beta} (D^e H^{er_a})^c H^{-c(er_a+\rho_B)} \\
&= C_a^{s_e} H^{s_\beta} (D^e H^{-\rho_B})^c = C_a^{s_e} H^{s_\beta} (GAcc^{-b} H^{-\rho_B})^c = \\
&= C_a^{s_e} H^{s_\beta} G^c C_B^{-c} \\
\alpha_6 &= C_{r_a}^{r_e} G^{r_\beta} H^{r_\delta} = C_{r_a}^{s_e+ce} G^{s_\beta - c(er_a+\rho_B)} H^{s_\delta - c(er'_a+\rho'_B)} \\
&= C_{r_a}^{s_e} G^{s_\beta} H^{s_\delta} (G^{r_a} H^{r'_a})^{ce} G^{-c(er_a+\rho_B)} H^{-c(er'_a+\rho'_B)} = C_{r_a}^{s_e} G^{s_\beta} H^{s_\delta} G^{-c\rho_B} H^{-c\rho'_B} \\
&= C_{r_a}^{s_e} G^{s_\beta} H^{s_\delta} C_{\rho_B}^{-c} \\
\alpha_7 &= G^{r_{\rho_B}} H^{r_{\rho'_B}} = G^{s_{\rho_B}+c\rho_B} H^{s_{\rho'_B}+c\rho'_B} = G^{s_{\rho_B}} H^{s_{\rho'_B}} (G^{\rho_B} H^{\rho'_B})^c \\
&= G^{s_{\rho_B}} H^{s_{\rho'_B}} C_{\rho_B}^c
\end{aligned}
$$

Security. Security of our scheme holds with the partial opening of Acc, i.e., when it is ensured outside the protocol that Acc is a valid commitment of the set. The proof is similar to the one of theorem 6. The main technical difference is in the extraction of the opening of C_B, because Acc is not a random generator sampled at the setup phase. However, from partial opening we know that it is $\mathsf{Acc} = G^{\mathsf{prod}}$ for a random generator G. This will allow us to state an alternative to lemma 6 to justify the extraction of the opening of C_B.

Theorem 11. *Let $\mathbb{Z}_N^*$ be an RSA group where strong-RSA assumption holds, then the above protocol is honest-verifier zero knowledge protocol and, also, if $\lambda_s + 1 < \mu$ and $\lambda_s < \log(N)/2$, is knowledge sound with partial opening of* Acc *for R_{Coprime}.*

Proof. Zero-Knowledge can be proven with standard techniques, similar to the ones in the proof of theorem 6 and is therefore omitted.

For the knowledge soundness, let an adversary of the knowledge soundness $\mathcal{A}$ that is able to convince the verifier $\mathcal{V}$ with a probability at least ϵ. We will construct an extractor $\mathcal{E}$ that extracts the witness $(e, r, r_2, r_3, \beta, \delta)$. Using rewinding $\mathcal{E}$ gets two accepted transcripts

$$(C_a, C_{r_a}, C_B, C_{\rho_B}, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, c, s_b, s_e, s_{\rho_B}, s_r, s_{r_a}, s_{r'_a}, s_{\rho'_B}, s_\beta, s_\delta)$$

$$(C_a, C_{r_a}, C_B, C_{\rho_B}, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, c', s'_b, s'_e, s'_{\rho_B}, s'_r, s'_{r_a}, s'_{r'_a}, s'_{\rho'_B}, s'_\beta, s'_\delta)$$

on two different challenges c and c'. $\mathcal{E}$ aborts if it cannot get two such transcripts (abort1).

We denote $\Delta c \coloneqq c' - c, \Delta s_b \coloneqq s_b - s'_b, \Delta s_e \coloneqq s_e - s'_e, \Delta s_{\rho_B} \coloneqq s_{\rho_B} - s'_{\rho_B}, \Delta s_r \coloneqq s_r - s'_r, \Delta s_{r_a} \coloneqq s_{r_a} - s'_{r_a}, \Delta s_{r'_a} \coloneqq s_{r'_a} - s'_{r'_a}, \Delta s_{\rho'_B} \coloneqq s_{\rho'_B} - s'_{\rho'_B}, \Delta s_\beta \coloneqq s_\beta - s'_\beta, \Delta s_\delta \coloneqq s_\delta - s'_\delta$ then

$$C_B^{\Delta c} = \mathsf{Acc}^{\Delta s_b} H^{\Delta s_{\rho_B}} \Rightarrow C_B = \pm \mathsf{Acc}^{\hat{b}} H^{\hat{\rho_B}} \tag{4.1}$$

$$C_e^{\Delta c} = G^{\Delta s_e} H^{\Delta s_r} \Rightarrow C_e = \pm G^{\hat{e}} H^{\hat{r}} \tag{4.2}$$

$$C_{r_a}^{\Delta c} = G^{\Delta s_{r_a}} H^{\Delta s_{r'_a}} \Rightarrow C_{r_a} = \pm G^{\hat{r_a}} H^{\hat{r'_a}} \tag{4.3}$$

$$1 = C_a^{\Delta s_e} H^{\Delta s_\beta} G^{-\Delta c} C_B^{\Delta c} \tag{4.4}$$

$$1 = C_{r_a}^{\Delta s_e} H^{\Delta s_\delta} G^{\Delta s_\beta} C_{\rho_B}^{\Delta c} \tag{4.5}$$

$$C_{\rho_B}^{\Delta c} = G^{\Delta s_{\rho_B}} H^{\Delta s_{\rho'_B}} \Rightarrow C_{\rho_B} = \pm G^{\hat{\rho_B}} H^{\hat{\rho'_B}} \tag{4.6}$$

define the (possibly rational) numbers $\hat{b} \coloneqq \frac{\Delta s_b}{\Delta c}$, $\hat{e} \coloneqq \frac{\Delta s_e}{\Delta c}$, $\hat{r} \coloneqq \frac{\Delta s_r}{\Delta c}$, $\hat{r_a} \coloneqq \frac{\Delta s_{r_a}}{\Delta c}$, $\hat{r'_a} \coloneqq \frac{\Delta s_{r'_a}}{\Delta c}$, $\hat{\rho_B} \coloneqq \frac{\Delta s_{\rho_B}}{\Delta c}$, $\hat{\rho'_B} \coloneqq \frac{\Delta s_{\rho'_B}}{\Delta c}$.

$\mathcal{E}$ aborts in case Δc doesn't divide: Δs_e and Δs_r(abort 2a), Δs_{r_a} and $\Delta s_{r'_a}$(abort 2b), Δs_{ρ_B} and $\Delta s_{\rho'_B}$(abort 2c). And finally, $\mathcal{E}$ aborts if Δc doesn't divide Δs_b and Δs_{ρ_B} (abort 2d). Therefore, after these aborts didn't happen we can infer the equivalent equalities on the right of equations 4.2,4.3,4.6 and 6.2.

If we replace equations 4.3 and 4.6 in equation 4.5 we get
$1 = \left(\pm G^{\hat{r_a}} H^{\hat{r'_a}}\right)^{\Delta s_e} H^{\Delta s_\beta} G^{\Delta s_\beta} \left(\pm G^{\hat{\rho_B}} H^{\hat{\rho'_B}}\right)^{\Delta c}$ or
$1 = (\pm 1)^{\Delta s_e} (\pm 1)^{\Delta c} G^{\hat{r_a}\Delta s_e + \hat{\rho_B}\Delta c + \Delta s_\beta} H^{\hat{r'_a}\Delta s_e + \hat{\rho'_B}\Delta c + \Delta s_\beta}$. Since $G, H, 1$ are quadratic residues

then $(\pm 1)^{\Delta s_e}(\pm 1)^{\Delta c} = 1$, hence $1 = G^{\hat{r_a}\Delta s_e + \hat{\rho_B}\Delta c + \Delta s_\beta} H^{\hat{r'_a}\Delta s_e + \hat{\rho'_B}\Delta c + \Delta s_\beta}$. Then under the DLOG assumption $\hat{r_a}\Delta s_e + \hat{\rho_B}\Delta c + \Delta s_\beta = 0 = \hat{r'_a}\Delta s_e + \hat{\rho'_B}\Delta c + \Delta s_\beta$, which gives us that

$$\Delta s_\beta = -\hat{r_a}\Delta s_e - \hat{\rho_B}\Delta c \tag{4.7}$$

Finally, we replace equations 6.2 and 4.7 in equation 4.4 we get $1 = C_a^{\Delta s_e} H^{-\hat{r_a}\Delta s_e - \hat{\rho_B}\Delta c} G^{-\Delta c} \left(\pm \mathsf{Acc}^{\hat{b}} H^{\hat{\rho_B}}\right)^{\Delta c}$ or $1 = (\pm 1)^{\Delta c} C_a^{\Delta s_e} \mathsf{Acc}^{\hat{b}\Delta c} G^{-\Delta c} H^{-\hat{r_a}\Delta s_e}$ or $\left(\pm \mathsf{Acc}^{\hat{b}} G^{-1}\right)^{\Delta c} = \left(C_a^{-1} H^{r_a}\right)^{\Delta s_e}$. But as noted above Δc divides Δs_e so $\pm \mathsf{Acc}^{\hat{b}} G^{-1} = \pm \left(C_a^{-1} H^{r_a}\right)^{\hat{e}} \Rightarrow \mathsf{Acc}^{\hat{b}} G^{-1} = \pm \left(C_a^{-1} H^{\hat{r_a}}\right)^{\hat{e}} \Rightarrow \left(\frac{C_a}{H^{\hat{r_a}}}\right)^{\hat{e}} \mathsf{Acc}^{\hat{b}} = \pm G$. We discern two cases:

- $\underline{\left(\frac{C_a}{H^{\hat{r_a}}}\right)^{\hat{e}} \mathsf{Acc}^{\hat{b}} = +G}$: Then $\mathcal{E}$ sets $\tilde{D} \leftarrow \frac{C_a}{H^{\hat{r_a}}}$, $\tilde{e} \leftarrow \hat{e} := \frac{\Delta s_e}{\Delta c}$, $\tilde{r} \leftarrow \hat{r} := \frac{\Delta s_r}{\Delta c}$ and $\tilde{b} \leftarrow \hat{b} := \frac{\Delta s_b}{\Delta c}$
- $\underline{\left(\frac{C_a}{H^{\hat{r_a}}}\right)^{\hat{e}} \mathsf{Acc}^{\hat{b}} = -G}$: Then $\hat{e}$ should be odd otherwise if $\hat{e} = 2\rho$ then $G = -\left(\frac{C_a}{H^{\hat{r_a}}}\right)^{2\rho} \mathsf{Acc}^{\hat{b}}$ would be a non-quadratic residue. So $\mathcal{E}$ sets $\tilde{D} \leftarrow -\frac{C_a}{H^{\hat{r_a}}}$, $\tilde{e} \leftarrow \hat{e} := \frac{\Delta s_e}{\Delta c}$, $\tilde{r} \leftarrow \hat{r} := \frac{\Delta s_r}{\Delta c}$ and $\tilde{b} \leftarrow \hat{b} := \frac{\Delta s_b}{\Delta c}$. It is clear that $\tilde{D}^{\tilde{e}} \mathsf{Acc}^{\tilde{b}} = G$.

Finally the $\mathcal{E}$ outputs $(\tilde{e}, \tilde{r}, \tilde{D}, \tilde{b})$.

Now we show that the probability the extractor terminates with outputting a valid witness is $O(\epsilon)$. If the extractor does not abort then it clearly outputs a valid witness (under the factoring assumption). For the first abort, with a standard argument it can be shown that the extractor is able to extract two accepting transcripts with probability $O(\epsilon)$ (for the probabilistic analysis we refer to [85]). Thus $Pr[\mathsf{abort1}] = 1 - \mathsf{O}(\epsilon)$. For the aborts abort 2a, abort 2b and abort 2c they happen with negligible probability ($\leq \frac{2}{1-2^{-\lambda_s+1}} Pr[\mathcal{B} \text{ solves } sRSA]$ each, for any PPT adversary $\mathcal{B}$) under the strong RSA assumption according to lemma 6. For abort 2d we cannot directly use the same lemma as Acc is not a random generator that is part of the crs. However, with a similar argument and using partial extractability we show below that the probability for this abort is the same. Putting them together the probability of success of $\mathcal{E}$ is at least $O(\epsilon) - \frac{8}{1-2^{-\lambda_s+1}} Pr[\mathcal{B} \text{ solves } sRSA] = O(\epsilon) - \mathsf{negl}(\lambda_s)$.

For equation 6.2, we get from partial opening that $\mathsf{Acc} = G^{\mathsf{prod}_P}$, where $P := \{\mathsf{Hprime}(u) \mid u \in S\}$, so

$$C_B^{\Delta c} = G^{\prod_{u \in S} \mathsf{Hprime}(u) \cdot \Delta s_b} H^{\Delta s_{\rho_B}}$$

We use a similar to [85] argument to prove that Δc divides Δs_b and Δs_{ρ_B} under the strong RSA assumption, given that $\lambda_s + 1 < \mu$. Then

$$C_B = \pm \mathsf{Acc}^{\hat{b}} H^{\hat{\rho_B}} \tag{4.8}$$

Lemma 7. *Let $\lambda_s + 1 < \mu$ and $\lambda_s < \log(N)/2$ then Δc divides Δs_b and Δs_{ρ_B} under the strong RSA assumption.*

Proof. An adversary against the strong RSA assumption receives $H \in \mathsf{QR}_N$ and does the following: sets $G = H^\tau$ for $\tau \leftarrow_\$ [0, 2^{\lambda_s} N^2]$ and sends (G, H) to the adversary $\mathcal{A}$ which outputs a proof π_{Coprime}. Then we rewind to get another successful proof π'_{Coprime} and we use the extractor as above to get $C_B^{\Delta c} = G^{\prod_{u \in S} \mathsf{Hprime}(u) \cdot \Delta s_b} H^{\Delta s_{\rho_B}}$ or

$$C_B^{\Delta c} = H^{\tau \prod_{u \in S} \mathsf{Hprime}(u) \cdot \Delta s_b + \Delta s_{\rho_B}}$$

We can exclude the case that Δc divides $\prod_{u\in S}\mathsf{Hprime}(u)$, since Δc is smaller than the domain of the hash function Hprime, i.e. $\Delta c < \mathsf{Hprime}(u)$ for each $u \in S$, which comes from $\lambda_s + 1 < \mu$. Assume that $\Delta c \nmid \Delta s_b \vee \Delta c \nmid \Delta s_{\rho_B}$. we discern two cases:

- Δc doesn't divide $\tau\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B}$: then $\gcd(\Delta c, \tau\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B}) = g$ and there are χ,ψ such that $\chi\cdot\Delta c+\psi\cdot\left(\tau\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B}\right) = g$. Thus

$$H^g = H^{\chi\cdot\Delta c+\psi\cdot\left(\tau\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B}\right)} = H^{\chi\Delta c}\cdot C_B^{\psi\Delta c} = \left(H^\chi\cdot C_B^\psi\right)^{\Delta c}$$

  Since g divides Δc we get $H = \pm\left(H^\chi\cdot C_B^\psi\right)^{\frac{\Delta c}{g}}$. However H is a quadratic residue (thus C_B is so), meaning that $H = \left(H^\chi\cdot C_B^\psi\right)^{\frac{\Delta c}{g}}$, thus $(H^\chi\cdot C_B^\psi, \frac{\Delta c}{g})$ is a solution to the strong RSA problem.

- Δc divides $\tau\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B}$: let q^ℓ be the maximal q-power that divides Δc (i.e. q^ℓ is a factor of Δ) and doesn't divide at least one of Δs_b and Δs_{ρ_B}, where q is prime. Such a q^ℓ should exist otherwise Δc would divide both Δs_b and Δs_{ρ_B}, which we assumed it doesn't. Notice that if q^ℓ divided Δs_b then it would also divide Δs_{ρ_B}, as q^ℓ divides $\tau\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B}$ (from assumption), so $q^\ell \nmid \Delta s_b$.

$$q^\ell \mid \left(\tau\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B}\right) \Rightarrow \tau\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B} = 0 \pmod{q^\ell}$$

We can write $\tau := \tau_1 + \tau_2\,\mathsf{ord}(H)$. Notice that τ_2 is information theoretically hidden to the adversary and thus is uniformly random in $[0, 2^{\lambda_s}N^2/\mathsf{ord}(H)] \supset [0, 2^{\lambda_s}N]$ in its view.

$$\Rightarrow \tau_1\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b + \tau_2\mathsf{ord}(H)\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b+\Delta s_{\rho_B} = 0 \pmod{q^\ell}$$

$$\Rightarrow \tau_2\cdot\Delta s_b = \left(-\tau_1\prod_{u\in S}\mathsf{Hprime}(u)\cdot\Delta s_b - \Delta s_{\rho_B}\right)\cdot\left(\prod_{u\in S}\mathsf{Hprime}(u)\right)^{-1}\cdot(\mathsf{ord}(H))^{-1} \pmod{q^\ell}$$

To see that $\prod_{u\in S}\mathsf{Hprime}(u)$ has an inverse modulo q^ℓ note that since $\Delta c < \mathsf{Hprime}(u)$ implies $q^\ell < \mathsf{Hprime}(u)$, so $\gcd(\prod_{u\in S}\mathsf{Hprime}(u), q^\ell) = 1$. For the inverse of $\mathsf{ord}(H)$ note that $H \in \mathsf{QR}_N$ so $\mathsf{ord}(H) \in \{q_1, q_2, q_1q_2\}$, where $N = (2q_1+1)(2q_2+1)$ is the RSA modulus. Then from $\lambda_s < \log(N)/2$ we get $\Delta c < q_1, q_2$ and thus $\gcd(\mathsf{ord}(H), q^\ell) = 1$.

As noted above, τ_2 is uniformly random in a superset of $[0, 2^{\lambda_s}N]$. But $q^\ell < \Delta c < N$, so $2^{\lambda_s}N$ is at least 2^{λ_s} larger than q^ℓ. Thus τ_2 is statistically close to uniform in $\{0, 1, \ldots, q^\ell - 1\}$ (with $2^{-\lambda_s}$ error), $Pr_{\tau_2}[\tau_2 = C \pmod{q^\ell}] \approx \frac{1}{q^\ell}$. Furthermore, for any Δs_b, $Pr_{\tau_2}[\tau_2\cdot\Delta s_b = C \pmod{q^\ell}] \approx \frac{1}{q^\ell}\cdot\gcd(q^\ell, \Delta s_b) \leq \frac{1}{q^\ell}\cdot q^{\ell-1}$ (since q^ℓ doesn't divide Δs_b). This is because for variable τ_2, the equation $\tau_2\Delta s_b = C \pmod{q^\ell}$ has $\gcd(q^\ell, \Delta s_b)$ solutions.

In conclusion, the probability that the above equation holds is at most $\frac{1}{q} + 2^{-\lambda_s} \leq \frac{1}{2} + 2^{-\lambda_s}$.

To summarize we showed that the probability to fall in the second case is at most $\frac{1}{2} + 2^{-\lambda_s}$. So with probability to fall in the first case, and thus solve the strong RSA problem, is at least $\frac{1}{2} - 2^{-\lambda_s}$. □

By a simple argument identical to the one of section 4.4.4, we can also conclude about the range of the extracted $\tilde{e}$: $s_e \overset{?}{\in} \left[-2^{\lambda_z+\lambda_s+\mu+1}, 2^{\lambda_z+\lambda_s+\mu+1}\right]$ implies $-2^{\lambda_z+\lambda_s+\mu+2} \leq \hat{e} \leq 2^{\lambda_z+\lambda_s+\mu+2}$. □

4.5.2 Alternative Instantiation of Protocol for R_{Coprime}

Below we propose another interactive ZK protocol for R_{Coprime}. The difference with the above is that it doesn't have the limitation of $\lambda_s + 1 < \mu$ and $\lambda_s < \log(N)/2$. Also, partial opening of Acc isn't needed. This comes with a cost of 2 more group elements in the proof size, 4 more exponentiations for the prover and 2 more for the verifier.

1. Prover computes $C_a = DH^{r_a}, C_{r_a} = G^{r_a}H^{r'_a}, C_b = G^bH^{\rho_b}, C_B = Acc^bH^{\rho_B}, C_{\rho_B} = G^{\rho_B}H^{\rho'_B}$ and sends to the verifier:

   $\underline{\mathcal{P} \to \mathcal{V}} : C_a, C_b, C_{r_a}, C_B, C_{\rho_B}$
2. Prover and Verifier perform a protocol for the relation:

   $R((\mathsf{Acc}, C_e, C_a, C_{r_a}, C_b, C_B, C_{\rho_B}), (e, r, r_a, r'_a, b, \rho_b, \rho_B, \rho'_B, D, B, \beta, \delta)) = 1$ iff

   $$C_b = G^bH^{\rho_b} \wedge C_B = Acc^bH^{\rho_B} \wedge C_e = G^eH^r \wedge C_{r_a} = G^{r_a}H^{r'_a}$$
   $$\wedge C_{\rho_B} = G^{\rho_B}H^{\rho'_B} \wedge C_a^eC_B = GH^\beta \wedge C_{r_a}^eC_{\rho_B} = G^\beta H^\delta$$

   Let λ_s be the size of the challenge space, λ_z be the statistical security parameter and μ the size of e.

   - Prover samples:

     $$r_b, r_e \leftarrow\!\!\$ \left(-2^{\lambda_z+\lambda_s+\mu}, 2^{\lambda_z+\lambda_s+\mu}\right)$$
     $$r_{\rho_b}, r_{\rho_B}, r_r, r_{r_a}, r_{r'_a}, r_{\rho'_B} \leftarrow\!\!\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right)$$
     $$r_\beta, r_\delta \leftarrow\!\!\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}\right)$$

     and computes:

     $$\alpha_1 = G^{r_b}H^{r_{\rho_b}}, \quad \alpha_2 = \mathsf{Acc}^{r_b}H^{r_{\rho_B}}, \quad \alpha_3 = G^{r_e}H^{r_r}, \quad \alpha_4 = G^{r_{r_a}}H^{r_{r'_a}},$$
     $$\alpha_5 = C_a^{r_e}H^{r_\beta}, \quad \alpha_6 = C_{r_a}^{r_e}G^{r_\beta}H^{r_\delta}, \quad \alpha_7 = G^{r_{\rho_B}}H^{r_{\rho'_B}}$$

     $\underline{\mathcal{P} \to \mathcal{V}} : (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7)$
   - Verifier samples the challenge $c \leftarrow \{0,1\}^{\lambda_s}$

     $\underline{\mathcal{V} \to \mathcal{P}} : c$

- Prover computes the response:

$$s_b = r_b - cb, \quad s_e = r_e - ce$$
$$s_{\rho_b} = r_{\rho_b} - c\rho_b, \quad s_{\rho_B} = r_{\rho_B} - c\rho_B, \quad s_r = r_r - cr, s_{r_a} = r_{r_a} - cr_a, \quad s_{r'_a} = r_{r'_a} - cr'_a,$$
$$s_{\rho'_B} = r_{\rho'_B} - c\rho'_B, \quad s_\beta = r_\beta + c(er_a + \rho_B), \quad s_\delta = r_\delta + c(er'_a + \rho'_B)$$

$\underline{\mathcal{P} \to \mathcal{V}} : (s_b, s_e, s_{\rho_b}, s_{\rho_B}, s_r, s_{r_a}, s_{r'_a}, s_{\rho'_B}, s_\beta, s_\delta)$

- Verifier checks if:

$$\alpha_1 \stackrel{?}{=} C_b^c G^{s_b} H^{s_{\rho_b}}, \quad \alpha_2 \stackrel{?}{=} C_B^c \mathsf{Acc}^{s_b} H^{s_{\rho_B}}, \quad \alpha_3 \stackrel{?}{=} C_e^c G^{s_e} H^{s_r}, \quad \alpha_4 \stackrel{?}{=} C_{r_a}^c G^{s_{r_a}} H^{s_{r'_a}},$$
$$\alpha_5 \stackrel{?}{=} C_a^{s_e} H^{s_\beta} G^c C_B^{-c}, \quad \alpha_6 \stackrel{?}{=} C_{r_a}^{s_e} H^{s_\delta} G^{s_\beta} C_{\rho_B}^{-c}, \quad \alpha_7 \stackrel{?}{=} C_{\rho_B}^c G^{s_{\rho_B}} H^{s_{\rho'_B}},$$
$$s_e \stackrel{?}{\in} \left[-2^{\lambda_z+\lambda_s+\mu+1}, 2^{\lambda_z+\lambda_s+\mu+1}\right]$$

Coprime2 protocol

- On common reference string $\mathsf{crs} = (\mathbb{Z}_N^*, G, H)$
- $\mathsf{Prove}(\mathsf{crs}, (C_e, \mathsf{Acc}), (e, r, (D, b)))$:
  1. samples $r_a, r_{a'}, \rho_b, \rho_B, \rho_{B'} \leftarrow_\$ (-\lfloor N/4 \rfloor, \lfloor N/4 \rfloor)$ and computes $C_a = DH^{r_a}, C_{r_a} = G^{r_a}H^{r'_a}, C_b = G^b H^{\rho_b}, C_B = Acc^b H^{\rho_B}, C_{\rho_B} = G^{\rho_B} H^{\rho'_B}$.
  2. Computes the non-interactive version of the above protocol
     $r_b, r_e \leftarrow_\$ \left(-2^{\lambda_z+\lambda_s+\mu}, 2^{\lambda_z+\lambda_s+\mu}\right)$
     $r_{\rho_b}, r_{\rho_B}, r_r, r_{r_a}, r_{r'_a}, r_{\rho'_B} \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right)$
     $r_\beta, r_\delta \leftarrow_\$ \left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s+\mu}\right)$
     $\alpha_1 = G^{r_b} H^{r_{\rho_b}}, \alpha_2 = \mathsf{Acc}^{r_b} H^{r_{\rho_B}}, \alpha_3 = G^{r_e} H^{r_r}, \alpha_4 = G^{r_{r_a}} H^{r_{r'_a}}, \alpha_5 = C_a^{r_e} H^{r_\beta}, \alpha_6 = C_{r_a}^{r_e} G^{r_\beta} H^{r_\delta}, \alpha_7 = G^{r_{\rho_B}} H^{r_{\rho'_B}}$
     $c \leftarrow \mathsf{H}(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, C_e, Acc)$
     $s_b = r_b - cb, s_e = r_e - ce, s_{\rho_b} = r_{\rho_b} - c\rho_b, s_{\rho_B} = r_{\rho_B} - c\rho_B, s_r = r_r - cr, s_{r_a} = r_{r_a} - cr_a, s_{r'_a} = r_{r'_a} - cr_{r_a}, s_{\rho'_B} = r_{\rho'_B} - c\rho'_B, s_\beta = r_\beta + c(er_a + \rho_B), s_\delta = r_\delta + c(er'_a + \rho'_B)$

  Returns $\pi \leftarrow (C_a, C_{r_a}, C_b, C_B, C_{\rho_B}, \alpha_1, \alpha_2, \alpha_3, \alpha_4, cs_b, s_e, s_{\rho_b}, s_{\rho_B}, s_r, s_{r_a}, s_{r'_a}, s_{\rho'_B}, s_\beta, s_\delta)$
- $\mathsf{VerProof}(\mathsf{crs}, (C_e, Acc), \pi)$: returns 1 iff $\alpha_1 = C_b^c G^{s_b} H^{s_{\rho_b}} \wedge \alpha_2 = C_B^c \mathsf{Acc}^{s_b} H^{s_{\rho_B}} \wedge \alpha_3 = C_e^c G^{s_e} H^{s_r} \wedge \alpha_4 = C_{r_a}^c G^{s_{r_a}} H^{s_{r'_a}} \wedge \alpha_5 = C_a^{s_e} H^{s_\beta} G^c C_B^{-c} \wedge \alpha_6 = C_{r_a}^{s_e} H^{s_\delta} G^{s_\beta} C_{\rho_B}^{-c} \wedge \alpha_7 = C_{\rho_B}^c G^{s_{\rho_B}} H^{s_{\rho'_B}} \wedge \wedge s_e \in \left[-2^{\lambda+\lambda_s+\mu+1}, 2^{\lambda+\lambda_s+\mu+1}\right]$

Figure 4.13: Our second Coprime2 protocol instantiation.

Correctness. Here we show the correctness of the protocol.

$$\begin{aligned}
\alpha_1 &= G^{r_b}H^{r_{\rho_b}} = G^{s_b+cb}H^{s_{\rho_b}+cr_b} = G^{s_b}H^{s_{\rho_b}}(G^bH^{\rho_b})^c \\
&= G^{s_b}H^{s_{\rho_b}}C_b^c \\
\alpha_2 &= \mathsf{Acc}^{r_b}H^{r_{\rho_B}} = \mathsf{Acc}^{s_b+cb}H^{s_{\rho_B}+c\rho_B} = \mathsf{Acc}^{s_{r_b}}H^{s_{\rho_B}}(\mathsf{Acc}^bH^{\rho_B})^c \\
&= \mathsf{Acc}^{s_b}H^{s_{\rho_B}}C_B^c \\
\alpha_3 &= G^{r_e}H^{r_r} = G^{s_e+ce}H^{s_r+cr} = G^{s_e}H^{s_r}(G^eH^r)^c \\
&= G^{s_e}H^{s_r}C_e^c \\
\alpha_4 &= G^{r_{r_a}}H^{r_{r'_a}} = G^{s_{r_a}+cr_a}H^{s_{r'_a}+cr'_a} = G^{s_{r_a}}H^{s_{r'_a}}(G^{r_a}H^{r'_a})^c \\
&= G^{s_{r_a}}H^{s_{r'_a}}C_{r_a}^c \\
\alpha_5 &= C_a^{r_e}H^{r_\beta} = C_a^{s_e+ce}H^{s_\beta-c(er_a+\rho_B)} = C_a^{s_e}H^{s_\beta}(D^eH^{er_a})^cH^{-c(er_a+\rho_B)} \\
&= C_a^{s_e}H^{s_\beta}(D^eH^{-\rho_B})^c = C_a^{s_e}H^{s_\beta}(GAcc^{-b}H^{-\rho_B})^c = \\
&= C_a^{s_e}H^{s_\beta}G^cC_B^{-c} \\
\alpha_6 &= C_{r_a}^{r_e}G^{r_\beta}H^{r_\delta} = C_{r_a}^{s_e+ce}G^{s_\beta-c(er_a+\rho_B)}H^{s_\delta-c(er'_a+\rho'_B)} \\
&= C_{r_a}^{s_e}G^{s_\beta}H^{s_\delta}(G^{r_a}H^{r'_a})^{ce}G^{-c(er_a+\rho_B)}H^{-c(er'_a+\rho'_B)} = C_{r_a}^{s_e}G^{s_\beta}H^{s_\delta}G^{-c\rho_B}H^{-c\rho'_B} \\
&= C_{r_a}^{s_e}G^{s_\beta}H^{s_\delta}C_{\rho_B}^{-c} \\
\alpha_7 &= G^{r_{\rho_B}}H^{r_{\rho'_B}} = G^{s_{\rho_B}+c\rho_B}H^{s_{\rho'_B}+c\rho'_B} = G^{s_{\rho_B}}H^{s_{\rho'_B}}(G^{\rho_B}H^{\rho'_B})^c \\
&= G^{s_{\rho_B}}H^{s_{\rho'_B}}C_{\rho_B}^c
\end{aligned}$$

Security.

Theorem 12. *Let $\mathbb{Z}_N^*$ be an RSA group where strong-RSA assumption holds, then the above protocol is an honest-verifier zero knowledge and knowledge sound protocol for R_{Coprime}.*

Proof. Zero-Knowledge can be proven with standard techniques, similar to the ones in the proof of theorem 6 and is therefore omitted.

For the knowledge soundness, let an adversary of the knowledge soundness $\mathcal{A}$ that is able to convince the verifier $\mathcal{V}$ with a probability at least ϵ. We will construct an extractor $\mathcal{E}$ that extracts the witness $(e, r, r_2, r_3, \beta, \delta)$. Using rewinding $\mathcal{E}$ gets two accepted transcripts

$$(C_a, C_b, C_{r_a}, C_B, C_{\rho_B}, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, c, s_b, s_e, s_{\rho_b}, s_{\rho_B}, s_r, s_{r_a}, s_{r'_a}, s_{\rho'_B}, s_\beta, s_\delta)$$

$$(C_a, C_b, C_{r_a}, C_B, C_{\rho_B}, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, c', s'_b, s'_e, s'_{\rho_b}, s'_{\rho_B}, s'_r, s'_{r_a}, s'_{r'_a}, s'_{\rho'_B}, s'_\beta, s'_\delta)$$

on two different challenges c and c'. $\mathcal{E}$ aborts if it cannot get two such transcripts (abort1).

We denote $\Delta c := c' - c, \Delta s_b := s_b - s'_b, \Delta s_e := s_e - s'_e, \Delta s_{\rho_b} := s_{\rho_b} - s'_{\rho_b}, \Delta s_{\rho_B} := s_{\rho_B} - s'_{\rho_B}, \Delta s_r := s_r - s'_r, \Delta s_{r_a} := s_{r_a} - s'_{r_a}, \Delta s_{r'_a} := s_{r'_a} - s'_{r'_a}, \Delta s_{\rho'_B} := s_{\rho'_B} - s'_{\rho'_B}, \Delta s_\beta := s_\beta - s'_\beta, \Delta s_\delta := s_\delta - s'_\delta$ then

$$C_b^{\Delta c} = G^{\Delta s_b}H^{\Delta s_{\rho_b}} \Rightarrow C_b = \pm G^{\hat{b}}H^{\hat{\rho_b}} \tag{4.9}$$

$$C_B^{\Delta c} = \mathsf{Acc}^{\Delta s_b}H^{\Delta s_{\rho_B}} \Rightarrow C_B = \pm \mathsf{Acc}^{\hat{b}}H^{\hat{\rho_B}} \tag{4.10}$$

$$C_e^{\Delta c} = G^{\Delta s_e} H^{\Delta s_r} \Rightarrow C_e = \pm G^{\hat{e}} H^{\hat{r}} \tag{4.11}$$

$$C_{r_a}^{\Delta c} = G^{\Delta s_{r_a}} H^{\Delta s_{r'_a}} \Rightarrow C_{r_a} = \pm G^{\hat{r_a}} H^{\hat{r'_a}} \tag{4.12}$$

$$1 = C_a^{\Delta s_e} H^{\Delta s_\beta} G^{-\Delta c} C_B^{\Delta c} \tag{4.13}$$

$$1 = C_{r_a}^{\Delta s_e} H^{\Delta s_\delta} G^{\Delta s_\beta} C_{\rho_B}^{\Delta c} \tag{4.14}$$

$$C_{\rho_B}^{\Delta c} = G^{\Delta s_{\rho_B}} H^{\Delta s_{\rho'_B}} \Rightarrow C_{\rho_B} = \pm G^{\hat{\rho_B}} H^{\hat{\rho'_B}} \tag{4.15}$$

define the (possibly rational) numbers $\hat{b} := \frac{\Delta s_b}{\Delta c}$, $\hat{\rho_b} := \frac{\Delta s_{\rho_b}}{\Delta c}$, $\hat{e} := \frac{\Delta s_e}{\Delta c}$, $\hat{r} := \frac{\Delta s_r}{\Delta c}$, $\hat{r_a} := \frac{\Delta s_{r_a}}{\Delta c}$, $\hat{r'_a} := \frac{\Delta s_{r'_a}}{\Delta c}$, $\hat{\rho_B} := \frac{\Delta s_{\rho_B}}{\Delta c}$, $\hat{\rho'_B} := \frac{\Delta s_{\rho'_B}}{\Delta c}$.

$\mathcal{E}$ aborts in case Δc doesn't divide: Δs_b and Δs_{ρ_b}(abort 2a), Δs_e and Δs_r(abort 2b), Δs_{r_a} and $\Delta s_{r'_a}$(abort 2c), Δs_{ρ_B} and $\Delta s_{\rho'_B}$(abort 2d). And finally, $\mathcal{E}$ aborts if Δc doesn't divide Δs_{ρ_B} (abort 2e). Therefore, after these aborts didn't happen we can infer the equivalent equalities on the right of equations 4.9,4.11,4.12,4.15 and 4.10.

If we replace equations 4.12 and 4.15 in equation 4.14 we get $1 = \left(\pm G^{\hat{r_a}} H^{\hat{r'_a}}\right)^{\Delta s_e} H^{\Delta s_\beta} \cdot G^{\Delta s_\beta} \left(\pm G^{\hat{\rho_B}} H^{\hat{\rho'_B}}\right)^{\Delta c}$ or $1 = (\pm 1)^{\Delta s_e} (\pm 1)^{\Delta c} G^{\hat{r_a}\Delta s_e + \hat{\rho_B}\Delta c + \Delta s_\beta} H^{\hat{r'_a}\Delta s_e + \hat{\rho'_B}\Delta c + \Delta s_\beta}$. Since $G, H, 1$ are quadratic residues then $(\pm 1)^{\Delta s_e}(\pm 1)^{\Delta c} = 1$, hence $1 = G^{\hat{r_a}\Delta s_e + \hat{\rho_B}\Delta c + \Delta s_\beta} \cdot H^{\hat{r'_a}\Delta s_e + \hat{\rho'_B}\Delta c + \Delta s_\beta}$. Then under the DLOG assumption $\hat{r_a}\Delta s_e + \hat{\rho_B}\Delta c + \Delta s_\beta = 0 = \hat{r'_a}\Delta s_e + \hat{\rho'_B}\Delta c + \Delta s_\beta$, which gives us that

$$\Delta s_\beta = -\hat{r_a}\Delta s_e - \hat{\rho_B}\Delta c \tag{4.16}$$

Finally, we replace equations 4.10 and 4.16 in equation 4.13 we get $1 = C_a^{\Delta s_e} H^{-\hat{r_a}\Delta s_e - \hat{\rho_B}\Delta c} G^{-\Delta c} \left(\pm \mathsf{Acc}^{\hat{b}} H^{\hat{\rho_B}}\right)^{\Delta c}$ or $1 = (\pm 1)^{\Delta c} C_a^{\Delta s_e} \mathsf{Acc}^{\hat{b}\Delta c} G^{-\Delta c} H^{-\hat{r_a}\Delta s_e}$ or $\left(\pm \mathsf{Acc}^{\hat{b}} G^{-1}\right)^{\Delta c} = \left(C_a^{-1} H^{r_a}\right)^{\Delta s_e}$. But as noted above Δc divides Δs_e so $\pm \mathsf{Acc}^{\hat{b}} G^{-1} = \pm \left(C_a^{-1} H^{r_a}\right)^{\hat{e}} \Rightarrow \mathsf{Acc}^{\hat{b}} G^{-1} = \pm \left(C_a^{-1} H^{\hat{r_a}}\right)^{\hat{e}} \Rightarrow \left(\frac{C_a}{H^{\hat{r_a}}}\right)^{\hat{e}} \mathsf{Acc}^{\hat{b}} = \pm G$. We discern two cases:

- $\left(\frac{C_a}{H^{\hat{r_a}}}\right)^{\hat{e}} \mathsf{Acc}^{\hat{b}} = +G$: Then $\mathcal{E}$ sets $\tilde{D} \leftarrow \frac{C_a}{H^{\hat{r_a}}}$, $\tilde{e} \leftarrow \hat{e} := \frac{\Delta s_e}{\Delta c}$, $\tilde{r} \leftarrow \hat{r} := \frac{\Delta s_r}{\Delta c}$ and $\tilde{b} \leftarrow \hat{b} := \frac{\Delta s_b}{\Delta c}$
- $\left(\frac{C_a}{H^{\hat{r_a}}}\right)^{\hat{e}} \mathsf{Acc}^{\hat{b}} = -G$: Then $\hat{e}$ should be odd otherwise if $\hat{e} = 2\rho$ then $G = -\left(\frac{C_a}{H^{\hat{r_a}}}\right)^{2\rho} \mathsf{Acc}^{\hat{b}}$ would be a non-quadratic residue. So $\mathcal{E}$ sets $\tilde{D} \leftarrow -\frac{C_a}{H^{\hat{r_a}}}$, $\tilde{e} \leftarrow \hat{e} := \frac{\Delta s_e}{\Delta c}$, $\tilde{r} \leftarrow \hat{r} := \frac{\Delta s_r}{\Delta c}$ and $\tilde{b} \leftarrow \hat{b} := \frac{\Delta s_b}{\Delta c}$. It is clear that $\tilde{D}^{\tilde{e}} \mathsf{Acc}^{\tilde{b}} = G$.

Finally the $\mathcal{E}$ outputs $(\tilde{e}, \tilde{r}, \tilde{D}, \tilde{b})$.

Now we show that the probability the extractor terminates with outputting a valid witness is $O(\epsilon)$. If the extractor does not abort then it clearly outputs a valid witness (under factoring assumption). For the first abort, with a standard argument it can be shown that the extractor is able to extract two accepting transcripts with probability $O(\epsilon)$ (for the probabilistic analysis we refer to [85]). Thus $Pr[\mathsf{abort1}] = 1 - \mathsf{O}(\epsilon)$. For the aborts abort 2a, abort 2b, abort 2c and abort 2d they happen with negligible probability ($\leq \frac{2}{1-2^{-\lambda_s+1}} Pr[\mathcal{B}$ solves $sRSA]$ each, for any PPT adversary $\mathcal{B}$) under the strong RSA assumption according to lemma 6. For abort 2e we show in the

lemma below that in case it happens an adversary can solve the strong RSA problem. Putting them together the probability of success of $\mathcal{E}$ is at least $O(\epsilon) - \left(\frac{8}{1-2^{-\lambda_s+1}} + 1\right) Pr[\mathcal{B} \text{ solves } sRSA] = O(\epsilon) - \mathsf{negl}(\lambda_s)$.

Lemma 8. *If Δc divides Δs_b then it also divides $\Delta \rho_B$ under the strong RSA assumption.*

Proof. An adversary to the strong RSA assumption receives $H \in \mathsf{QR}_N$ and does the following: set $G = H^\tau$ for $\tau \leftarrow_\$ [0, 2^{\lambda_s} N^2]$ and send (G, H) to the adversary $\mathcal{A}$ which outputs a proof π_{Coprime2}. Then we rewind to get another successful proof π'_{Coprime2} and we use the extractor as above to get $C_B^{\Delta c} = \mathsf{Acc}^{\Delta s_b} H^{\Delta s_{\rho_B}}$.

Assume that $\Delta c \nmid \Delta \rho_B$. Since Δc divides Δs_b then there is a k such that $k \cdot \Delta c = \Delta s_b$. Then $C_B^{\Delta c} = \mathsf{Acc}^{k \cdot \Delta c} H^{\Delta s_{\rho_B}} \Rightarrow \left(C_B \mathsf{Acc}^{-k}\right)^{\Delta c} = H^{\Delta s_{\rho_B}}$. From assumption Δc doesn't divide $\Delta \rho_B$, so $\gcd(\Delta c, \Delta \rho_B) = g$ for a $g \neq \Delta c, \Delta \rho_B$. Hence, there are there are χ, ψ such that $\chi \cdot \Delta c + \psi \cdot \Delta \rho_B = g$. Thus, $H^g = H^{\chi \cdot \Delta c + \psi \cdot \Delta \rho_B} = H^{\chi \Delta c} \left(C_B \mathsf{Acc}^{-k}\right)^{\psi \Delta c} = \left(H^\chi C_B^\psi \mathsf{Acc}^{-\psi k}\right)^{\Delta c}$ so $H = \pm \left(H^\chi C_B^\psi \mathsf{Acc}^{-\psi k}\right)^{\frac{\Delta c}{g}}$. Now since H and Acc are quadratic residues (and so is C_B) we get that $H = \left(H^\chi C_B^\psi \mathsf{Acc}^{-\psi k}\right)^{\frac{\Delta c}{g}}$ and thus $\left(H^\chi C_B^\psi \mathsf{Acc}^{-\psi k}, \frac{\Delta c}{g}\right)$ is a solution to the strong RSA problem. □

By a simple argument identical to the one of section 4.4.4, we can also conclude about the range of the extracted $\tilde{e}$: $s_e \stackrel{?}{\in} \left[-2^{\lambda_z+\lambda_s+\mu+1}, 2^{\lambda_z+\lambda_s+\mu+1}\right]$ implies $-2^{\lambda_z+\lambda_s+\mu+2} \leq \hat{e} \leq 2^{\lambda_z+\lambda_s+\mu+2}$. □

4.6 A CP-SNARK for Set Membership in Bilinear Groups

In this section we propose another CP-SNARK, called $\mathsf{MemCP}_{\mathsf{VC}}$, for the set membership relation that works in bilinear groups. Unlike the schemes of Section 4.4, the CP-SNARK given in this section does not have short parameters; specifically it has a CRS linear in the size of the sets to be committed. On the other hand, it enjoys other features that are not satisfied by our previous schemes (nor by other schemes in the literature): first, it works solely in Bilinear Groups without having to deal with RSA groups; second, it allows to commit the set in an hiding manner and, for the sake of soundness, does not need to be opened by the adversary. This is possible thanks to the fact that the set is committed in a way that (under a knowledge assumption) guarantees that the prover knows the set.

More in detail, $\mathsf{MemCP}_{\mathsf{VC}}$ is a CP-SNARK for set membership where set elements are elements from the large field $\mathbb{F} = \mathbb{Z}_q$ where q is the order of bilinear groups. So $\mathcal{D}_{\mathsf{elm}} = \mathbb{F}$. In terms of set it supports all the subsets of $2^{\mathcal{D}_{\mathsf{elm}}}$ of cardinality bounded by n, $\mathcal{D}_{\mathsf{set}} = \{S \in 2^{\mathcal{D}_{\mathsf{elm}}} : \#S \leq n\}$, which we denote by $\mathcal{S}_n$, $\#$ symbol denotes the cardinality of a set. So S has elements in $\mathbb{F}$ and is a subset of $\mathcal{S}_n$.

4.6.1 Preliminaries and Building Blocks

Bilinear Groups. For ease of exposition we present our results with Type-1 groups where we assume that $\mathbb{G}_1 = \mathbb{G}_2$. Our results are under the $(\ell + 1)d$-Strong Diffie Hellman and the

(d, ℓ)-Extended Power Knowledge of Exponent assumptions, for which we refer the reader to [226].

A Polynomial-Pedersen Type-Based Commitment Scheme. First we present PolyCom, a type-based commitment scheme which was introduced in [61] extracted from the verifiable polynomial delegation scheme of [226]. The scheme has two types: one for ℓ-variate polynomials $f : \mathbb{F}^\ell \to \mathbb{F}$ over $\mathbb{F}$ of variable degree at most d, and one which is a standard Pedersen commitment for field elements. Let $\mathcal{W}_{\ell,d}$ be the set of all multisets of $\{1, \ldots, \ell\}$ where the cardinality of each element is at most d. The scheme is described in figure 4.14.

PolyCom scheme

- $\mathsf{Setup}(1^\lambda, \ell, d)$: samples a bilinear group of order q, $\mathsf{bp} := (q, g, \mathbb{G}_1, \mathbb{G}_T, e) \leftarrow \mathsf{BilGen}(1^\lambda)$, samples $\alpha, \beta, s_1, \ldots, s_\ell \leftarrow \mathbb{F}$. Computes $\mathsf{prk} \leftarrow \left\{g^{\prod_{i \in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}$ and $\mathsf{prk}^\alpha \leftarrow \left\{g^{\alpha \cdot \prod_{i \in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}$. Finally samples an $s_{\ell+1} \leftarrow_\$ \mathbb{F}$ and computes $h \leftarrow g^{s_{\ell+1}}$ and h^α.
  Return $\mathsf{ck} \leftarrow (\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^\alpha, g^\alpha, g^\beta, h, h^\alpha, h^\beta)$
- $\mathsf{Commit}(\mathsf{ck}, \mathsf{t}_{\mathbb{F}[\boldsymbol{s}]}, f)$: parses $\mathsf{ck} := (\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^\alpha, g^\alpha, g^\beta, h, h^\alpha, h^\beta)$ and uses $\mathsf{prk} := \left\{g^{\prod_{i \in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}$ and $\mathsf{prk}^\alpha := \left\{g^{\alpha \cdot \prod_{i \in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}$ to compute $g^{f(\boldsymbol{s})}$ and $g^{\alpha \cdot f(\boldsymbol{s})}$ respectively. Then samples a random $r_f \leftarrow_\$ \mathbb{F}$ and computes $c_{f,1} \leftarrow g^{f(\boldsymbol{s})} h^{r_f}$ and $c_{f,2} \leftarrow g^{\alpha \cdot f(\boldsymbol{s})} (h^\alpha)^{r_f}$
  Return $(c, o) \leftarrow ((c_{f,1}, c_{f,2}), r_f)$
- $\mathsf{Commit}(\mathsf{ck}, \mathsf{t}_q, y)$: parses $\mathsf{ck} := (\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^\alpha, g^\alpha, g^\beta, h, h^\alpha, h^\beta)$ and samples $r \leftarrow_\$ \mathbb{F}$. Computes $c_{y,1} \leftarrow g^y h^r$ and $c_{y,2} \leftarrow (g^\beta)^y (h^\beta)^r$ and return $(c, o) := ((c_{y,1}, c_{y,2}), r)$.
- $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_{\mathbb{F}[\boldsymbol{s}]}, c, f, o)$: parses $\mathsf{ck} := (\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^\alpha, g^\alpha, g^\beta, h, h^\alpha, h^\beta)$ and uses $\mathsf{prk} := \left\{g^{\prod_{i \in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}$ to compute $g^{f(\boldsymbol{s})}$. Parses $c := (c_{f,1}, c_{f,2})$. Output 1 iff $c_{f,1} = g^{f(\boldsymbol{s})} h^o \wedge e(c_{f,1}, g^\alpha) = e(c_{f,2}, g)$.
- $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c, y, o)$: parses $\mathsf{ck} := (\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^\alpha, g^\alpha, g^\beta, h, h^\alpha, h^\beta)$. Parses $c := (c_{y,1}, c_{y,2})$ Output 1 iff $c_{y,1} = g^y h^r \wedge e(c_{y,1}, g^\beta) = e(c_{y,2}, g)$.

Figure 4.14: PolyCom Commitment Scheme

Theorem 13. *Under the $(\ell+1)d$-Strong Diffie Hellman and the (d, ℓ)-Extended Power Knowledge of Exponent assumptions* PolyCom *is an extractable trapdoor commitment scheme.*

For the proof we refer to [61, 226].

Input-Hiding CP-SNARK for Polynomial Evaluation The main building block of our main protocol is a CP-SNARK $\mathsf{CP}_{\mathsf{PolyEval}}$ for the type-based commitment PolyCom. Loosely speaking the idea is to commit to the input $\boldsymbol{t}$ and the output y of a polynomial (with a Pedersen commitment), further commit to the polynomial f itself (with a polynomial commitment) and then prove that the opening of the committed polynomial evaluated on the opening of the committed input gives the committed output. The relation of the protocol is $R_{\mathsf{PolyEval}}((t_k)_{k \in [\ell]}, f, y)) = 1$ iff $f(t_1, \ldots, t_\ell) = y$:

$\mathbf{R} = (\mathsf{ck}, R_{\mathsf{PolyEval}})$ where $\mathbf{R}$ is over

$$(\boldsymbol{x}, \boldsymbol{w}) = ((x, c), (u, o, \omega)) = \left((\varnothing, (c_y, (c_{t_k})_{k\in[\ell]}, c_f)) \,,\, ((y, (t_k)_{k\in[\ell]}, f), (r_y, (r_{t_k})_{k\in[\ell]}, r_f), \varnothing) \right)$$

We will present a CP-SNARK for this relation, $\mathsf{CP}_{\mathsf{PolyEval}}$, in section 4.6.3. $\mathsf{CP}_{\mathsf{PolyEval}}$ is based on a similar protocol for polynomial evaluation given in [61] which was in turn based on the verifiable polynomial delegation scheme of zk-vSQL [226]. In those protocols, however, the input $\mathbf{t}$ is public whereas in ours we can keep it private and committed.

Range Proof CP-NIZK. We make use of $\mathsf{CP}_{\mathsf{range}}$, a CP-NIZK for the following relation on PedCom commitments c and two given integers $A < B$:

$$R_{\mathsf{range}}\left((c_e, A, B), (e, r_q)\right) = 1 \;\text{ iff }\; c = g^e h^{r_q} \;\wedge\; A < e_q < B$$

$\mathsf{CP}_{\mathsf{range}}$ can have various instantiations such as Bulletproofs [52].

Multilinear Extensions of vectors. Let $\mathbb{F}$ be a field and $n = 2^\ell$. The multilinear extension of a vector $\boldsymbol{a} = (a_0, \ldots, a_{n-1})$ in $\mathbb{F}$ is a polynomial $f_{\boldsymbol{a}} : \mathbb{F}^\ell \to \mathbb{F}$ with variables $x_1, \ldots, x_\ell$ defined as

$$f_{\boldsymbol{a}}(x_1, \ldots, x_\ell) = \sum_{i=0}^{n-1} a_i \cdot \prod_{k=1}^{\ell} \mathsf{select}_{i_k}(x_k)$$

where $i_\ell i_{\ell-1} \ldots i_2 i_1$ is the bit representation of i and $\mathsf{select}_{i_k}(x_k) = \begin{cases} x_k, & \text{if } i_k = 1 \\ 1 - x_k, & \text{if } i_k = 0 \end{cases}$

A property of Multilinear extension of $\boldsymbol{a}$ is that $f_{\boldsymbol{a}}(i_1, \ldots, i_\ell) = a_i$ for each $i \in [n]$.

The EDRAX Vector Commitment Scheme We describe the EDRAX Vector Commitment:

Definition 27. *Let a bilinear group* $\mathsf{bp} = (q, g, \mathbb{G}_1, \mathbb{G}_T, e) \leftarrow \mathcal{RG}(1^\lambda)$ *generated by a group generator. Let* $n = 2^\ell$ *be the length of the vector and* $2^{[\ell]}$ *be the powerset of* $[\ell] = \{1, \ldots, \ell\}$

- $\mathsf{KeyGen}(1^\lambda, n) \to (\mathsf{prk}, \mathsf{vrk}, \mathsf{upk}_0, \ldots, \mathsf{upk}_{n-1})$: *samples random* $s_1, \ldots, s_\ell \leftarrow\!\!\$\, \mathbb{F}$ *and computes* $\mathsf{prk} \leftarrow \left\{ g^{\prod_{i\in S} s_i} : S \in 2^{[\ell]} \right\}$ *and* $\mathsf{vrk} \leftarrow \{g^{s_1}, \ldots, g^{s_\ell}\}$. *For each* $i = 0, \ldots, n-1$ *computes the update key* $\mathsf{upk}_i \leftarrow \left\{ g^{\prod_{k=1}^{t} \mathsf{select}_{i_k}(s_k)} : t = 1, \ldots, \ell \right\} := \{\mathsf{upk}_{i,t} : t = 1, \ldots, \ell\}$.
- $\mathsf{Com}(\mathsf{prk}, a_0, \ldots, a_{n-1}) \to \mathsf{dig}_{\boldsymbol{a}}$: *let* $\boldsymbol{a} := (a_0, \ldots, a_{n-1})$. *Computes* $\mathsf{dig}_{\boldsymbol{a}} \leftarrow g^{f_{\boldsymbol{a}}(s_1, \ldots, s_\ell)}$ *where* $f_{\boldsymbol{a}}$ *is the multilinear extension of vector* $\boldsymbol{a}$ *as described above.*
- $\mathsf{Prove}(\mathsf{prk}, i, \boldsymbol{a}) \to (a_i, \pi_i)$: *let* $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ *be an* ℓ*-variable. Compute* $q_1, \ldots, q_\ell$ *such that* $f_{\boldsymbol{a}}(\boldsymbol{x}) - f_{\boldsymbol{a}}(i_1, \ldots, i_\ell) = \sum_{k=1}^{\ell} (x_k - i_k) q_k(\boldsymbol{x})$ *and* $\pi_i \leftarrow \left\{ g^{q_1(\boldsymbol{s})}, \ldots, g^{q_\ell(\boldsymbol{s})} \right\}$ *(where* $g^{q_i(\boldsymbol{s})}$ *is evaluated by using* $\mathsf{prk} := \left\{ g^{\prod_{i\in S} s_i} : S \in 2^{[\ell]} \right\}$ *without* $\boldsymbol{s}$*).*
- $\mathsf{Ver}(\mathsf{vrk}, \mathsf{dig}, i, a, \pi) \to b$: *parse* $\pi := (w_1, \ldots, w_\ell)$ *and outputs* 1 *iff* $e(\mathsf{dig}/g^a, g) = \prod_{k=1}^{\ell} e(g^{s_k - i_k}, w_k)$
- $\mathsf{UpdateCom}(\mathsf{dig}, i, \delta, \mathsf{upk}_i) \to \mathsf{dig}'$: *computes* $dig' \leftarrow dig \cdot \left[g^{\prod_{k=1}^{\ell} \mathsf{select}_{i_k}(s_k)} \right]^\delta := \mathsf{dig} \cdot \left[\mathsf{upk}_{i,\ell} \right]^\delta = g^{(a_i + \delta) \cdot \prod_{k=1}^{\ell} \mathsf{select}_{i_k}(s_k) + \sum_{j=0, j\neq i}^{n-1} a_j \cdot \prod_{k=1}^{\ell} \mathsf{select}_{j_k}(s_k)}$

- $\mathsf{UpdateCom}(\pi, i, a', \mathsf{upk}_i) \to \pi'$: *Parses* $\pi := (w_1, \ldots, w_\ell)$ *and computes* $w'_k \leftarrow w_k \cdot g^{\Delta_i(\boldsymbol{s})}$ *for each* $k = 1, \ldots, \ell$*, where* $\Delta_k(\boldsymbol{x})$ *are the delta polynomials computed by the* DELTAPOLYNOMIALS *algorithm (for more details about the algorithm and its correctness we refer to [73]).*

The above scheme is proven in [73] to satisfy the Soundness property under the q-Strong Bilinear Diffie-Hellman assumption.

The type-based commitment scheme of $\mathsf{MemCP}_{\mathsf{VC}}$. We define the type-based commitment $\mathsf{C}_{EdraxPed}$ for our CP-SNARK $\mathsf{MemCP}_{\mathsf{VC}}$. We recall we need a commitment that allows one to commit to both elements and sets. We build this based on a hiding variant of EDRAX Vector Commitment [73], which in turn relies on a polynomial commitment. Therefore, we use a special case of PolyCom for polynomials of maximum variable degree $d = 1$. Let $\ell := \lceil \log(n) \rceil$ and $2^{[\ell]}$ be the powerset of $[\ell] = \{1, ..., \ell\}$ then $\mathcal{W}_{\ell,1} = 2^{[\ell]}$. Furthermore, for any $n' \leq n$ let $L : \mathcal{S}_{n'} \to \mathbb{F}^{n'}$ be a function that maps a set of cardinality n' to its corresponding vector according to an ordering. The description of the scheme can be found in figure 4.15. Essentially the idea is to take the set, fix some ordering so that we can encode it with a vector, and then commit to such vector using the vector commitment of [73], which in turn commits to a vector by committing to its multilinear extension polynomial.

$\mathsf{C}_{EdraxPed}$ scheme

- $\mathsf{Setup}(1^\lambda, \ell)$: executes $\mathsf{ck} \leftarrow \mathsf{PolyCom.Setup}(1^\lambda, \ell, 1)$
- $\mathsf{Commit}(\mathsf{ck}, \mathsf{t}_S, S)$: computes $\boldsymbol{v}S \leftarrow L(S)$ and then the corresponding multilinear extension of $\boldsymbol{v}S$, $f_{\boldsymbol{v}S}$. Returns $(c, o) \leftarrow \mathsf{PolyCom.Commit}(\mathsf{ck}, \mathsf{t}_{\mathbb{F}[\boldsymbol{s}]}, f_{\boldsymbol{v}S})$.
- $\mathsf{Commit}(\mathsf{ck}, \mathsf{t}_q, y)$: returns $(c, o) \leftarrow \mathsf{PolyCom.Commit}(\mathsf{ck}, \mathsf{t}_q, y)$
- $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_S, c, S, o)$: computes $\boldsymbol{v}S \leftarrow L(S)$ and then the corresponding multilinear extension of $\boldsymbol{v}S$, $f_{\boldsymbol{v}S}$. Outputs $\mathsf{PolyCom.VerCommit}(\mathsf{ck}, \mathsf{t}_{\mathbb{F}[\boldsymbol{s}]}, c, f_{\boldsymbol{v}S}, o)$.
- $\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c, y, o)$: outputs $\mathsf{PolyCom.VerCommit}(\mathsf{ck}, \mathsf{t}_q, c, y, o)$.

Figure 4.15: The $\mathsf{C}_{EdraxPed}$ Commitment Scheme.

4.6.2 CP-SNARK for Set membership using EDRAX Vector Commitment

Here we present a CP-SNARK for set membership that uses a Vector Commitment - an EDRAX [73] variant - to commit to a set. The idea is to transform a set to a vector (using for example lexicographical order) and then commit to the vector with a vector commitment. Then the set membership is proven with a zero knowledge proof of opening of the corresponding position of the vector. However to preserve zero knowledge we additionally need to hide the position of the element. For this we construct a zero knowledge proof of knowledge of an opening of a position that does not give out the position. Finally, since the position is hidden we additionally need to ensure that the prover is not cheating by providing a proof for a position that exceeds the length of the vector. For this we, also, need a proof of range for the position, i.e. that $i < n$.

In this section the domain of the elements is a field, $\mathcal{D}_{\mathsf{elm}} := \mathbb{F}$, and the domain of the set is all the subsets of $2^{\mathcal{D}_{\mathsf{elm}}}$ of cardinality bounded by n, $\mathcal{D}_{\mathsf{set}} = \{S \in 2^{\mathcal{D}_{\mathsf{elm}}} : \#S \leq n\}$, which we

denote by $\mathcal{S}_n$ (the $\#$ symbol denotes the cardinality of a set). So S has elements in $\mathbb{F}$ and is a subset of $\mathcal{S}_n$.

The type-based commitment of our scheme is $\mathsf{C}_{EdraxPed}$ (fig. 4.15) that is presented in the previous section, and the relation is

$\mathbf{R} = (\mathsf{ck}, R_{\mathsf{VCmem}})$ where $\mathbf{R}$ is over

$$(\boldsymbol{x}, \boldsymbol{w}) = ((x, c), (u, o, \omega)) = \\ = \left(\left(\#S, (c_y, (c_{i_k})_{k\in[\ell]}, c_S)\right), \left((y, (i_k)_{k\in[\ell]}, S), (r_y, (r_{i_k})_{k\in[\ell]}, r_S), \varnothing\right)\right)$$

$R_{\mathsf{VCmem}}(\#S, \left(y, (i_k)_{k\in[\ell]}, S\right)) = 1$ iff $y = L(S)[i] \wedge i < \#S \wedge i = \sum_{k=1}^{\ell} i_k 2^{k-1}$

Note that in the above the prover should normally give exactly $\ell = \lceil \log(\#S) \rceil$ commitments. In case $\ell < \lceil \log(\#S) \rceil$ the position is not fully hiding since it is implicit that $i < 2^{\ell-1}$ so the verifier gets a partial information about the position.

For this we will compose a CP-SNARK $\mathsf{CP}_{\mathsf{PolyEval}}$ and a CP-NIZK $\mathsf{CP}_{\mathsf{range}}$ for the relations $R_{\mathsf{PolyEval}}((i_k)_{k\in[\ell]}, f, y)) = 1$ iff $f(i_1, \ldots, i_\ell) = y$ and $R_{\mathsf{range}}(T, (i_k)_{k\in[\ell]}) = 1$ iff $i < T$ respectively and the commitment scheme $\mathsf{C}_{EdraxPed}$. So $\mathsf{CP}_{\mathsf{VCmem}}$ is a conjuction of the former, where the common commitments are $(c_{i_k})_{k\in[\ell]}$.

CP-SNARK for R_{VCmem}

- $\mathsf{KeyGen}(\mathsf{ck}, R_{\mathsf{VCmem}})$: computes $(\mathsf{ek}_1, \mathsf{vk}_1) \leftarrow \mathsf{CP}_{\mathsf{PolyEval}}.\mathsf{KeyGen}(\mathsf{ck}, R_{\mathsf{PolyEval}})$ and $(\mathsf{ek}_2, \mathsf{vk}_2) \leftarrow \mathsf{CP}_{\mathsf{range}}.\mathsf{KeyGen}(\mathsf{ck}, R_{\mathsf{range}})$

  Return $(\mathsf{ek}, \mathsf{vk}) \leftarrow ((\mathsf{ek}_1, \mathsf{ek}_2), (\mathsf{vk}_1, \mathsf{vk}_2))$
- $\mathsf{Prove}(\mathsf{ek}, \#S, (c_y, (c_{i_k})_{k\in[\ell]}, c_S), (y, (i_k)_{k\in[\ell]}, S), (r_y, (r_{i_k})_{k\in[\ell]}, r_S), \varnothing)$: Parse $\mathsf{ek} := (\mathsf{ek}_1, \mathsf{ek}_2)$ and compute $\pi_1 \leftarrow \mathsf{CP}_{\mathsf{PolyEval}}.\mathsf{Prove}\big(\mathsf{ek}_1, \varnothing, (c_y, (c_{i_k})_{k\in[\ell]}), c_S), (y, (i_k)_{k\in[\ell]}, S), (r_y, (r_{i_k})_{k\in[\ell]}, r_S), \varnothing\big)$.

  Parse $\mathsf{ck} := (\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^{\alpha}, g^{\alpha}, g^{\beta}, h, h^{\alpha}, h^{\beta})$ and further $\mathsf{bp} := (q, g, \mathbb{G}_1, \mathbb{G}_T, e)$ to get (g, h), then compute $i \leftarrow \sum_{k=1}^{\ell} i_k 2^{k-1}$ and $r_i \leftarrow \sum_{k=1}^{\ell} r_{i_k} 2^{k-1}$ and the corresponding commitment $c_i \leftarrow g^i h^{r_i}$. Notice that c_i is a commitment to i with $o = r_i$.

  Compute $\pi_2 \leftarrow \mathsf{CP}_{\mathsf{range}}.\mathsf{Prove}\left(\mathsf{ek}_1, (1, \#S), c_i, i, r_i, \varnothing\right)$

  Return $\pi = (\pi_1, \pi_2)$
- $\mathsf{VerProof}(\mathsf{vk}, \#S, (c_y, (c_{i_k})_{k\in[\ell]}, c_S), \pi)$: parses $\mathsf{vk} := (\mathsf{vk}_1, \mathsf{vk}_2)$ and $\pi := (\pi_1, \pi_2)$. Then computes homomorhically $c_{i,1} \leftarrow \prod_{k=1}^{\ell} (c_{i_k,1})^{2^{k-1}}$ and $c_{i,2} \leftarrow \prod_{k=1}^{\ell} (c_{i_k,2})^{2^{k-1}}$.

  Return 1 iff
  $\mathsf{CP}_{\mathsf{PolyEval}}.\mathsf{VerProof}(\mathsf{vk}_1, \varnothing, (c_y, (c_{i_k})_{k\in[\ell]}), c_S), \pi_1) \wedge$
  $\mathsf{CP}_{\mathsf{range}}.\mathsf{VerProof}(\mathsf{vk}_2, (1, \#S), c_i, \varnothing)$.

Figure 4.16: $\mathsf{MemCP}_{\mathsf{VC}}$.

Theorem 14. *Let* $\mathsf{CP}_{\mathsf{PolyEval}}$ *and* $\mathsf{CP}_{\mathsf{range}}$ *be zero knowledge CP-SNARKs for the relations* R_{PolyEval} *and* R_{range} *respectively under the commitment scheme* PolyCom *then the above scheme is a zero knowledge CP-SNARK for the relation* R_{VCmem} *and the commitment scheme* $\mathsf{C}_{EdraxPed}$*. Further it is a CP-SNARK for* R_{mem} *under the same commitment scheme.*

Proof. Zero Knowledge comes directly from the zero knowledge of $\mathsf{CP}_{\mathsf{PolyEval}}$ and $\mathsf{CP}_{\mathsf{PolyEval}}$.

For Knowledge Soundness, let an adversary $\mathcal{A}(\mathbf{R}, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$ outputting $(x, c) := \big(\#S,\ (c_y, (c_{i_k})_{k\in[\ell]}, c_S)\big)$ and π such that $\mathsf{VerProof}(\mathsf{vk}, \#S, (c_y, (c_{i_k})_{k\in[\ell]}, c_S), \pi) = 1$. We will construct an extractor $\mathcal{E}$ that on input $(\mathbf{R}, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$ outputs a valid witness $w := \big((y, (i_k)_{k\in[\ell]}, S), (r_y, (r_{i_k})_{k\in[\ell]}, r_S), \varnothing\big)$.

$\mathcal{E}$ uses the extractors of $\mathcal{E}_{\mathsf{PolyEval}}$, $\mathcal{E}_{\mathsf{range}}$ of $\mathsf{CP}_{\mathsf{PolyEval}}$ and $\mathsf{CP}_{\mathsf{range}}$. $\mathcal{E}_{\mathsf{PolyEval}}$ outputs $(y, (i_k)_{k\in[\ell]}, f)$, $(r_y, (r_{i_k})_{k\in[\ell]}, r_f)$ such that $f(i_1, \ldots, i_\ell) = y \wedge \mathsf{PolyCom.VerCommit}(\mathsf{ck}, \mathsf{t}_{\mathbb{F}[\boldsymbol{s}]}, c_S, f, r_f) = 1 \wedge \mathsf{PolyCom.VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_y, y, r_y) = 1 \bigwedge_{k=1}^{\ell} \mathsf{PolyCom.VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_{i_k}, i_k, r_{i_k}) = 1$. Further, from the Extended Power Knowledge of Exponent assumption we know that f is an ℓ-variate polynomial of maximum variable degree 1. Therefore it corresponds to a multilinear extension of a unique vector $\boldsymbol{v}S$, which is efficiently computable. The extractor computes the vector $\boldsymbol{v}S$ from f and the corresponding set S. It is clear that, since f is the multilinear extension of the S and $\mathsf{PolyCom.VerCommit}(\mathsf{ck}, \mathsf{t}_{\mathbb{F}[\boldsymbol{s}]}, c_S, f, r_f) = 1$, $\mathsf{C}_{EdraxPed}.\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_S, c_S, S, r_f) = 1$. $\mathsf{C}_{EdraxPed}.\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_y, y, r_y) = 1 \bigwedge_{k=1}^{\ell} \mathsf{C}_{EdraxPed}.\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_{i_k}, i_k, r_{i_k}) = 1$ is straightforward from the definition of the $\mathsf{C}_{EdraxPed}$ commitment scheme for field elements type.

$\mathcal{E}$ uses the extractor of the commitment scheme $\mathsf{PolyCom}$, $\mathcal{E}_{\mathsf{PolyCom}}$, that outputs for each $k = 1, \ldots, \ell$ i_k, r_{i_k} such that $c_{i_k,1} = g^{i_k} h^{r_{i_k}} \wedge e(c_{i_k,1}, g^{\beta}) = e(c_{i_k,2}, g)$ or $\mathsf{C}_{EdraxPed}.\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_{i_k}, r_{i_k}) = 1$. $\mathcal{E}_{\mathsf{range}}$ outputs (i, r_i) such that $i < \#S \wedge \mathsf{PolyCom.VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_i, i, r_i) = 1$ which means that $c_{i,1} = g^i h^{r_i}$. Since the proof π is verified then $c_{i,1} = \prod_{k=1}^{\ell} (c_{i_k,1})^{2^{k-1}}$ or $g^i h^{r_i} = g^{\sum_{k=1}^{\ell} i_k 2^{k-1}} h^{\sum_{k=1}^{\ell} r_{i_k} 2^{k-1}}$. From the binding property of the Pedersen commitment we get that $i = \sum_{k=1}^{\ell} i_k 2^{k-1}$ and $r_i = \sum_{k=1}^{\ell} r_{i_k} 2^{k-1}$.

Putting them together the extractor outputs $\big((y, (i_k)_{k\in[\ell]}, S), (r_y, (r_{i_k})_{k\in[\ell]}, r_f), \varnothing\big)$ such that $\mathsf{C}_{EdraxPed}.\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_y, r_y) = 1 \bigwedge_{i=1}^{\ell} \mathsf{C}_{EdraxPed}.\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_q, c_{i_k}, r_{i_k}) = 1 \wedge$
$\mathsf{C}_{EdraxPed}.\mathsf{VerCommit}(\mathsf{ck}, \mathsf{t}_S, c_f, S, r_f) = 1$ and further $y = L(S)[i] \wedge i < \#S \wedge i = \sum_{k=1}^{\ell} i_k 2^{k-1}$. It is straightforward that $y = L(S)[i] \wedge i < \#S$ means that $y \in S$ which leads to $R_{\mathsf{mem}}(y, S) = 1$. □

4.6.3 Input-hiding CP-SNARKs for Polynomial Evaluation

Here, we present an instantiation of a a zero knowledge CP-SNARK for the relation R_{PolyEval} presented in section 4.6.1.

To give an intuition of the protocol we recall that zk-vSQL uses lemma 9 to prove the correct evaluation of the polynomial, that we recall below.

Lemma 9 ([170]). *Let $f : \mathbb{F}^\ell \to \mathbb{F}$ be a polynomial of variable degree d. For all $\boldsymbol{t} := (t_1, \ldots, t_\ell) \in \mathbb{F}^\ell$ there exist efficiently computable polynomials $q_1, \ldots, q_\ell$ such that: $f(\boldsymbol{z}) - f(\boldsymbol{t}) = \sum_{i=1}^{\ell} (z_i - t_i) q_i(\boldsymbol{z})$.*

With this one can verify in time linear in the number of variables that $f(\boldsymbol{t}) = y$ by checking iff $g^{f(\boldsymbol{t})} g^{-y} = \prod_{i=1}^{\ell} e(g^{s_i}, w_i)$, given the values $g^{f(\boldsymbol{s})}, \{g^{s_i}\}_{i=1}^{\ell}, \{w_i = g^{q_i(\boldsymbol{s})}\}_{i=1}^{\ell}$ We are interested in the committed values of $f, y = f(\boldsymbol{t})$ and $\boldsymbol{t}$, c_f, c_y, c_t respectively, that hide them. For this we will use instead the equation below for verification:

$$
\begin{aligned}
\big(f(\boldsymbol{z}) + r_f z_{\ell+1}\big) - \big(f(\boldsymbol{t}) + r_y z_{\ell+1}\big) = \\
&\sum_{k=1}^{\ell} (z_k - t_k) q_k(\boldsymbol{z}) + z_{\ell+1}(r_f - r_y) = \\
&\sum_{k=1}^{\ell} (z_k - t_k)(q_k(\boldsymbol{z}) + r_k z_{\ell+1}) + z_{\ell+1} \left(r_f - r_y - \sum_{k=1}^{\ell} r_k (z_k - t_k) \right) = \\
&\sum_{k=1}^{\ell} [z_k - (t_k + r_{t_k} z_{\ell+1})] \cdot [q_k(\boldsymbol{z}) + r_k z_{\ell+1}] + \\
&\quad + z_{\ell+1} \left(r_f - r_y - \sum_{k=1}^{\ell} r_k (z_k - t_k) + \sum_{k=1}^{\ell} r_{t_k} [q_k(\boldsymbol{z}) + r_k z_{\ell+1}] \right)
\end{aligned}
$$

The equation indicates us how to construct the protocol which we present in figure 4.17.

CP-SNARK for R_{PolyEval}

- $\mathsf{KeyGen}(\mathsf{ck}, R_{\mathsf{PolyEval}})$: Parse $\mathsf{ck} := (\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^\alpha, g^\alpha, g^\beta, h, h^\alpha, h^\beta)$ and compute $\mathsf{vrk} \leftarrow \{g^{s_1}, \ldots, g^{s_\ell}\}$.
  Return $(\mathsf{ek}, \mathsf{vk}) \leftarrow \left(\left(\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^\alpha, g^\alpha, g^\beta, h, h^\alpha, h^\beta\right) , \left(\mathsf{bp}, \mathsf{vrk}, g^\alpha, g^\beta, h\right) \right)$
- $\mathsf{Prove}(\mathsf{ek}, \varnothing, (c_y, (c_{t_k})_{k\in[\ell]}, c_f), (y, (t_k)_{k\in[\ell]}, f), (r_y, (r_{t_k})_{k\in[\ell]}, r_f), \varnothing)$: Let

$$\begin{aligned}\mathsf{ck} &:= (\mathsf{bp}, \mathsf{prk}, \mathsf{prk}^\alpha, g^\alpha, g^\beta, h, h^\alpha, h^\beta)\\ &:= \Big((q, g, \mathbb{G}_1, \mathbb{G}_T, e), \left\{g^{\Pi_{i\in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}, :=\\ &\qquad \left\{g^{\alpha\cdot\Pi_{i\in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}, g^\alpha, g^\beta, g^{s_{\ell+1}}, g^{\alpha s_{\ell+1}}, g^{\beta s_{\ell+1}}\Big)\end{aligned}$$

  and

  1. Sample $r_1, ..., r_\ell \leftarrow \mathbb{F}$ and compute $q_1, \ldots, q_\ell$ such that

$$\begin{aligned}&(f(\boldsymbol{z}) + r_f z_{\ell+1}) - (f(\boldsymbol{t}) + r_y z_{\ell+1}) =\\ &= \sum_{k=1}^{\ell} [z_k - (t_k + r_{t_k} z_{\ell+1})] \cdot [q_k(\boldsymbol{z}) + r_k z_{\ell+1}] +\\ &\quad + z_{\ell+1}\left(r_f - r_y - \sum_{k=1}^{\ell} r_k(z_k - t_k) + \sum_{k=1}^{\ell} r_{t_k}\left[q_k(\boldsymbol{z}) + r_k z_{\ell+1}\right]\right)\end{aligned}$$

     By using $\mathsf{prk} := \left\{g^{\Pi_{i\in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}$ and h compute $w_k = g^{q_k(\boldsymbol{s}) + r_k s_{\ell+1}}$ for each $k = 1, \ldots, \ell$ and $w_{\ell+1} = g^{r_f - r_y - \sum_{k=1}^{\ell} r_k(s_k - t_k) + \sum_{k=1}^{\ell} r_{t_k}[q_k(\boldsymbol{s}) + r_k s_{\ell+1}]}$
  2. By using $\mathsf{prk}^\alpha := \left\{g^{\alpha\cdot\Pi_{i\in W} s_i} : W \in \mathcal{W}_{\ell,d}\right\}$ and h^α compute $w'_k = g^{\alpha\cdot(q_k(\boldsymbol{s}) + r_k s_{\ell+1})}$ for each $k = 1, \ldots, \ell$ and $w'_{\ell+1} = g^{\alpha\cdot\left(r_f - r_y - \sum_{k=1}^{\ell} r_k(s_k - t_k) + \sum_{k=1}^{\ell} r_{t_k}[q_k(\boldsymbol{s}) + r_k s_{\ell+1}]\right)}$

  Return $\pi = \{w_1, ..., w_\ell, w_{\ell+1}, w'_1, ..., w'_\ell, w'_{\ell+1}\}$
- $\mathsf{VerProof}(\mathsf{vk}, \varnothing, (c_y, (c_{t_k})_{k\in[\ell]}, c_f), \pi)$: Parse $\pi := \{w_1, ..., w_\ell, w_{\ell+1}, w'_1, ..., w'_\ell, w'_{\ell+1}\}$, $\mathsf{vk} := \left(\mathsf{bp}, \mathsf{vrk}, g^\alpha, g^\beta, h\right)$ and $c_y := (c_{y,1}, c_{y,2})$, $c_{t_k} := (c_{t_k,1}, c_{t_k,2})$ for each $k = 1, \ldots, \ell]$ and $c_f := (c_{f,1}, c_{f,2})$
  Return 1 iff
  1. $e(c_{y,1}, g^\beta) = e(c_{y,2}, g)$
  2. $e(c_{f,1}, g^\alpha) = e(c_{f,2}, g)$
  3. $e(c_{t_k,1}, g^\beta) = e(c_{t_k,2}, g)$ for all $k = 1, ..., \ell$
  4. $e(w_k, g^\alpha) = e(w'_k, g)$ for all $k = 1, ..., \ell, \ell+1$
  5. $e\left(c_f \cdot c_y^{-1}, g\right) = \prod_{k=1}^{\ell} e\left(g^{s_k} c_{t_k}^{-1}, w_k\right) \cdot e\left(g^{s_{\ell+1}}, w_{\ell+1}\right)$

Figure 4.17: Our CP-SNARK instantiation for the R_{PolyEval} relation.

Theorem 15. *Under the $(\ell+1)d$-Strong Diffie Hellmann and the (d,ℓ)-Extended Power Knowledge of Exponent assumptions, $\mathsf{CP}_{\mathsf{PolyEval}}$ is a Knowledge Extractable CP-SNARK for the relation R_{PolyEval} and the commitment scheme* PolyCom.

Proof. Below is a proof sketch, which however is quite similar to the one of $\mathsf{CP}_{\mathsf{poly}}$ in [61].

Knowledge Soundness. The proof comes directly from Evaluation Extractability of vSQL (see [226]) with the difference that here t_k for each $k \in [\ell]$ should also be extracted. However, its extraction is straightforward from the extractability of the commitment scheme.

Zero-Knowledge. Consider the following proof simulator algorithm $\mathcal{S}_{\mathsf{prv}}(\mathsf{td}, c_f, (c_{t_k})_{k\in[\ell]}, c_y)$:

- Use td to get α and $s_{\ell+1}$.
- For $k = 1$ to ℓ, sample $\xi_k \leftarrow_{\$} \mathbb{Z}_q$ and sets $w_k \leftarrow g^{\xi_k}$.
- Compute $w_{\ell+1}$ such that $e\left(c_{f,1} \cdot c_{y,1}^{-1}, g\right) = \prod_{k=1}^{\ell} e\left(g^{s_k} c_{t_k,1}^{-1}, w_k\right) \cdot e\left(g^{s_{\ell+1}}, w_{\ell+1}\right)$ holds. That is: $w_{\ell+1} \leftarrow \left(c_f \cdot c_y^{-1} \cdot \prod_{k=1}^{\ell} \left(g^{-s_k} c_{t_k,1}\right)^{\xi_k}\right)^{s_{\ell+1}^{-1}}$
- Use α to compute $w'_k = w_k^a$ for all $k \in [\ell+1]$
- Return $\{w_1, ..., w_\ell, w_{\ell+1}, w'_1, \ldots, w'_\ell, w'_{\ell+1}\}$

It is straightforward to check that proofs created by $\mathcal{S}_{\mathsf{prv}}$ are identically distributed to the ones returned by $\mathsf{CP}_{\mathsf{PolyEval}}.\mathsf{Prove}$. $(w_k)_{k\in[\ell]}$'s are uniformely distributed in both cases. For $w_{\ell+1}$ there is a function W such that $w_{\ell+1} = W(c_{f,1}, c_{y,1}, \mathsf{vk}, (c_{t_k,1})_{k\in[\ell]}, (w_k)_{k\in[\ell]})$ in both cases. Since the inputs are either identical or identically distributed, the outputs $w_{\ell+1}$ are also identically distributed in the case of of $\mathcal{S}_{\mathsf{prv}}$ and $\mathsf{CP}_{\mathsf{PolyEval}}.\mathsf{Prove}$. □

4.7 Applications

In this section, we discuss applications of our solutions for proving set (non-)membership in a succinct and modular way.

As one can note, in our solutions the set of committed elements is public and not hidden to the verifier. Nevertheless, our solutions can still capture some applications in which the "actual" data in the set is kept private. This is for example the case of anonymous cryptocurrencies like Zerocash. In this scenario, the public set of elements to be accumulated, U, is derived by creating a commitment to the underlying data, X, e.g., $u = COMM(x)$. To support this setting, we can use our solutions for arbitrary elements (so supporting virtually any commitment scheme). Interestingly, though, we can also use our (more efficient) solution for sets of primes if commitments are prime numbers. This can be done by using for example the *hash-to-prime* method described in Section 4.4.1 or another method for Pedersen commitments that we explain below in the context of Zerocash.

We now discuss concrete applications for which our constructions are suitable, both for set-membership and set non-membership. In particular these are applications in which: (1) the prover time must be small; (2) the size of the state (i.e.: the accumulator value and commitments) must be small (potentially constant); (3) the verifier time should be small; and (4) the time to

update the accumulator—adding or deleting an element—should be fast. As we discuss below, our RSA-based constructions are suitable candidate for settings with these constraints.

Zerocash. Zerocash [24] is a UTXO-type (Unspent Transaction Output) cryptocurrency protocol which extends Bitcoin with privacy-preserving (*shielded*) transactions. When performing a shielded transaction users need to prove they are spending an output note from a token they had previously received. Users concerned with privacy should not reveal which note they are spending, else their new transaction could be linked to the original note that contained the note commitment. This would reveal information *both to the public and the sender of the initial transaction, and hence partially reveal the transaction graph*. In order to keep transactions unlinkable, the protocol uses zkSNARKs to prove a set membership relation, namely that a note commitment is in a publicly known set of "usable" note commitments.

Zcash is a full-fledged digital currency using Zerocash as the underlying protocol. In its current deployment, Sapling [132], it employs Pedersen commitments of the notes and makes a zero-knowledge set membership proof of these commitments using a Pedersen-Hash-based Merkle tree approach. This is the part of the protocol that can be replaced by one of our RSA-based solutions in order to obtain a speedup in proving time. In particular, we could slightly modify the note commitments in order to enable the use of our scheme $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ for sets of prime numbers, which gives the best efficiency. We can proceed as follows. Let us recall that the note commitments are represented by their x coordinates in the underlying elliptic curve group. We can then modify them so that the sender chooses a blinding factor such that the commitment representation of a note is a prime number, and we can add a consensus rule that enforces this check. With this change, we can achieve a solution that is significantly more efficient than that currently used in Zcash. Currently Zcash uses a Merkle Tree whose depth is 32. In this setting, *we would be able to reduce proving time of set-membership from* 1.12*s to* 54.51*ms*, trading it for larger proof sizes. We note that in this application, the set-membership proof about $u \in S$ is accompanied by another predicate $P(u)$. In the proof statement of the Zcash protocol, proving that $P(u)$ is satisfied takes considerably less time than the membership proof, hence this is why our solution would improve the overall proving time considerably, albeit the proof having more components. Another interesting comment is that our solution significantly reduces the size of the circuit, hence the need of a succinct proof system is reduced and one may even consider instantiations with other proof systems, such as Bulletproofs, that would offer transparency at the price of larger proofs and verification time.

Asset Governance. In the context of *blockchain-based asset transfers protocols*, a governance system must be established to determine who can create new assets. In many cases these assets must be publicly traceable (i.e., their total supply must be public), yet in others, where the assets can be issued privately, validators still need to verify that the assets were issued by an authorized issuer. Specifically, there may be a public set of rules, X (where a rule $= (pk, [a, b])$), defining which entities (public keys) are allowed to issue which assets (defined by a range of *asset types*), forming an "issuance whitelist". When one of those issuers wants to issue a new asset, they need to prove (in zero knowledge) that their public key belongs to the issuance whitelist, which entails set membership, as well as prove that the asset type they issued is within the allowed range of asset types (as defined in the original rule). In this case, the accumulated set of rules is public to all, and this public information may also include a mapping between rules and prime numbers. Our RSA-based scheme for sets of primes (Section 4.4.3) can suit this scenario.

Anonymous Broadcast. In a peer-to-peer setting, anonymous broadcast allows users in a group

to broadcast a message without revealing their identity. They can only broadcast *once* on each topic. One approach described in [1] works by asking users to put down a deposit which they will lose if they try and broadcast multiple messages on the same topic. In this approach users joining a group deposit their collateral in a smart contract. Whoever has the private key used by the client for the deposit can claim the sum. The approach in [1] makes sure that the key is leaked if one broadcasts more than one message. To enforce this leakage we require that at broadcast time users *(i)* derive an encryption key K that depends on their private key and the topic, and *(ii)* compute an encryption of the private key by the newly derived K. Then the users publish both the ciphertext and a secret share of the encryption key K, and prove (in zero-knowledge) their public key is part of the group and that *(i)* and *(ii)* were performed correctly. Which specific share needs to be revealed depends on the broadcasted message, thus making it likely two different shares will be leaked for two different messages.

This way, broadcasting multiple messages on the same topic reveals the user's private key, allowing other users to remove them from the group by calling a function in the smart contract and receive part of the deposit.

A particularly interesting use case for anonymous broadcast is that in which the group is comprised of validators participating in a consensus algorithm, who would like to broadcast messages without exposing their node's identity and thus prevent targeted DoS attacks. This setting requires proofs to be computed extremely fast while verification performance requirements are less strict. Our $\mathsf{MemCP}_{\mathsf{RSAPrm}}$ can satisfy these performance requirements trading for a modest increase in proof size.

Financial Identities. In the financial world, regulations establish that financial organizations must *know* who their costumers are [102]. This is called a KYC check and allows to reduce the risk of fraud. Some common practices for KYC often undermines user privacy as they involve collecting a lot of personal information on them. Zero-knowledge proofs allow for an alternative approach. In modern systems, one can expect that individuals or companies will be able to prove that they belong to a set of accepted or legitimate identities. A privacy-preserving KYC check would then be reduced to generating a set-membership proof in zero-knowledge. Often some further information is required, e.g. the credit score of the individual. In such cases our CP-SNARK for set membership can be combined with one proving an additional predicate $P(\mathsf{id})$ on the identity in a modular fashion.

Regarding applications of non-membership proofs, we expand on the well-known concept of "blacklists", where identities (or credentials) must be shown to *not* belong to a certain set of identities (or credentials). As an example, in the context of financial identities, anti-money laundering regulations (AML) [189] require customers not to be in a list of fraudulent identities. Here one can use our non-membership construction to generate a proof that the customer does not belong to the set of money launderers (or those thought to be). Because, as in the set-membership case, a user may have to prove additional information about their identity, here we can also benefit from a modular framework. Furthermore, modularity allows us to cheaply prove both membership and non-membership (at the same time) for the same identity id together with some additional information $P(\mathsf{id})$: holding commitment $c(\mathsf{id})$ one can produce the following tuple of proofs: (1) a membership proof ($\mathsf{id} \in S$); (2) a non-membership proof ($\mathsf{id} \notin S'$); (3) a CP-SNARK proof that includes the statement to be proven on that identity ($P(\mathsf{id})$).

We note that in some cases, a central authority, who controls the white and black lists, is trusted to ensure the integrity of the lists. This means that the identities can be added or removed from the lists, which means that our RSA-based construction is ideal given the comparatively

reduced cost of updating the dynamic accumulator.

Zerocoin Vulnerability. Another specific application of our RSA-based constructions is that of solving the security vulnerability of the implementation of the Zerocoin protocol [164] used in the Zcoin cryptocurrency [218]. The vulnerability in a nutshell: when proving equality of values committed under the RSA commitment and the prime-order group commitment, the equality may not hold over the integers, and hence one could easily produce collisions in the prime order group. Our work can provide different ways to solve this problem by generating a proof of equality over the integers.

4.8 Instantiation over Hidden Order Groups

In sections 4.4, 4.5 we construct zero knowledge protocols for set membership/non-membership, where the sets are committed using an RSA accumulator. The integer commitment scheme IntCom, the RSA accumulator-based commitments to sets $\mathsf{SetCom}_{\mathsf{RSA}}$, $\mathsf{SetCom}_{\mathsf{RSA}'}$, the proof of equality modEq, the argument of knowledge of a root Root and the argument of knowledge of coprime element $\mathsf{Coprime}$ are all working over RSA groups.

Although in our work above we specify the group to be an RSA group, we note that our protocols can also work over any Hidden Order Group. For example Class Groups [49] or the recently proposed groups from Hyperelliptic Curves [94, 146].

Here we describe the (slight) modifications, in the protocols and the assumptions under which they would be secure, that are necessary to switch to (general) Hidden Order Groups.

Let $\mathsf{Ggen}(1^\lambda)$ be a probabilistic algorithm that generates such a group $\mathbb{G}$ with order in a specific range $[\mathsf{minord}(\mathbb{G}), \mathsf{maxord}(\mathbb{G})]$ such that $\frac{1}{\mathsf{minord}(\mathbb{G})}, \frac{1}{\mathsf{maxord}(\mathbb{G})}, \frac{1}{\mathsf{maxord}(\mathbb{G})-\mathsf{minord}(\mathbb{G})} \in \mathsf{negl}(\lambda)$.

The additional assumption that we need to make is that it is hard to find any group element in $\mathbb{G}$ of low (poly-size) order. This is the Low Order Assumption [39]]. We refer to Section 3.3.1 for the definition of the assumption.

We note that specifically for RSA groups, for Low Order assumption to hold, we have to work in the quotient group $\mathbb{Z}_N^*/\{1,-1\}$ [215], since otherwise -1 would trivially break the assumption. So $\mathbb{Z}_N^*/\{1,-1\}$ would be an instantiation of a Hidden Order Group where the Low Order assumption holds.

In terms of constructions, one difference regards the upper bound on the order of $\mathbb{G}$ that is used in the protocols. More precisely, throughout the main core of our work we use N as an upper bound for the order of the group $\mathbb{Z}_N^*$ and $N/2$ as an upper bound for the order of the quadratic residues subgroup QR_N. Similarly, in a Hidden Order Group $\mathbb{G}$ generated by Ggen, although the order of the group is unknown, a range in which the order lies is known $[\mathsf{minord}(\mathbb{G}), \mathsf{maxord}(\mathbb{G})]$. So the maximum order $\mathsf{maxord}(\mathbb{G})$ can be used, instead of N, as an upper bound. In many cases these values are used either to securely sample a random value or to bound the size of a value needed for a security proof. For example a random value that is sampled from $\left(-\lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}, \lfloor N/4 \rfloor 2^{\lambda_z+\lambda_s}\right)$ in the RSA group instantiation will be sampled from $\left(-\frac{\mathsf{maxord}(\mathbb{G})}{2} 2^{\lambda_z+\lambda_s}, \frac{\mathsf{maxord}(\mathbb{G})}{2} 2^{\lambda_z+\lambda_s}\right)$ in the case of hidden order groups.

Here we give other specific changes that need to be made to instantiate our protocols in general hidden order groups. For IntCom, the verification equation becomes $C = G^x H^r$ (without the $\pm$). Then the argument of knowledge of opening of such a commitment would be

secure under the strong RSA and low order assumptions. The set commitments $\mathsf{SetCom}_{\mathsf{RSA}}$, $\mathsf{SetCom}_{\mathsf{RSA}'}$ remain the same and are binding under the strong RSA assumption for Ggen (and collision resistance of Hprime for the case of $\mathsf{SetCom}_{\mathsf{RSA}}$). For modEq, the same difference as for the AoK of an opening of an IntCom commitment is inherited. For Root and Coprime, the proposition 1 needs to be slightly modified: $A = B^{\frac{x}{y}}$ can be without $\pm$, and can be proven under the low order assumption instead. Finally, in the proof of security of protocol Coprime, in lemma 7 the assumption $\lambda_s < \log(N)/2$ is not needed as long as the low order assumption holds (an adversary that can find $H, \Delta c$ such that $\gcd(\mathsf{ord}(H), q^\ell) = 1$ can be used to break low order assumption).

5

ZERO-KNOWLEDGE PROOFS FOR BATCH SET MEMBERSHIP

The results of this chapter appear in a paper under the title "Succinct Zero-Knowledge Batch Proofs for RSA Accumulators" published at the ACM CCS 2022 conference [60].

5.1 Technical Contributions

We advance the research line of zero-knowledge proofs for set membership by proposing new techniques to efficiently use zkSNARKs with RSA accumulators. We propose new, more scalable protocols for succinct zero-knowledge proofs of batch membership.

Succinct proofs of batch membership. The main technical result of this chapter is a commit-and-prove [64] zkSNARK for batch membership, that is: Given an RSA accumulator acc to a set $S = \{x_1, \ldots, x_n\}$ and a succinct Pedersen commitment c_u to a vector of values $(u_1, \ldots, u_m)$, it holds $u_i \in S$ for every $i = 1, \ldots, m$. Thanks to the commit-and-prove feature, our scheme can be efficiently and modularly composed with other commit-and-prove[22] zkSNARKs [61] in order to prove further properties of the committed elements, e.g., $\forall i : u_i \in S \wedge P(u_1, \ldots, u_m) = \mathsf{true}$ (P could be for example a numerical range check). We dub our construction harisa[23].

Our technical contributions include: a new randomization method for RSA accumulator witnesses (needed to obtain zero-knowledge) and a new way to prove the accumulator verification in zero-knowledge in a SNARK *without* encoding RSA group operations in the constraint system. The latter is based on a novel combination of (non-succinct) sigma protocols, succinct proof of knowledge of exponent [40], and zkSNARKs for integer arithmetic.

5.2 Technical Overview

We now present a high-level overview of our techniques.

[22] Roughly, the verification algorithm of the zkSNARK takes as input short commitments to a long (potentially private) input. This property is useful as the elements for which we prove set membership need to stay private, but still "referred to", e.g. for proving additional properties on them.

[23] harisa stands for "elements-Hiding Argument for RSA accumulators".

Our core protocol is a succinct zero-knowledge proof of set membership for a batch of elements. Given a (public) set $S = \{x_1, \ldots, x_n\}$ and a commitment to $u_1, \ldots, u_m$, we aim at proving that $u_1, \ldots, u_m \in S$. We require for privacy that the u_i's remain hidden (and thus we provide them only as a commitment in the public input). We also require for efficiency that proof size and verification time should not depend either on the batch size m or the set size n.

We start from applying RSA accumulators [18] to compress the set into a succinct digest. Given random group element g in a group of unknown order (e.g. an RSA or class group [49]), one can produce a compressed representation of the set[24] as $\mathsf{acc} = g^{x_1 \cdot x_2 \cdots x_n}$. RSA accumulators enjoy succinct batch-membership proofs: to prove that $u_1, \ldots, u_m \in S$ it suffices to provide a single group element (a *witness*) $W = g^{\prod_{i\in[n]} x_i / \prod_{i\in[m]} u_j}$, which the verifier can check as $W^{u_1 \cdots u_m} = \mathsf{acc}$.

Though succinct, the batch-membership proofs of RSA accumulators do not hide the u_i elements, as the verifier should know them in order to perform the exponentiation. To address this problem, we could use a non-interactive zero-knowledge proof of exponentiation, which can be obtained using a Σ-protocol [83] (a three-message zero-knowledge scheme) made non-interactive through the Fiat-Shamir transform [101]. In it the prover computes: $R \leftarrow W^r$ for a sufficiently large random r; a random oracle challenge $h \leftarrow H(\mathsf{acc}||g||W||R)$, an integer $k \leftarrow r + h \cdot \prod_{i\in[m]} u_i$. The verifier accepts this zk-proof (R, h, k) if $h = H(\mathsf{acc}||g||W||R)$ and $R \cdot \mathsf{acc}^h = W^k$.

This protocol however does not yet achieve our goal, which is to generate a zero-knowledge batch-membership proof for committed u_i's. Towards this goal, we need to solve the following technical challenges. (A) The verifier needs to know the witness W, which can itself leak information about the elements it proves membership of. For example for $m = 1$ one can efficiently find the element u_1 by brute-force testing all elements of the set S, $W^{x_i} \stackrel{?}{=} \mathsf{acc}$ (recall that the set is public). The x_{i^*} for which the test passes will be u_1. (B) The proof (R, h, k) above simply shows existence of an exponent u such that $W^u = \mathsf{acc}$, in particular it does not link this statement to committed $(u_1, \ldots, u_m)$ such that $u = u_1 \cdots u_m$. (C) The proof is not succinct since the integer k is $O(m)$-bits long. (D) Most notably, the Σ-protocol described above is not even sound, unless the challenge is binary, $h \in \{0, 1\}$ [17, 200].

Our key contribution is an efficient technique to efficiently prove the verification of this Σ-protocol using a SNARK. Notably, we do not need encode any RSA group operations in the SNARK constraint system[25]. To obtain this result we combine three main ideas:

1. We introduce a novel randomization method for an RSA accumulator witness, $W \mapsto \hat{W}$, so that $\hat{W}$ provably doesn't leak any information about the u_i's.

   Our hiding-witness transformation works as follows: let $p_1, \ldots, p_{2\lambda}$ be the first 2λ prime numbers. We always (artificially) add these primes to the accumulator, i.e., the accumulator of a set S is an RSA accumulator $\hat{\mathsf{acc}}$ of $\hat{S} \leftarrow S \cup \{p_1, \ldots, p_{2\lambda}\}$: $\hat{\mathsf{acc}} \leftarrow \mathsf{acc}^{p_1 \cdots p_{2\lambda}}$ (we assume that S does not contain any of the p_i's). Then, to produce the hiding witness $\hat{W}$ we raise to the exponent each prime p_i with probability $1/2$. A bit more formally, we sample $b_i \leftarrow_\$ \{0, 1\}$ and set $\hat{W} \leftarrow W^{\prod_{i\in[2\lambda]} p_i^{1-b_i}}$, b_i's should remain hidden. We formally prove that under a cryptographic assumption (DDH-II, a variant of DDH [63]) $\hat{W}$ is computationally

[24] The elements of the set should be primes (or hashed to ones) for the RSA accumulator to securely apply.

[25] A "constraint system" is an encoding of the property proved by the SNARK. Its size, the number of constraints, is a key efficiency metric when evaluating proof schemes.

indistinguishable from random and thus $\hat{W}$, alone, hides u_i's (see section 5.4.1). Notice that $\hat{W}$ can be verified through the equality $\hat{W}^{\prod_{i\in[m]} u_i \cdot \prod_{i\in[2\lambda]} p_i^{b_i}} = \hat{\mathsf{acc}}$. Therefore we can use the NIZK for exponentiation described above, but for base $\hat{W}$ and exponent $e := \prod_{i\in[m]} u_i \cdot \prod_{i\in[2\lambda]} p_i^{b_i}$.

This technique solves the challenge (A) as it turns an RSA accumulator verification into a ZK verification. Yet challenges (B) and (C) remain: k is not short and the protocol only proves the existence of e such that $\hat{W}^e = \hat{\mathsf{acc}}$—which says nothing about membership of *legitimate* elements from S. For example e can contain only elements of $\{p_1, \ldots, p_{2\lambda}\}$ and no element from S.

2. To solve (B) we "link" the Σ-protocol to $c_{\boldsymbol{u}}$, a commitment to all u_i's, by using a zkSNARK that proves the correct computation of k from the committed legitimate u_i's. Namely it proves that, for $c_{\boldsymbol{u}}$, a commitment to $\boldsymbol{u}$, and $c_{r,s}$, a commitment to integers $s = \prod_{i\in[2\lambda]} p_i^{b_i}$ and r, the equality $k = r + h \cdot s \cdot \prod_{i\in[m]} u_i$ holds over the integers and $u_i > p_{2\lambda}$ for each $i \in [m]$. Recall that $p_{2\lambda}$ is the largest of all p_i's, so $u_i > p_{2\lambda}$ translates to $u_i \neq p_j$ for all $j \in [2\lambda]$. This means that the exponent of e contains elements u_i's committed a-priori and that they are *legitimate* (not one of the artificially added p_i's).

3. Although the above careful interplay between RSA Accumulators, Σ-protocols, zkSNARKs and our hiding technique for RSA accumulators witnesses gives a secure zero-knowledge proof of set membership, it is not yet succinct, as the verifier needs to receive the $O(m)$-long integer k. To solve this technical challenge, we apply a succinct proof of knowledge of exponent PoKE [40]. Instead of sending k, the prover sends $B = \hat{W}^k$ accompanied with a succinct proof that there is an integer k such that $B = \hat{W}^k$. Adding the PoKE proof however breaks the link between the Σ-protocol and the zkSNARK as the latter is supposed to generate a proof for a public k. To solve this last challenge, we "open the box" of PoKE verification and observe that the verifier receives the short integer $\hat{k} = k \mod \ell$, where ℓ is a random prime challenge of 2λ bits. Therefore, the last idea of our protocol is to let the zkSNARK prove the same statement as above but for $\hat{k}$, namely that $\hat{k} = r + h \cdot s \cdot \prod_{i\in[m]} u_i \mod \ell$.

A special mention needs to be made to (D), the soundness of the Σ-protocol. Standard impossibility results [17, 200] show that the Σ-protocols over groups of unknown order (as the groups of RSA accumulators) can have at most $1/2$ soundness-error, meaning that they need many repetitions (e.g. $\lambda = 128$) to leverage them to fully sound (with negligible soundness-error). This usually makes the protocols prohibitively expensive.

The general intuition of the impossibility is that (using usual rewinding techniques) the extractor gets (R, h, k) and (R, h', k') such that $\mathsf{acc}^{h-h'} = W^{k-k'}$. However, we cannot imply to $acc = W^{(k-k')/(h-h')}$ because $(h - h')^{-1}$-in the exponent cannot be efficiently computed in groups of unknown order. So we are bound to set $h \in \{0, 1\}$ (so that $h - h' = 1$). In our solution, the zkSNARK proof described in (2) makes the extraction of the Sigma-protocol possible. This is possible because this proof guarantees that, in the two executions, $k = r + suh$ and $k' = r + suh'$, for committed r, s, u. This way, we get that $\mathsf{acc}^{h-h'} = W^{su(h-h')}$, from which we can conclude the desired result $\mathsf{acc} = W^{su}$.

Our technique of using a zkSNARK for the correct computation of the last message of a Σ-protocol over groups of unknown order, is generic for *any* such protocol and gives a way to efficiently bypass the impossibility results [17, 200] without inexpensive repetitions. We expect this to be of independent interest.

5.3 Definitions and Building Blocks

We give informal definitions for the main cryptographic primitives used in our constructions.

5.3.1 Relations for batch set-membership

Remark 13 (Syntactic Sugar for SNARKs/CP-SNARKs)**.** *For convenience we will use the following notational shortcuts. We make explicit what the private input of the prover is by adding semicolon in a relation and in a prover's algorithm (e.g., $R(\mathbb{x};\mathbb{w})$). We explicitly mark relations as "commit-and-prove" by a tilde. We leave the assumed commitment scheme implicit when it's obvious from the context. Occasionally, we will also explicitly mark the commitment inputs by squared box around them (e.g. $\boxed{c_u}$) and we will assume implicitly that the relation includes checking the opening of these commitments (and we will not make explicit the openings). We assume that in the commitment $\boxed{c_u}$ the subscript u defines the variable u the commitment opens to. Analogously the opening for c_u is automatically defined as o_u. Example: $\tilde{R}_{\mathsf{ck}}(\boxed{c_u}, h; r) = 1 \Leftrightarrow h = \mathsf{SHA256}(u||r)$ is a shortcut for $\tilde{R}_{\mathsf{ck}}(c_u, h; r, u, o_u) = 1 \Leftrightarrow h = \mathsf{SHA256}(u||r) \wedge c_u = \mathsf{Commit}(\mathsf{ck}, u; o_u)$.*

Our focus in this work is on building efficient CP-SNARKs for the following relation parametrized by an accumulator scheme Acc and parameters $\mathsf{pp}_{\mathsf{Acc}}$:

$$\tilde{R}^{\mathrm{mem}}_{\mathsf{ck}}(\boxed{c_U}, \mathsf{acc}; W) = 1 \Leftrightarrow \mathsf{Acc.VfyMem}(\mathsf{pp}, \mathsf{acc}, U, W) = 1$$

In a nutshell, a CP-SNARK for $\tilde{R}^{\mathrm{mem}}_{\mathsf{ck}}$ can prove that c_U is a commitment to a vector of values such that each of them is in the set accumulated in acc.

The specific notion of knowledge soundness we assume for CP-SNARKs for these relations is the one where the malicious prover is allowed to select an arbitrary set S to be accumulated but the accumulator acc is computed honestly from S. Given an accumulator scheme Acc, we informally talk about this specific notion as "security under the Trusted Accumulator-Model for Acc". We do not provide formal details since this model corresponds to the notion of partial-extractable soundness (see Section 4.2.3). This trusted accumulator model fits several applications where the accumulator is maintained by the network.

In the next section we recall an interesting byproducts of having modular commit-and-prove SNARKs for the relations $\tilde{R}^{\mathrm{mem}}_{\mathsf{ck}}$.

5.3.2 Composing (commit-and-prove) set-membership relations

The advantage of having CP-SNARKs for the set-membership relation (rather than just SNARKs) is that one can use the composition of section 3.7.3.3 to obtain *efficient* zkSNARKs for proving properties of elements in an accumulated set, e.g., to show that $\exists U = \{u_1, \ldots, u_n\}$ such that a property P holds for U (say, every u_i is properly signed) and $U \subset S$, where S is accumulated in some acc. In particular, such a zkSNARK can be obtained via the simple and efficient composition of a CP-SNARK for $\tilde{R}^{\mathrm{mem}}_{\mathsf{ck}}$ (like the ones we construct in our work) and any other CP-SNARK for P.

5.4 harisa: Zero-Knowledge CP-SNARK for Batch Set-Membership

In this section we show the construction of a CP-SNARK for the relation $\tilde{R}^{\text{mem}}_{\mathsf{ck}}$ defined in Section 5.3.2, where: the accumulator is the classical RSA accumulator (see Section 3.5.2, Figure 3.2) where the accumulated elements are prime numbers larger than the 2λ-th prime (1619 for $\lambda = 128$), and the commitment scheme for the commit-and-prove functionality is the Pedersen scheme (see Figure 3.1). In section 5.6.1 we discuss how this construction can be easily extended to accumulate arbitrary elements via an efficient hash-to-prime function.

5.4.1 RSA Accumulators with hiding witnesses

We describe a method to turn a witness W of an RSA accumulator into another witness that computationally hides all the elements u_i it proves membership of. As discussed in Section 5.2 this constitutes the first building block towards achieving a zero-knowledge membership proof for committed elements.

Let $\mathbb{P}_n = \{2, 3, 5, 7, \ldots, p_n\}$ be the set of the first n prime numbers. Our method relies on two main ideas.

First, prover and verifier modify the accumulator acc so as to contain the first 2λ primes by computing $\hat{\mathsf{acc}} \leftarrow \mathsf{acc}^{\left(\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i\right)}$. Note, $\hat{\mathsf{acc}} = g_?^{\left(\prod_{x_i \in S} x_i\right) \cdot \left(\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i\right)} = \mathsf{Accum}(\mathsf{pp}, S \cup \mathbb{P}_{2\lambda})$.

Second, we build a randomized witness for $X \subset S$ as the witness for $(X \cup P) \subset (S \cup \mathbb{P}_{2\lambda})$ where P is a randomly chosen subset of $\mathbb{P}_{2\lambda}$. In more detail, given W, the prover computes $\hat{W}$ as follows:

- choose at random 2λ bits $b_1, \ldots, b_{2\lambda} \leftarrow\!\!\$\ \{0,1\}$ and let $s := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{b_i}$ and $\bar{s} := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-b_i}$.
- $\hat{W} \leftarrow W^{\bar{s}} = g_?^{\left(\prod_{x_i \in S \setminus X} x_i\right) \cdot \left(\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-b_i}\right)}$.

Essentially, we have s as the product of the randomly chosen primes, $\bar{s}$ as the product of the primes not chosen, and we denote with $p^* := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i$ the product of all the first 2λ primes. Finally, by $\mathcal{D}_{2\lambda}$ we denote the distribution of $\bar{s}$, according to the sampling method described above. Note that $s\bar{s} = p^*$. Also, the new witness $\hat{W}$ could be verified by checking $\hat{W}^s \prod_{x_i \in X} x_i = \hat{\mathsf{acc}}$.

Our first technical contribution is proving that this randomization is sufficient. More precisely, we use a computational assumption over groups of unknown order, called DDH-II, and we show that under DDH-II $\hat{W}$ is computationally indistinguishable from a random $R \leftarrow\!\!\$\ \mathbb{G}_?$. We stress that this hiding property holds only for the value $\hat{W}$ *alone*, i.e., when the random subset of $\mathbb{P}_{2\lambda}$ is not revealed. As we show later, this is sufficient for our purpose as we can hide the integer s in the same way as we hide the elements we prove membership of.

In the following section we state and explain the DDH-II assumption. In brief, this is a variant of the classical DDH assumption where the random exponents follow specific, not uniform, distributions. Next, we prove that under DDH-II $\hat{W}$ is computationally indistinguishable from random.

5.4.1.1 The DDH-II assumption

First, we state the DDH-II assumption, which is parametrized by a generator $\mathsf{GGen}_?(1^\lambda)$ of a group (of unknown order in our case) and by a well-spread distribution $\mathcal{WS}_{2\lambda}$ (in our case $\mathcal{D}_{2\lambda}$).

A distribution $\mathcal{WS}_{2\lambda}$ with domain $\mathcal{X}_{2\lambda}$ is called *well-spread* if $\Pr[X = x | X \leftarrow\!\!\$\, \mathcal{WS}_{2\lambda}] \leq 2^{-2\lambda}$ for each $x \in \mathcal{X}_{2\lambda}$ (Intuition: the elements sampled from this distribution are "sufficiently random").

Assumption 1 (DDH-II)**.** *Let $\mathbb{G}_? \leftarrow \mathsf{GGen}_?(1^\lambda)$ and $g_? \leftarrow\!\!\$\, \mathbb{G}_?$. Let $\mathcal{WS}_{2\lambda}$ be a well-spread distribution with domain $\mathcal{X}_{2\lambda} \subseteq [1, \mathsf{minord}(\mathbb{G}_?)]$. Then for any PPT $\mathcal{A}$:*

$$\left|\Pr[\mathcal{A}(g_?^x, g_?^y, g_?^{xy}) = 0] - \Pr[\mathcal{A}(g_?^x, g_?^y, g_?^t) = 0]\right| = \mathsf{negl}$$

where $x \leftarrow\!\!\$\, \mathcal{WS}_{2\lambda}$ and $y, t \leftarrow\!\!\$\, [1, \mathsf{maxord}(\mathbb{G}_?)2^\lambda]$.[26]

Our distribution of interest $\mathcal{D}_{2\lambda}$ can be shown well-spread: there are $2^{2\lambda}$ outcomes and are all distinct, $\bar{s} = \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-bi}$ are distinct since they are different products of the same primes (no p_i can be used twice). It follows that $\Pr[\bar{s} \leftarrow\!\!\$\, \mathcal{D}_\lambda] = 1/2^{2\lambda}$ for every $\bar{s}$.

Remark 14. *The constraint that the domain should be in $[1, \mathsf{minord}(\mathbb{G}_?)]$ is for the following reason: If a sampled x is larger than $\mathsf{ord}(g_?)$ then in the exponent of $g_?^x$ a reduction modulo $\mathsf{ord}(g_?)$ will implicitly happen leading to a $g_?^x = g_?^{x'}$ for some $x' \neq x$. This can turn $g_?^x$ more frequently sampled, which can potentially help the adversary distinguish between $(g_?^x)^y$ and $g_?^t$.*

Different variants of DDH-II have been proven secure in the generic group model [196, 160] for prime order groups [86, 19]. We can prove it secure for groups of unknown order similarly with minor technical modifications related to GGM proofs in such groups [87]. In Section 5.7 we present a proofs of security in the GGM.

Remark 15. *The need of an at least $2^{2\lambda}$-large domain $\mathcal{X}_{2\lambda}$ (and at most $2^{-2\lambda}$ probability) for λ security parameter comes from well-known subexponential attacks on DLOG [179, 181].*

5.4.1.2 Security Proof of our hiding witnesses

Theorem 16. *For any parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, set S (where $S \cap \mathbb{P}_{2\lambda} = \emptyset$), $R \leftarrow\!\!\$\, \mathbb{G}_?$ and $\hat{W}$ computed as described above it holds:*

$$\left|\Pr[\mathcal{A}(\mathsf{pp}, S, \hat{W}) = 0] - \Pr[\mathcal{A}(\mathsf{pp}, S, R) = 0]\right| = \mathsf{negl}$$

for any PPT $\mathcal{A}$, under the DDH-II assumption for $\mathbb{G}_?$ and $\mathcal{D}_{2\lambda}$.

Proof. Call $\mathcal{A}$ an adversary achieving a non-negligible advantage ϵ above, i.e. $\epsilon := \left|\Pr[\mathcal{A}(\mathsf{pp}, S, \hat{W}) = 0] - \Pr[\mathcal{A}(\mathsf{pp}, S, R) = 0]\right|$ We construct an adversary $\mathcal{B}$ against DDH-II that, using adversary $\mathcal{A}$, gains the same advantage. $\mathcal{B}$ receives $(\mathbb{G}_?, g_?, g_?^{\bar{s}}, g_?^r, g_?^{b\bar{s}r+(1-b)t})$, where $\bar{s} \leftarrow\!\!\$\, \mathcal{D}_{2\lambda}$ and $r, t \leftarrow\!\!\$\, [1, \mathsf{maxord}(\mathbb{G}_?)2^\lambda]$. Then it chooses arbitrarily an element u and sets $S = \{u\}$, $\mathsf{pp} \leftarrow (\mathbb{G}_?, g_?^r)$ and $V = g_?^{b\bar{s}r+(1-b)t}$. $\mathcal{B}$ sends (pp, S, V) to the adversary $\mathcal{A}$, who outputs a bit b^*. Finally, $\mathcal{B}$ outputs b^*.

First, notice that $g_?^r$ is statistically close to a random group element of $\mathbb{G}_?$, meaning that $\mathcal{A}$ cannot distinguish pp from parameters generated by $\mathsf{Acc.Setup}(1^\lambda)$. Furthermore if $b = 0$ then V is again a (statistically indistinguishable element from a) uniformly random group element of $\mathbb{G}_?$ therefore $\Pr[\mathcal{B} = 0 | b = 0] = \Pr[\mathcal{A}(\mathsf{pp}, S, R) = 0]$. On the other hand, if $b = 1$ then $V = g_?^{r \cdot \bar{s}} = \hat{W}_u$ is a witness of u so $\Pr[\mathcal{B} = 0 | b = 1] = \Pr[\mathcal{A}(\mathsf{pp}, S, \hat{W}) = 0]$. Therefore we conclude that the probability of $\mathcal{B}$ to win the DDH-II is ϵ. □

[26] Since the order of the group is unknown, we cannot efficiently produce uniformly random elements with $y, t \leftarrow\!\!\$\, [1, \mathsf{ord}(g_?)]$. However, $y, t \leftarrow\!\!\$\, [1, \mathsf{maxord}(\mathbb{G})2^\lambda]$ still produces statistically close to uniform elements.

5.4.2 Building Blocks

5.4.2.1 Succinct proofs of knowledge of exponent (PoKE)

We recall the succinct proofs of knowledge of a DLOG for hidden order groups, introduced by Boneh et al. [40]. More formally, PoKE is a protocol for the relation

$$R^{\mathsf{PoKE}}(A, B; x) = 1 \Leftrightarrow A^x = B$$

parametrized by a group of unknown order $\mathbb{G}_?$ and a random group element $g_? \in \mathbb{G}_?$. The statement consists of group elements $A, B \in \mathbb{G}_?$ while the witness is an arbitrarily large $x \in \mathbb{Z}$.

Figure 7.6 gives a description of the protocol. For simplicity we directly expose its non-interactive version (after Fiat-Shamir). Although the interactive version of the protocol is secure with λ-sized challenges its non-interactive version is only secure with 2λ-sized challenges, due to a subexponential attack [39].

$\underline{\mathsf{Setup}(1^\lambda):}$
- $(\mathbb{G}_?, g_?) \leftarrow \mathsf{GGen}_?(1^\lambda)$
- **return** $\mathsf{crs} := (\mathbb{G}_?, g_?)$

$\underline{\mathsf{Prove}(\mathsf{crs}, A, B; x):}$
- $\ell \leftarrow \mathsf{Hprime}(\mathsf{crs}, A, B)$
- $Q \leftarrow A^{\lfloor \frac{x}{\ell} \rfloor}, res \leftarrow x \mod \ell$
- **return** $\pi = (Q, res)$

$\underline{\mathsf{VerProof}(\mathsf{crs}, A, B, \pi):}$
- Parse π as (Q, res)
- $\ell \leftarrow \mathsf{Hprime}(\mathsf{crs}, A, B)$
- Reject if $A, B, Q \notin \mathbb{G}_?$ or $res \notin [0, \ell - 1]$
- Reject if $Q^\ell A^{res} \neq B$

Figure 5.1: The succinct argument of knowledge PoKE [40]. Hprime denotes a cryptographic hash function that outputs a prime of size 2λ, modeled as a random oracle.

Remark 16. *We note that the proof of fig. 7.6 is not originally secure for arbitrary bases A, but rather for random ones. For arbitrary bases extra care should be taken, that give a proof of additional 2 group elements. We wil show that the protocol still suffices for our needs, since we combine it with a SNARK for the relation $res = x \mod \ell$. In a nutshell, a PoKE for random bases with a SNARK for $res = x \mod \ell$ give a succinct proof of knowledge of exponent for arbitrary bases.*

This protocol is succinct: proof size and verifier's work are independent of the size of x, $O(\lambda)$ and $O(\|\ell\|) = O(\lambda)$ respectively.

5.4.2.2 CP-SNARK for integer arithmetic relations

We assume an efficient CP-SNARK $\mathsf{cp}\Pi^{\mathrm{modarithm}}$ for the following relation:

$$\tilde{R}^{\mathrm{modarithm}}_{\mathsf{ck}}(\boxed{c_{\boldsymbol{u}}}, \boxed{c_{s,r}}, h, \ell, \hat{k}) = 1 \Leftrightarrow \hat{k} = s \cdot h \cdot \prod_{i \in [m]} u_i + r \mod \ell$$

Above, $\boldsymbol{u} = (u_1, \ldots, u_m) \in \mathbb{Z}^m$ is a vector of integers with a corresponding multi-integer commitment $c_{\boldsymbol{u}}$; $r, s \in \mathbb{Z}$ are integers committed with a corresponding multi-integer commitment $c_{s,r}$ and $\ell, h \in \mathbb{Z}$, $\hat{k} \in [0, \ell - 1]$ are (small) integers known as public inputs by both prover and verifier.

The above relation is equivalent to the integer relation:

$$\tilde{R}^{\text{arithm}}_{\mathsf{ck}}(\boxed{c_{\boldsymbol{u}}}, \boxed{c_{s,r}}, h, \ell, \hat{k}; q) = 1 \Leftrightarrow q\ell + \hat{k} = s \cdot h \prod_i u_i + r$$

In fact this is how a modulo operation is encoded in a SNARK circuit. q here is a witness given to the SNARK.[27]

5.4.2.3 CP-SNARK for inequalities

We need a CP-SNARK $\mathsf{cp\Pi}^{\text{bound}}$ for the relation (where B is a public integer):

$$\tilde{R}^{\text{bound}}_{\mathsf{ck}}(\boxed{c_{\boldsymbol{u}}}, B) = 1 \Leftrightarrow \bigwedge_{i \in [n]} u_i > B$$

5.4.3 Our Construction for Batched Set Membership (harisa)

Here we describe our CP-SNARK for the relation $\tilde{R}^{\text{mem}}_{\mathsf{ck}}$ for RSA accumulators and Pedersen commitments to vectors of integers. Let us recall the setting in more detail.

Prover and verifier hold an accumulator acc to a set S and a commitment $c_{\boldsymbol{u}}$. The set's domain are prime numbers greater than $p_{2\lambda}$, the 2λ-th prime. The protocol works in the "trusted accumulator model" (section 5.3.1), which means the set is assumed to be public but the verifier does not take it as an input, it only uses acc, for efficiency reasons.[28]

The prover knows a batch of set elements $\boldsymbol{u} = (u_1, \ldots, u_m)$ that are an opening of the commitment $c_{\boldsymbol{u}}$, and its goal is to convince the verifier that all the u_i's are in S. To this end, we assume that the prover has an accumulator witness $W_{\boldsymbol{u}}$ as an input, either precomputed or given by a witness-providing entity. In this sense, the prover's goal translates into convincing the verifier that it has $W_{\boldsymbol{u}}$ such that $W_{\boldsymbol{u}}^{\prod_i u_i} = \mathsf{acc}$ (see also section 5.5 where we further refine this setting).

We give a full description of the CP-SNARK in Figure 5.2. We refer to the technical overview (sec. 5.2) for a high-level explanation. Below we provide additional comments.

To begin with, both prover and verifier transform the accumulator acc into $\hat{\mathsf{acc}}$, the one corresponding to the same set with the additional small prime numbers from $\mathbb{P}_{2\lambda}$.[29] Next, the prover transforms W into a hiding witness as $\hat{W} = W^{\bar{s}}$ via our masking method of section 5.4.1, and then computes a (Fiat-Shamir-transformed) zero-knowledge Σ-protocol for the accumulator's verification $\hat{W}^{su^*} = \hat{\mathsf{acc}}$. However, since the last message k of the protocol is not succinct, it computes a PoKE for the relation $(\hat{\mathsf{acc}}^h R) = (\hat{W}_{\boldsymbol{u}})^k$ (exponent k), which is the verification equation of the Σ-protocol. The PoKE verification requires a check $Q^\ell \hat{W}_{\boldsymbol{u}}^{\hat{k}}$ where $\hat{k}$ is supposed

[27] For the sake of our general protocol, it is not necessary that q remains hidden. It is only important that the proof is succinct w.r.t. its size. However, $\boldsymbol{u}$, s and r should remain hidden.

[28] This is a common consideration in scalable systems. The accumulator to the set is either computed once by the verifier or validated by an incentivized majority of parties that is supposed to maintain it.

[29] This operation can also be precomputed, we make it explicit only to show that they can both work with a classical RSA accumulator as an input.

to be $k \mod \ell$. The last step of the proof is to show that $\hat{k}$ is not just "some exponent" but it is exactly $r + hsu^* \mod \ell$ with u^* being the product of all the u_i's committed in $c_{\boldsymbol{u}}$. To do so, the prover generates a proof with the $\mathsf{cp}\Pi^{\text{arithm}}$ CP-SNARK over the commitments $c_{\boldsymbol{u}}$, $c_{s,r}$ (r is the masking randomness of the Σ-protocol sampled in the first move). Also, for soundness we require that s and r are committed *before* receiving the random oracle challenge h. Finally, the prover generates a proof with $\mathsf{cp}\Pi^{\text{bound}}$ over the commitment $c_{\boldsymbol{u}}$ to ensure that the elements are in the right domain.[30]

We present our construction in fig. 5.2. This construction is obtained by applying Fiat-Shamir in the random oracle model (ROM) and additional optimizations to its interactive counterpart which we describe in the appendix (fig. 5.3).

5.4.4 Security Proof

In fig. 5.3 we describe an interactive version of our construction and prove its security below.

Theorem 17. *Let H, Hprime be modeled as random oracles and $\mathsf{cp}\Pi^{modarithm}$, $\mathsf{cp}\Pi^{bound}$ be secure CP-SNARKs. The construction in fig. 5.2 for the relation $\tilde{R}^{mem}_{\mathsf{ck}}$ is a secure CP-SNARK: succinct, knowledge-sound under the adaptive root assumption, and zero-knowledge under the DDH-II assumption.*

Proof. For succinctness, one can inspect that the proof size is proportional to that of $\mathsf{cp}\Pi^{\text{arithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$ plus some small constant overhead. Similarly for the verifier's cost. So succinctness is inherited from succinctness of $\mathsf{cp}\Pi^{\text{arithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$.

The proof for its interactive version (fig. 5.3) is in the appendix, theorem 18. Then knowledge-soundness and zero-knowledge come directly from the (tight) security of the Fiat-Shamir transformation for constant-round protocols [12], in the random oracle model. □

Theorem 18. *Let $\mathsf{cp}\Pi^{modarithm}$, $\mathsf{cp}\Pi^{bound}$ be secure CP-SNARKs then the construction in fig. 5.3 for the relation $\tilde{R}^{mem}_{\mathsf{ck}}$ is a secure CP-NIZK: succinct, knowledge-sound under the adaptive root assumption and zero-knowledge under the DDH-II assumption.*

Proof. Succinctness: Comes from inspection and from the assumption that $\mathsf{cp}\Pi^{\text{modarithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$ are succinct.

$(2, M)$-*Special Soundness*: assume that we have a tree of $(2, M)$ successful transcripts, for $M = \mathsf{poly} > \left\lceil \frac{\|p^*\|+\|u^*\|+\lambda}{2\lambda} \right\rceil$, i.e.

$$\left\{ \left(\hat{W}_{\boldsymbol{u}}, c_{s,r}, R \right), h, \ell^{(j)}, \left(Q^{(i)}, res^{(j)} \right), \pi_2^{(j)}, \pi_3^{(j)} \right\}_{j=1}^{M}$$

and

$$\left\{ \left(\hat{W}_{\boldsymbol{u}}, c_{s,r}, R \right), \tilde{h}, \tilde{\ell}'^{(j)}, \left(\tilde{Q}^{(i)}, \tilde{res}^{(j)} \right), \tilde{\pi}_2^{(j)}, \tilde{\pi}_3^{(j)} \right\}_{j=1}^{M}$$

We construct an extractor Ext that works as follows.

Ext uses the extractor of $\mathsf{cp}\Pi^{\text{modarithm}}$ to extract $\boldsymbol{u}^{(j)}, s^{(j)}, r^{(j)}$, openings of $c_{\boldsymbol{u}}$ and $c_{s,r}$ respectively, such that $res^{(j)} = s^{(j)} h \prod_i u_i^{(j)} + r^{(j)} \mod \ell^{(j)}$. From the binding of the commitments we get that $\boldsymbol{u}^{(j)} = \boldsymbol{u}^{(j')}, s^{(j)} = s^{(j')}, r^{(j)} = r^{(j')}$ for each transcript $j \neq j'$ and

[30] For the sake of generality we present π_2, π_3 as distinct proofs. In practice they can be proved by the same CP-SNARK and save on proof-size.

$\underline{\mathsf{Setup}\left(1^\lambda, \mathsf{ck}, \mathsf{pp}\right):}$

$\mathsf{crs}_2 \leftarrow \mathsf{cp}\Pi^{\text{modarithm}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\text{modarithm}}_{\mathsf{ck}})$

$\mathsf{crs}_3 \leftarrow \mathsf{cp}\Pi^{\text{bound}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\text{bound}}_{\mathsf{ck}})$

$\textbf{return } \mathsf{crs} := (\mathsf{ck}, \mathsf{pp}, \mathsf{crs}_2, \mathsf{crs}_3)$

$\underline{\mathsf{Prove}\left(\mathsf{crs}, \mathsf{acc}, c_{\boldsymbol{u}}; W_{\boldsymbol{u}}, \boldsymbol{u}, o_{\boldsymbol{u}}\right):}$

$\hat{\mathsf{acc}} \leftarrow \mathsf{acc}^{\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i}$

Let $u^* = \prod_i u_i,\ p^* = \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i$

Sample $b_1, \ldots, b_{2\lambda} \leftarrow\!\!\$\ \{0,1\}$

Let $s := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{b_i},\ \bar{s} := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-b_i}$

$\hat{W}_{\boldsymbol{u}} \leftarrow W_{\boldsymbol{u}}^{\bar{s}}$

Sample $r \leftarrow\!\!\$\ \{0,1\}^{\|p^*\|+\|u^*\|+2\lambda}$

$c_{s,r} \leftarrow \mathsf{Commit}_{\mathsf{ck}}(s, r; o_{s,r})$

$R \leftarrow \hat{W}_{\boldsymbol{u}}^{r}$

$h \leftarrow H(\mathsf{crs}||\mathsf{acc}||c_{\boldsymbol{u}}||c_{s,r}||\hat{W}_{\boldsymbol{u}}||R)$

$k \leftarrow r + (u^* s)h$

$\pi_1 \leftarrow \Pi^{\mathsf{PoKE}}.\mathsf{Prv}\big((\mathbb{G}_?, g_?), \hat{W}_{\boldsymbol{u}}, \hat{\mathsf{acc}}^h R; k\big)$

Parse π_1 as $(Q, \hat{k})$

$\ell \leftarrow \mathsf{Hprime}((\mathbb{G}_?, g_?), \hat{W}_{\boldsymbol{u}}, \hat{\mathsf{acc}}^h R)$

$\pi_2 \leftarrow \mathsf{cp}\Pi^{\text{modarithm}}.\mathsf{Prv}(\mathsf{crs}_2, c_{\boldsymbol{u}}, c_{s,r}, h, \ell, \hat{k}; \boldsymbol{u}, o_{\boldsymbol{u}}, r, s, o_{s,r})$

$\pi_3 \leftarrow \mathsf{cp}\Pi^{\text{bound}}.\mathsf{Prv}(\mathsf{crs}_3, c_{\boldsymbol{u}}, p_{2\lambda}; \boldsymbol{u}, o_{\boldsymbol{u}})$

$\textbf{return } \pi = \big(\hat{W}_{\boldsymbol{u}}, R, c_{s,r}, \pi_1, \pi_2, \pi_3\big)$

$\underline{\mathsf{VerProof}\left(\mathsf{crs}, \mathsf{acc}, c_{\boldsymbol{u}}, \pi\right):}$

$\hat{\mathsf{acc}} \leftarrow \mathsf{acc}^{\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i}$

Parse π as $(\hat{W}_{\boldsymbol{u}}, R, c_{s,r}, \pi_1, \pi_2, \pi_3)$ and π_1 as $(Q, \hat{k})$

$\ell \leftarrow \mathsf{Hprime}((\mathbb{G}_?, g_?), \hat{W}_{\boldsymbol{u}}, \hat{\mathsf{acc}}^h R)$

$h \leftarrow H(\mathsf{crs}||\mathsf{acc}||c_{\boldsymbol{u}}||c_{s,r}||\hat{W}_{\boldsymbol{u}}||R)$

Reject if $\Pi^{\mathsf{PoKE}}.\mathsf{Vfy}((\mathbb{G}_?, g_?), \hat{W}_{\boldsymbol{u}}, \hat{\mathsf{acc}}^h R, \pi_1) \neq 1$

Reject if $\mathsf{cp}\Pi^{\text{modarithm}}.\mathsf{Vfy}(\mathsf{crs}_2, c_{\boldsymbol{u}}, c_{s,r}, h, \ell, \hat{k}, \pi_2) \neq 1$

Reject if $\mathsf{cp}\Pi^{\text{bound}}.\mathsf{Vfy}(\mathsf{crs}_3, c_{\boldsymbol{u}}, p_{2\lambda}, \pi_3) \neq 1$

Figure 5.2: harisa: our scheme for proving set membership of a committed element. We let H denote a cryptographic hash function modeled as a random oracle.

$j, j' \in [M]$, since they refer to the same commitments. So we denote the extracted values as $\boldsymbol{u}, s, r$ and get:

$$sh \prod_i u_i + r = res^{(j)} \mod \ell^{(j)}, \text{ for each } j \in [M]$$

$\underline{\mathsf{Setup}\left(1^\lambda, \mathsf{ck}, \mathsf{pp}_{\mathsf{Acc}}\right):}$

$\mathsf{crs}_2 \leftarrow \mathsf{cp}\Pi^{\text{arithm}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\text{arithm}}_{\mathsf{ck}})$

$\mathsf{crs}_3 \leftarrow \mathsf{cp}\Pi^{\text{bound}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, R^{\text{bound}})$

$\mathbf{return}\ \mathsf{crs} := (\mathsf{ck}, \mathsf{pp}_{\mathsf{Acc}}, \mathsf{crs}_2, \mathsf{crs}_3)$

$\underline{P\left(\mathsf{crs}, \hat{\mathsf{acc}}, c_{\boldsymbol{u}}; W_{\boldsymbol{u}}, \boldsymbol{u}, o_{\boldsymbol{u}}, S\right)}$		$\underline{V(\mathsf{crs}, \hat{\mathsf{acc}}, c_{\boldsymbol{u}})}$
Sample $b_1, \ldots, b_{2\lambda} \leftarrow_{\$} \{0,1\}$		
let $s := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{b_i},\ \bar{s} := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-b_i}$		
$\hat{W}_{\boldsymbol{u}} \leftarrow W_{\boldsymbol{u}}^{\bar{s}}$		
Sample $r \leftarrow_{\$} \{0,1\}^{\|p^*\|+\|u^*\|+\lambda}$		
$c_{s,r} \leftarrow \mathsf{Commit}_{\mathsf{ck}}(s, r; o_{s,r})$		
$R \leftarrow \hat{W}_{\boldsymbol{u}}^r$		
	$\xrightarrow{\pi_3, \hat{W}_{\boldsymbol{u}}, c_{s,r}, R}$	
		$h \leftarrow_{\$} \{0,1\}^\lambda$
	$\xleftarrow{h}$	
$k \leftarrow r + (s \prod_i u_i)h$		
		$\ell \leftarrow_{\$} \mathbb{P}_{2^{2\lambda}}$
	$\xleftarrow{\ell}$	
$Q \leftarrow A^{\lfloor \frac{k}{\ell} \rfloor}, res \leftarrow k \mod \ell$		
$\pi_2 \leftarrow \mathsf{cp}\Pi^{\text{modarithm}}.\mathsf{Prv}(\mathsf{crs}_2, \boxed{c_{\boldsymbol{u}}}, \boxed{c_{s,r}}, h, \ell, res)$		
$\pi_3 \leftarrow \mathsf{cp}\Pi^{\text{bound}}.\mathsf{Prv}(\mathsf{crs}_3, c_{\boldsymbol{u}}, p_{2\lambda}; \boldsymbol{u}, o_{\boldsymbol{u}})$		
	$\xrightarrow{(Q, res), \pi_2}$	

Accept iff:

1. $Q \in \mathbb{G}_?$ and $res \in [0, \ell - 1]$
2. $Q^\ell \hat{W}_{\boldsymbol{u}}^{res} = \hat{acc}^h R$ and
3. $\mathsf{cp}\Pi^{\text{modarithm}}.\mathsf{Vfy}(\mathsf{crs}_2, c_{\boldsymbol{u}}, c_{s,r}, h, \ell, res, \pi_2) = 1$
4. $\mathsf{cp}\Pi^{\text{bound}}.\mathsf{Vfy}(\mathsf{crs}_{\text{bound}}, c_{\boldsymbol{u}}, p_{2\lambda}, \pi_3) = 1$

Figure 5.3: Interactive version of our protocol for batch membership.

Using the Chinese Remainder Theorem we get a k such that

$$k = sh\prod_i u_i + r \mod \left(\prod_{j=1}^{M} \ell^{(j)}\right)$$

M can be set sufficiently large (but still polynomial-sized) so that $\prod_{j=1}^{M} \ell^{(j)} > sh\prod_i u_i + r$ and thus $k = sh\prod_i u_i + r$ over the integers. Furthermore, $k = res^{(j)} \mod \ell^{(j)}$ for each $j \in [M]$.

As shown in [40] the fact that for any accepting proof, (ℓ, Q, res), it holds that $Q^{\ell}\hat{W}_{\boldsymbol{u}}^{res} = \mathsf{acc}^h R$ and $k = res \mod \ell$ (the latter in our case is ensured by the SNARK) then under the adaptive root assumption we get:

$$\hat{W}_{\boldsymbol{u}}^{k} = \hat{\mathsf{acc}}^h R$$

(we refer to [40] appendix C.2 for the formal reduction).

Then the extractor does the same for the second set of transcripts to get $\tilde{k}, \tilde{\boldsymbol{u}}, \tilde{s}, \tilde{r}$ such that $\hat{W}_{\tilde{\boldsymbol{u}}}^{\tilde{k}} = \hat{\mathsf{acc}}^{\tilde{h}} R$ and $\tilde{k} = \tilde{s}\tilde{h}\prod_i \tilde{u}_i + \tilde{r}$ over the integers. Now since $\tilde{\boldsymbol{u}}, \tilde{s}, \tilde{r}$ refer to the same commitment as $\boldsymbol{u}, s, r$ (recall that the commitment were sent a priori) from the binding of the pedersen commitment we get that $\tilde{\boldsymbol{u}} = \boldsymbol{u}, \tilde{s} = s, \tilde{r} = r$, which gives us that $\tilde{k} = s\tilde{h}\prod_i u_i + r$.

From the above we have: $\hat{W}_{\boldsymbol{u}}^{k} = \hat{\mathsf{acc}}^h R$ and $\hat{W}_{\tilde{\boldsymbol{u}}}^{\tilde{k}} = \hat{\mathsf{acc}}^{\tilde{h}} R$. Combining the two we get that

$$\hat{W}_{\boldsymbol{u}}^{k-\tilde{k}} = \hat{\mathsf{acc}}^{h-\tilde{h}} \Leftrightarrow$$

$$\hat{W}_{\boldsymbol{u}}^{sh\prod_i u_i + r - s\tilde{h}\prod_i u_i - r} = \hat{\mathsf{acc}}^{h-\tilde{h}} \Leftrightarrow$$

$$\hat{W}_{\boldsymbol{u}}^{(s\prod_i u_i)(h-\tilde{h})} = \hat{\mathsf{acc}}^{h-\tilde{h}}$$

From the low order assumption (which is implied by the adaptive root assumption) we get $\hat{W}_{\boldsymbol{u}}^{s\prod_i u_i} = \hat{\mathsf{acc}}$.

Finally, the extractor runs once the extractor of $\mathsf{cp}\Pi^{\text{bound}}$ to get that $u_i > 2\lambda$.

To conclude the proof, $(2, M)$-special soundness implies knowledge-soundness [11].

Zero-Knowledge: It comes directly from the standard rewinding-simulation Σ-method and the use of the simulators of $\mathsf{cp}\Pi^{\text{modarithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$. □

5.5 Discussion on the Generation and Maintenance of Accumulator Witnesses

The setting In the previous we presented our zero-knowledge protocol. But zero-knowledge proof aside, the membership proof computation is already fairly expessive for RSA accumulators. We consider though that the memberhip proof W is already obtained before computing the zero-knowledge proof and is just taken as input. There are different scenarios where it is plausible that users hold precomputed witnesses for their set elements of interest. These scenarios include for example UTXO-like settings and whitelists (where the elements represent respectively an unspent transaction and an identity).

Aggregating witnesses for singletons Consider a party holding a "set of interest" $\hat{U}$ (the subset of accumulated elements in which it has a stake to prove set membership). As mentioned above we assume that each party holds an accumulator witness for each of the elements in $\hat{U}$. When requested to batch prove membership for $u_1, \ldots, u_m \in \hat{U}$, the party can obtain a single witness for the whole batch (like the one assumed as input in fig. 5.2) without recomputing it from scratch. In RSA accumulators, we can in fact apply a process of *aggregation* among the witnesses. Aggregation uses Shamir's trick [192][31] and proceeds in a tree-like fashion. For a batch of size m it consists of roughly m GCD computations, and a similar number of products and RSA exponentiations with integer inputs of varying size.

Witness generation and maintenance Here we discuss how proving parties can obtain and maintain witnesses for elements in their set of interest[32].

A straightforward way for a user to obtain a witness to their elements of interest is to precompute it from scratch. For a single witness, this involves performing roughly N exponentiations with exponents of 256 bits in an RSA group (where N is the whole set size). There are efficient ways to reuse work and distribute it in parallel for subsets of elements. The naive approach to generate a witness requires less than a minute on an ordinary laptop for a set of size 2^{16}, but it can be costly for larger sets. In order to mitigate this, there exist more sophisticated highly-parallelizable approaches to generate witnesses. For example, those described in Section 4.4 in [168]. As an alternative this can be delegated to a service provider as described for updates in [40] (notice that the witnesses from this service providers does not need to be trusted and can be efficiently verified through the standard accumulator verification algorithm).

Does a party need to recompute their witness from scratch if the accumulator (and its underlying set) changes over time? Fortunately not. A party observing updates to the accumulators can update their witnesses cheaply. For example, appending an element x to the set requires updating the witness by simply exponentiating the old witness to x. Other types of updates (e.g., removal of an element) can be handled through Shamir's trick[33]

If a party cannot observe all updates or if the update process is too demanding, this can be delegate to a (non trusted) service provider as described in [40].

A note on storage: if storing all witnesses for singletons of interests is too demanding, this can be mitigated through some of the disaggregation & aggregation techniques described in [59] and storing only witnesses for "chunks" of elements of interest[34]. A similar technique can also be useful to reduce the complexity of handling updates.

[31] See page 12 in [59].

[32] Here we provide an overview of techniques that can be useful to make these issues practical. Which approach is best is something highly sensitive to idiosyncratic aspects of the domain and a full analysis is out of the scope of this paper.

[33] These operations can be concretely inexpensive for meaningful sizes of the subset of interest. For example, we measure the time required to update 64 witnesses after an element is removed from the set to be around $0.3s$ on an ordinary laptop. Performing a similar update in the event of an element being added to the set is even faster.

[34] The concrete costs for aggregation and disaggregation correspond to the cost of updating witnesses for respectively deletions and additions of elements since they use the same techniques. See also numbers reported in footnote 33.

5.6 Extending our CP-SNARK for batch membership

5.6.1 Dealing with sets of arbitrary elements

The scheme described in the section 5.4 works for sets whose elements are suitably large prime numbers. Working with primes can be a limitation in practical applications. Here we describe how to get rid of this limitation and can support sets of arbitrary elements, such as binary strings. The idea is common in previous work and is to use a suitable collision-resistant hash function that maps arbitrary strings to prime numbers. What is a bit more complicated in our setting is that in order to prove membership of an arbitrary element, we need to prove the mapping to a prime.

Thanks to the commit-and-prove modularity of our protocol we can do this extension easily. This is the same idea used in [31]. Say that the prover holds a commitment $\hat{c}$ to a vector of binary strings $(\hat{u}_1, \ldots, \hat{u}_m)$. To prove the mapping the prover creates a commitment c to the primes $(u_1, \ldots, u_m)$ such that $u_i = \mathsf{Hprime}(\hat{u}_i)$, runs our CP-SNARK with c and adds a proof π_{Hprime} showing that $c, \hat{c}$ commit to elements such that $\forall i : u_i = \mathsf{Hprime}(\hat{u}_i)$. The latter proof can be generated via a CP-SNARK for this hashing relation. In particular, although a computation of Hprime involves several computations of a collision resistant hash function until reaching a prime, for the sake of proving one can use nondeterminism and prove a single hash evaluation (see [31] for details).

5.6.2 Succinct batch proofs of non-membership

We observe that by using a CP-SNARK for batch membership it is also possible to prove batch non-membership, if one accumulates sets using an interval-based encoding. The idea is that the elements of the set $S = \{x_i\}_i$ are ordered, $x_1 < x_2 < \cdots < x_n$, and the accumulator actually contains hashes of consecutive pairs $u_i = H(x_{i-1}, x_i)$. This way, proving that $x \notin S$ translates into proving that there is an element $u_j = H(x_{j-1}, x_j)$ in the accumulator such that $x_{j-1} < x < x_j$. The idea of interval-based non-membership proofs was introduced by Buldas et al. in the context of Merkle trees [51].

5.7 Security proof of DDH-II in the generic group model

Here we formally prove that the DDH-II is secure in the generic group model for hidden-order groups (see Section 3.3.4).

The proof closely follows the one of DDH-II over prime order groups [86] adopting the hidden order group-GGM techniques of [87].

Useful notation:

- $\mathsf{GroupGen}(1^\lambda)$ randomized algorithm that samples a hidden order group $\mathbb{G}$ with order $|\mathbb{G}| \in [B, C]$
- $\alpha(\mathsf{GroupGen}(1^\lambda))$: maximal probability of the largest prime factor p of the group order.
- $\beta(\mathsf{GroupGen}(1^\lambda), M)$: probability that $p < M$, given an arbitrary integer M.

Assumption 2 (f-DDH-II)**.** *Let* $\mathbb{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$ *be a hidden order group with* $1/B = \mathsf{negl}$, $\alpha(\mathsf{GroupGen}(1^\lambda)) = \mathsf{negl}$ *and* $g \leftarrow_\$ \mathbb{G}$*. Let* $\{\mathcal{D}_\lambda\}_\lambda$ *be a family of well-spread distributions*

where the domain of $\mathcal{D}$ is $[1, B]$. Then for any PPT $\mathcal{A}$,

$$\mathsf{Adv}_{\mathcal{A},\text{DDH-II}} := |Pr[\mathcal{A}(g^x, g^y, g^{xy}) = 0] - Pr[\mathcal{A}(g^x, g^y, g^t) = 0]| = \\ = \mathsf{negl}(\lambda),$$

where $x \leftarrow_\$ \mathcal{D}_\lambda$ and $y, t \leftarrow_\$ [1, 2^\lambda C]$

Theorem 19. *Assumption 2 holds in the generic group model.*

Proof. Let $\mathcal{A}$ be a polynomial-time generic adversary making at most $q = \mathsf{poly}$ generic oracle queries. The challenger chooses a random encoding $\sigma : \mathbb{G} \to \{0,1\}^\ell$, where $\ell \gg \log(|\mathbb{G}|)$. The challenger plays the following game with $\mathcal{A}$: $\mathcal{C}$ chooses $\sigma_0, \ldots, \sigma_4 \leftarrow_\$ \{0,1\}^\ell$ (that virtually correspond to $(g, g^x, g^y, g^{xy}, g^t)$ resp.). Each group element (and thus its corresponding encoding) is associated with a polynomial in $\mathbb{Z}[X, Y, T_0, T_1]$ (the polynomials are actually over $\mathbb{Z}_{|\mathbb{G}|}$, however the difference with a game where polynomials are over $\mathbb{Z}$ is negligible since the group is of unknown order (due to the low order assumption)). The challenger for the rest of the game maintains a list $\mathcal{L}$ of encoding-polynomial pairs, which is initiated as $\mathcal{L} := \{(1, \sigma_0), (X, \sigma_1), (X, \sigma_2), (T_0, \sigma_3), (T_1, \sigma_4)\}$.

Then $\mathcal{C}$ simulates the generic oracles as follows:

- Group Action: given σ_i, σ_j, $\mathcal{C}$ searches the list $\mathcal{L}$, recovers the corresponding polynomials F_i, F_j and computes $F_i + F_j$. If $F_i + F_j \in \mathcal{L}$ then $\mathcal{C}$ sends to $\mathcal{A}$ the corresponding (of $F_i + F_j$) value σ. Otherwise, samples a random $\sigma \leftarrow_\$ \{0,1\}^\ell$ and responds with this to $\mathcal{A}$. Also appends $(F_i + F_j, \sigma)$ to $\mathcal{L}$.
- Inversion: Similarly, given σ, $\mathcal{C}$ searches the list $\mathcal{L}$, recovers the corresponding polynomial F and computes $-F$. If $-F \in \mathcal{L}$ then $\mathcal{C}$ sends to $\mathcal{A}$ the corresponding (of $-F$) value σ. Otherwise, samples a random $\sigma \leftarrow_\$ \{0,1\}^\ell$ and responds with this to $\mathcal{A}$. Also appends $(-F, \sigma)$ to $\mathcal{L}$.
- Random element: $\mathcal{C}$ chooses a random $\sigma \leftarrow_\$ \{0,1\}^\ell$ and responds with this to $\mathcal{A}$

In all above queries $\mathcal{C}$ takes special care so that a new random σ doesn't coincide with an old $\sigma_i \in \mathcal{L}$ (which anyway happens with negligible probability.

Finally, $\mathcal{A}$ outputs a bit b'. Afterwards the challenger samples $x \leftarrow_\$ \mathcal{D}_\lambda$, $y, t \leftarrow_\$ [|\mathbb{G}|]$ and $b \leftarrow_\$ \{0,1\}$ and sets $X := x$, $Y := y$, $T_b := xy$, $T_{1-b} := t$.

It is clear that the simulation of $\mathcal{C}$ doesn't provide any information to $\mathcal{A}$, except for the case where two polynomials of $\mathcal{L}$ happen to collide for the randomly chosen instantiation of the problem (i.e. values x, y, t, b). It remains to show that the probability of such event, $F_i(x, y, xy, t) = F_j(x, y, xy, t)$ for i, j in the list, is negligible.

First let p be the biggest factor of the order of $\mathbb{G}$ and with degree d. Then there is a group H such that $\mathbb{G} \cong \mathbb{Z}_{p^d} \times H$. The random choice of x, y, t is equivalent to as random independent choice of $x', y', t' \leftarrow_\$ \mathbb{Z}_{p^d}$ and $x'', y'', t'' \leftarrow_\$ H$. Then if $F_i(x, y, xy, t) = F_j(x, y, xy, t)$ over $\mathbb{Z}_{|\mathbb{G}|}$ it should also hold that $F_i(x', y', x'y', t') = F_j(x', y', x'y', t') \mod p^r$. According to [87, Lemma 3] this probability is at most:

$$\frac{q(q-1)}{2} \cdot \left(q\alpha(\mathsf{GroupGen}(1^\lambda)) + \beta(\mathsf{GroupGen}(1^\lambda), M) + \frac{2}{M} \right)$$

for any $0 < M < \min\{2^{H_\infty(\mathcal{D}_\lambda)}, p\}$. The $q(q-1)/2$ factor counts for the different combinations of i, j.

Now if we set $1/M = \sqrt{\alpha(\mathsf{GroupGen}(1^\lambda))}$ then $\beta(\mathsf{GroupGen}(1^\lambda), M) \leq M \cdot \alpha(\mathsf{GroupGen}(1^\lambda)) = \sqrt{\alpha(\mathsf{GroupGen}(1^\lambda))}$ (see [87, Fact 1]). Which gives the overall success probability of $\mathcal{A}$:

$$\frac{q^2(q-1)}{2}\alpha(\mathsf{GroupGen}(1^\lambda)) + \frac{q(q-1)}{2}\sqrt{\alpha(\mathsf{GroupGen}(1^\lambda))} \\ +q(q-1)\sqrt{\alpha(\mathsf{GroupGen}(1^\lambda))}$$

which is negligible, since $\alpha(\mathsf{GroupGen}(1^\lambda)) = \mathsf{negl}$ by assumption and $q = \mathsf{poly}$.

□

Fine-grained security If we set $M = 2^{H_\infty(\mathcal{D}_\lambda)}$ then the final success probability of the adversary gets

$$O\left(q^3\alpha + q^2\alpha 2^{H_\infty(\mathcal{D}_\lambda)} + q^2 2^{-H_\infty(\mathcal{D}_\lambda)}\right)$$

which gives us the constraints $H_\infty(\mathcal{D}_\lambda) \geq 2\lambda$ and $\alpha \leq 2^{-4\lambda}$, to avoid sub-exponential attacks. The former indicates that the well-spread distribution should have a domain of size at least $2^{2\lambda}$. The latter holds in all the known hidden order group: if $B > 2^{4\lambda}$ (which is true in all commonly used hidden order groups) then from the prime number theorem $\alpha(\mathsf{GroupGen}(1^\lambda)) \approx 2^{-4\lambda+\log 4\lambda} \ln 2$.

Part III

ADVANCED VECTOR COMMITMENTS

6

INCREMENTALLY AGGREGATABLE VECTOR COMMITMENTS

The results of this chapter appear in a paper under the title "Incrementally Aggregatable Vector Commitments and Applications to Verifiable Decentralized Storage" published at the ASIACRYPT 2020 conference [59].

6.1 Technical Contributions

The main technical contributions of the chapter are:

- **Defining the notion of Incrementally Aggregatable SVCs.** We put forth the notion of incremental aggregation of (sub)Vector Commitments. We formalize the notion providing proper definitions.

- **Precomputation for Incrementally Aggregatable SVCs.** To overcome the barrier of generating each opening in linear time[35] $O_\lambda(n)$, we propose an alternative preprocessing-based method. The idea is to precompute at commitment time an auxiliary information consisting of n/B openings, one for each batch of B positions of the vector. Next, to generate an opening for an arbitrary subset of m positions, one uses the incremental aggregation property in order to disaggregate the relevant subsets of precomputed openings, and then further aggregate for the m positions. Concretely, with this method, in our construction we can do the preprocessing in time $O_\lambda(n \log n)$ and generate an opening for m positions in time roughly $O_\lambda(mB \log n)$.

  With the VC of [40], a limited version of this approach is also viable: one precomputes an opening for each bit of the vector in $O_\lambda(n \log n)$ time; and then, at opening time, one uses their one-hop aggregation to aggregate relevant openings in time roughly $O_\lambda(m \log n)$. This however comes with a huge drawback: one must store one opening (of size $p(\lambda) = \mathsf{poly}(\lambda)$ where λ is the security parameter) for every bit of the vector, which causes a prohibitive storage overhead, i.e., $p(\lambda) \cdot n$ bits in addition to storing the vector $\boldsymbol{v}$ itself.

  With incremental aggregation, we can instead tune the chunk size B to obtain flexible time-memory tradeoffs.

[35] We use the notation $O_\lambda(\cdot)$ to include the factor depending on the security parameter λ. Writing "$O_\lambda(t)$" essentially means "$O(t)$ cryptographic operations".

- **Constructing VCs with incremental aggregation.** Turning to realizing SVC schemes with our new incremental aggregation property, we propose two SVC constructions that work in hidden-order groups [87] (instantiatable using classical RSA groups, class groups [49] or the recently proposed groups from Hyperelliptic Curves [94]).

  Our first SVC has constant-size public parameters and constant-size subvector openings, and its security relies on the Strong RSA assumption and an argument of knowledge in the generic group model. Asymptotically, its efficiency is similar to the SVC of Boneh et al. [40], but concretely we outperform [40].

  For the second construction, we show how to modify the RSA-based SVC of [145] (which in turn extends the one of [66] to support subvector openings) in order to make it with *constant-size* parameters and to achieve incremental aggregation. Compared to the first construction, it is more efficient and based on more standard assumptions, in the standard model.

- **Efficient Arguments of Knowledge of Subvector Opening.** As an additional result, we propose efficient arguments of knowledge (AoK) with constant-size proofs for our first VC. The first AoK can prove knowledge of the subvector that opens a commitment at a public set of positions, and it extends to proving that two commitments share a common subvector. The second AoK is similar except that the subvector one proves knowledge of is also committed; essentially one can create two vector commitments C and C' together with a short proof that C' is a commitment to a subvector of the vector committed in C.

  An immediate application of our first AoK is a *keyless proof of storage* (PoS) protocol with compact proofs. PoS allows a client to verify that a server is storing intactly a file via a short-communication challenge-response protocol. A PoS is said *keyless* if no secret key is needed by clients, a property useful in open systems where the client is a set of distrustful parties (e.g., verifiers in a blockchain) and the server may even be one of these clients. A classical keyless PoS is based on Merkle trees and random spot-checks [135], recently generalized to work with vector commitments [105]. A drawback of this construction is that proofs grow with the number of spot-checks (and the size of the tree) and become undesirably large in some applications, e.g., if need to be stored in a blockchain. With our AoK we can obtain openings of fixed size, as short as 2KB, which is 40x shorter than those based on Merkle trees in a representative setting without relying on SNARKs (that would be unfeasible in terms of time and memory)[36].

- **Defining VDS.** We define VDS as a collection of algorithms that capture all the properties above; these are the algorithms that can be executed by clients and storage nodes to maintain the system. A client for a file F is anyone who holds a digest δ_F with which it can: verify retrieval queries, verify and apply updates of F (that result in forks of δ_F into some other $\delta_{F'}$). A storage node for some blocks $F_I = \{F_i\}_{i \in I}$ of a file F is anyone that in addition to F_I stores the digest δ_F and a local state st_{F_I} with which it can: answer and certify retrieval queries for any subset of F_I; push and certify updates of F that involve blocks in F_I; verify and apply updates of F from other nodes. Finally, any node can aggregate retrieval certificates for different blocks of the same file.

  In our VDS notion, an update of F can be: (i) a modification of some blocks, (ii) appending new blocks, or (iii) deleting some blocks (from the end). In all cases, an update of F results

[36] We provide further details in Section 6.6

into a file F' and a new digest $\delta_{F'}$.

In terms of efficiency, in VDS the digests and every certificate (for both retrieval queries or modifications) are required to be of size at most $O(\log |F|)$; similarly, the storage node's local state st_{F_I} has size at most $O(|F_I| + \log |F|)$. In a nutshell, no node should run linearly in the size of the file (unless it is explicitly storing it in full).

The main security property of a VDS scheme intuitively requires that no efficient adversary can create a certificate for falsified data blocks (or updates) that passes verification. As an extra security property, we also consider the possibility that anyone holding a digest δ_F can check if the DSN is storing correctly F without having to retrieve it. Namely, we let VDS provide a Proof of Storage mechanism, which we define similarly to Proof of Retrievability [135] and Proof of Data Possession [10]. Similarly to the case of data retrieval queries, the creation of these proofs of storage must be possible while preserving the aforementioned properties of locality and no-central-coordination.

- **Constructing VDS.** We propose two constructions of VDS in hidden-order groups. Both our VDS schemes are obtained by extending our first and second SVC scheme respectively, in order to handle updates and to ensure that all such update operations can be performed locally. In particular we show crucial use of the new properties of our construction: subvector openings, incremental aggregation and disaggregation, and arguments of knowledge for sub-vector commitments (the latter for the first scheme only).

  Our two VDS schemes are based on the Strong RSA [18] and Strong distinct-prime-product root [145], and Low Order [39] assumptions and have similar performances. The second scheme has the interesting property that the storage node can perform and propagate updates by running in time that is independent of even its total local storage. Our first scheme instead supports an additional type of update that we call "CreateFrom". In it, a storage node holding a prefix F' of a file F can publish a new digest $\delta_{F'}$ corresponding to F' as a new file *and* convince any client about its correctness *without the need for the client to know neither* F *nor* F'.[37] As a potential use case for this feature, consider a network that is supposed to store the entire editing history of some data (e.g., one or more files of a Git project); namely the i-th block of the VDS file contains the data value after the i-th edit (e.g., the i-th Git commit). Then "CreateFrom" can be used to verifiably create a digest of any past version of the data (e.g., of a fork at any point in the past). Finally, our approach is not limited to a prefix of the file but to whatever subset of indices we want to create the new file from.

  It is worth noting that by abstracting the ideas of our constructions, other VDS schemes can be obtained using Merkle trees or RSA accumulators.[38] Compared to a Merkle-tree based solution, we can achieve constant-size certificates for every operation as well as to (efficiently) support compact proofs of storage without expensive SNARKs[39]. Compared to RSA Accumulators, our first VDS scheme takes advantage of our AoK thanks to which it supports CreateFrom updates and compact proofs of storage.

  Finally, we note that VDS shares similarities with the notion of updatable VCs [66] extended

[37] This can be seen as a deletion that can be performed without holding the blocks to be deleted and is more efficient to verify when the prefix F' is much smaller than F.

[38] In fact, a similar idea from RSA accumulators was discussed in [40].

[39] In Merkle trees certificates depend logarithmically on the file size and linearly on the number of blocks (since they are not aggregatable).

with incrementally aggregatable subvector openings. There are two main differences. First, in VDS updates can be applied with the help of a short advice created by the party who created the update, whereas in updatable VC this is possible having only the update's description. The second difference is that in VDS the public parameters must be short, otherwise nodes could not afford storing them. This is not necessarily the case in VCs and in fact, to the best of our knowledge, there exists no VC construction with short parameters that is updatable (according to the updatability notion of [66]) and has incrementally aggregatable subvector openings. We believe this is an interesting open problem.

6.2 Building Blocks

6.2.1 Succinct Arguments of Knowledge for Hidden Order Groups.

We recall two concrete succinct AoK protocols for the exponentiation relation in groups of unknown order that have been recently proposed by Boneh et. al. [40]. Both protocols work for a hidden order group $\mathbb{G}$ generated by Ggen in which the adaptive root assumption holds. Also, they are public-coin protocols that can be made non-interactive in the random oracle model using the Fiat-Shamir [101] heuristic and its generalization to multi-round protocols [26].

1. Protocol PoE: is an argument system for the following relation:

$$R_{\mathsf{PoE}} = \left\{((u, w, x) \in \mathbb{G}^2 \times \mathbb{Z}, \varnothing) \ : \ u^x = w \in \mathbb{G} \quad \right\}$$

   PoE is a sound argument system under the adaptive root assumption for Ggen. It is neither zero-knowledge nor knowledge sound. Its main feature is *succinctness*, as the verifier can get convinced about $u^x = w$ without having to execute the exponentiation herself. Moreover the information sent by the prover is only 1 group element.[40]

2. Protocol PoKE: is an argument of knowledge for the following relation, parametrized by a generator $g \in \mathbb{G}$:

$$R_{\mathsf{PoKE}} = \left\{(w, x) \in \mathbb{G} \times \mathbb{Z} \ : \ g^x = w \in \mathbb{G} \quad \right\}$$

   PoKE is an argument of knowledge that in [40] is proven secure in the generic group model for hidden order groups [87]. This protocol is also succinct consisting of only 1 group element and 1 field element in $\mathbb{Z}_{2^\lambda}$.

3. Protocol $\mathsf{PoKE2}$: is an argument of knowledge for the following relation, parametrized by a generator $g \in \mathbb{G}$:

$$R_{\mathsf{PoKE2}} = \left\{((w, u) \in \mathbb{G}^2, x \in \mathbb{Z}) \ : \ u^x = w \in \mathbb{G} \quad \right\}$$

   $\mathsf{PoKE2}$ is similar to PoKE but it is secure for arbitrary bases u chosen by the adversary, instead of bases randomly sampled a priori as in PoKE. Similarly, it is an argument of knowledge in the generic group model for hidden order groups and is also succinct, with a proof consisting of 2 group elements and 1 element of $\mathbb{Z}_{2^\lambda}$.

[40] Technically, this protocol is not succinct as there is no witness and the verifier must read and process the exponent x; however, verification is still more efficient than running the full exponentiation.

6.3 Vector Commitments with Incremental Aggregation

In this section, we recall the notion of vector commitment with subvector openings [66, 145, 40] and then we formally define our new incremental aggregation property.

Vector Commitments with Specializable Universal CRS. The notion of VCs defined above slightly generalizes the previous ones in which the generation of public parameters (aka common reference string) depends on a bound n on the length of the committed vectors. In contrast, in our notion Setup is length-independent. To highlight this property, we also call this primitive *vector commitments with universal CRS*.

Here we formalize a class of VC schemes that lies in between VCs with universal CRS (as defined above) and VCs with length-specific CRS (as defined in [66]). Inspired by the recent work of Groth et al. [127], we call these schemes VCs with *Specializable* (Universal) CRS. In a nutshell, these are schemes in which the algorithms $\mathsf{Com}, \mathsf{Open}$ and Ver work on input a length-specific CRS pp_n. However, this pp_n is generated in two steps: (i) a *length-independent, probabilistic* setup $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$, and (ii) a *length-dependent, deterministic* specialization $\mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$. The advantage of this model is that, being $\mathsf{Specialize}$ deterministic, it can be executed by anyone, and it allows to re-use the same pp for multiple vectors lengths.

Definition 28 (VCs with Specializable CRS)**.** *A VC scheme* VC *has a specializable CRS if there exists a DPT algorithm* $\mathsf{Specialize}(\mathsf{pp}, n)$ *that, on input a (universal) CRS* pp *generated by* $\mathsf{Setup}(1^\lambda, \mathcal{M})$ *and an integer* $n = \mathsf{poly}(\lambda)$*, produces a specialized CRS* pp_n *such that the algorithms* Com*,* Open *and* Ver *can be defined in terms of algorithms* $\mathsf{Com}^\star$*,* $\mathsf{Open}^\star$ *and* $\mathsf{Ver}^\star$ *as follows:*

- $\mathsf{Com}(\mathsf{pp}, \boldsymbol{v})$ *sets* $n := |\boldsymbol{v}|$*, runs* $\mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$ *and* $(C^\star, \mathsf{aux}^\star) \leftarrow \mathsf{Com}^\star(\mathsf{pp}_n, \boldsymbol{v})$*, and returns* $C := (C^\star, n)$ *and* $\mathsf{aux} := (\mathsf{aux}^\star, n)$.
- $\mathsf{Open}(\mathsf{pp}, I, \boldsymbol{y}, \mathsf{aux})$ *parses* $\mathsf{aux} := (\mathsf{aux}^\star, n)$*, runs* $\mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$ *and returns* $\pi_I \leftarrow \mathsf{Open}^\star(\mathsf{pp}_n, I, \boldsymbol{y}, \mathsf{aux}^\star)$.
- $\mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}, \pi_I)$ *parses* $C := (C^\star, n)$*, runs* $\mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$ *and returns* $\mathsf{Ver}^\star(\mathsf{pp}_n, C^\star, I, \boldsymbol{y}, \pi_I)$.

Basically, for a VC with specializable CRS it is sufficient to describe the algorithms Setup, $\mathsf{Specialize}, \mathsf{Com}^\star, \mathsf{Open}^\star$ and $\mathsf{Ver}^\star$. Furthermore, a concrete advantage is that when working on multiple commitments, openings and verifications that involve the same length n, one can execute $\mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$ only once.

6.3.1 Incrementally Aggregatable Subvector Openings

In a nutshell, aggregation means that different proofs of different subvector openings can be merged together into a single *short* proof which can be created *without* knowing the entire committed vector. Moreover, this aggregation is composable, namely aggregated proofs can be further aggregated. Following a terminology similar to that of aggregate signatures, we call this property *incremental aggregation* (but can also be called *multi-hop aggregation*). In addition to aggregating openings, we also consider the possibility to "disaggregate" them, namely from an opening of positions in the set I one can create an opening for positions in a set $K \subset I$.

We stress on the two main requirements that make aggregation and disaggregation non-trivial: all openings must remain short (independently of the number of positions that are being opened), and aggregation (resp. disaggregation) must be computable locally, i.e., without knowing the whole committed vector. Without such requirements, one could achieve this property by simply concatenating openings of single positions.

Definition 29 (Aggregatable Subvector Openings)**.** *A vector commitment scheme* VC *with subvector openings is called* aggregatable *if there exists algorithms* $\mathsf{VC.Agg}$, $\mathsf{VC.Disagg}$ *working as follows:*

$\mathsf{VC.Agg}(\mathsf{pp}, (I, \boldsymbol{v}_I, \pi_I), (J, \boldsymbol{v}_J, \pi_J)) \to \pi_K$ *takes as input two triples* $(I, \boldsymbol{v}_I, \pi_I), (J, \boldsymbol{v}_J, \pi_J)$ *where* I *and* J *are sets of indices,* $\boldsymbol{v}_I \in \mathcal{M}^{|I|}$ *and* $\boldsymbol{v}_J \in \mathcal{M}^{|J|}$ *are subvectors, and* π_I *and* π_J *are opening proofs. It outputs a proof* π_K *that is supposed to prove opening of values in positions* $K = I \cup J$*.*

$\mathsf{VC.Disagg}(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, K) \to \pi_K$ *takes as input a triple* $(I, \boldsymbol{v}_I, \pi_I)$ *and a set of indices* $K \subset I$*, and it outputs a proof* π_K *that is supposed to prove opening of values in positions* K*.*

The aggregation algorithm $\mathsf{VC.Agg}$ *must guarantee the following two properties:*

Aggregation Correctness. *Aggregation is (perfectly) correct if for all* $\lambda \in \mathbb{N}$*, all honestly generated* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$*, any commitment* C *and triple* $(I, \boldsymbol{v}_I, \pi_I)$ *s.t.* $\mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{v}_I, \pi_I) = 1$*, the following two properties hold:*

1. *for any triple* $(J, \boldsymbol{v}_J, \pi_J)$ *such that* $\mathsf{Ver}(\mathsf{pp}, C, J, \boldsymbol{v}_J, \pi_J) = 1$*,*

$$\Pr\left[\mathsf{Ver}(\mathsf{pp}, C, K, \boldsymbol{v}_K, \pi_K) = 1 \ : \ \pi_K \leftarrow \mathsf{VC.Agg}(\mathsf{pp}, (I, \boldsymbol{v}_I, \pi_I), (J, \boldsymbol{v}_J, \pi_J))\right] = 1$$

   where $K = I \cup J$ *and* $\boldsymbol{v}_K$ *is the ordered union* $\boldsymbol{v}_{I \cup J}$ *of* $\boldsymbol{v}_I$ *and* $\boldsymbol{v}_J$*;*

2. *for any subset of indices* $K \subset I$*,*

$$\Pr\left[\mathsf{Ver}(\mathsf{pp}, C, K, \boldsymbol{v}_K, \pi_K) = 1 \ : \ \pi_K \leftarrow \mathsf{VC.Disagg}(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, K)\right] = 1$$

   where $\boldsymbol{v}_K = (v_{i_l})_{i_l \in K}$*, for* $\boldsymbol{v}_I = (v_{i_1}, \ldots, v_{i_{|I|}})$*.*

Aggregation Conciseness. *There exists a fixed polynomial* $p(\cdot)$ *in the security parameter such that all openings produced by* $\mathsf{VC.Agg}$ *and* $\mathsf{VC.Disagg}$ *have length bounded by* $p(\lambda)$*.*

We remark that the notion of specializable CRS can apply to aggregatable VCs as well. In this case, we let $\mathsf{VC.Agg}^\star$ (resp. $\mathsf{VC.Disagg}^\star$) be the algorithm that works on input the specialized pp_n instead of pp.

6.4 Applications of Incremental Aggregation

We discuss two general applications of the incremental aggregation property of vector commitments.

One application is generating subvector openings in a distributed and decentralized way. Namely, assume a set of parties hold each an opening of some subvector. Then it is possible to create a (concise) opening for the union of their subvectors by using the $\mathsf{VC.Agg}$ algorithm.

Moreover, the incremental (aka multi-hop) aggregation allows these users to perform this operation in an arbitrary order, hence no coordination or a central aggregator party are needed. This application is particularly useful in our extension to verifiable decentralized storage.

The second application is to generate openings in a faster way via preprocessing. As we mentioned in the introduction, this technique is useful in the scenario where a user commits to a vector and then must generate openings for various subvectors, which is for example the use case when the VC is used for proofs of retrievability and IOPs [40].

So, here the goal is to achieve a method for computing subvector openings in time sub-linear in the total size of the vector, which is the barrier in all existing constructions. To obtain this speedup, the basic idea is to (A) compute and store openings for all the position at commitment time, and then (B) use the aggregation property to create an opening for a specific set of positions. In order to obtain efficiency using this approach it is important that both steps (A) and (B) can be computed efficiently. In particular, step (A) is challenging since typically computing one opening takes linear time, hence computing all of them would take quadratic time.

In this section, we show how steps (A) and (B) can benefit from disaggregation and aggregation respectively. As a preliminary for this technique, we begin by describing two generic extensions of (incremental) aggregation (resp. disaggregation) that support many inputs (resp. outputs). Then we show how these extended algorithms can be used for committing and opening with preprocessing.

6.4.1 Divide-and-Conquer Extensions of Aggregation and Disaggregation

We discuss how the incremental property of our aggregation and disaggregation can be used to define two extended versions of these algorithms. The first one is an algorithm that can aggregate many openings for different sets of positions into a single opening for their union. The second one does the opposite, namely it disaggregates one opening for a set I into many openings for partitions of I.

6.4.1.1 Aggregating Many Openings

We consider the problem of aggregating several openings for sets of positions $I_1, \ldots, I_m$ into a single opening for $\bigcup_{j=1}^{m} I_j$. Our syntax in Definition 29 only considers pairwise aggregation. This can be used to handle many aggregations by executing the pairwise aggregation in a sequential (or arbitrary order) fashion. Sequential aggregation might however be costly since it would require executing VC.Agg on increasingly growing sets. If $f_a(k)$ is the complexity of VC.Agg on two sets of total size k, then the total complexity of the sequential method is $\sum_{j=2}^{m} f(\sum_{l=1}^{j-1} |I_l| + |I_j|)$, which for example is quadratic in m, for $f_a(k) = \Theta(k)$.

In Fig. 6.1, we show an algorithm, VC.AggManyToOne, that is a nearly optimal solution for aggregating m openings based on a divide-and-conquer methodology. Assuming for simplicity that all I_j's have size bounded by some s, then the complexity of VC.AggManyToOne is given by the following recurrence relation:

$$T(m) = 2T\left(\frac{m}{2}\right) + f_a(s \cdot m)$$

which for example solves to $\Theta(s \cdot m \log m)$ if $f_a(n) \in \Theta(n)$, or to $\Theta(s \cdot m \log(sm) \log m)$ if $f_a(n) \in \Theta(n \log n)$.

$\mathsf{VC.AggManyToOne}(\mathsf{pp}, (I_j, \boldsymbol{v}_{I_j}, \pi_j)_{j\in[m]})$

1 : **if** $m = 1$ **return** π_1
2 : $m' \leftarrow m/2$
3 : $L \leftarrow \cup_{j=1}^{m'} I_j, \quad R \leftarrow \cup_{j=m'+1}^{m} I_j,$
4 : $\pi_L \leftarrow \mathsf{VC.AggManyToOne}(\mathsf{pp}, (I_j, \boldsymbol{v}_{I_j}, \pi_j)_{j=1,\ldots,m'})$
5 : $\pi_R \leftarrow \mathsf{VC.AggManyToOne}(\mathsf{pp}, (I_j, \boldsymbol{v}_{I_j}, \pi_j)_{j=m'+1,\ldots,m})$
6 : $\pi_{L\cup R} \leftarrow \mathsf{VC.Agg}(\mathsf{pp}, (L, \boldsymbol{v}_L, \pi_L), (R, \boldsymbol{v}_R, \pi_R))$
7 : **return** $\pi_{L\cup R}$

$\mathsf{VC.DisaggOneToMany}(\mathsf{pp}, B, I, \boldsymbol{v}_I, \pi_I)$

1 : **if** $n = |I| = B$ **return** π_I
2 : $n' \leftarrow n/2$
3 : $L \leftarrow \cup_{j=1}^{n'} i_j, \quad R \leftarrow \cup_{j=n'+1}^{m} i_j,$
4 : $\pi'_L \leftarrow \mathsf{VC.Disagg}(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, L)$
5 : $\pi'_R \leftarrow \mathsf{VC.Disagg}(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, R)$
6 : $\boldsymbol{\pi}_L \leftarrow \mathsf{VC.DisaggOneToMany}(\mathsf{pp}, B, L, \boldsymbol{v}_L, \pi'_L)$
7 : $\boldsymbol{\pi}_R \leftarrow \mathsf{VC.DisaggOneToMany}(\mathsf{pp}, B, R, \boldsymbol{v}_R, \pi'_R)$
8 : **return** $\boldsymbol{\pi}_L || \boldsymbol{\pi}_R$

Figure 6.1: Extensions of Aggregation and Disaggregation

6.4.1.2 Disaggregating from One to Many Openings

We consider the problem that is dual to the one above, namely how to disaggregate an opening for a set I into several openings for sets $I_1, \ldots, I_m$ that form a partition of I. Our syntax in Definition 29 only considers disaggregation from one set I to one subset K of I. Similarly to the aggregation case, disaggregating from one set to many subsets can be trivially obtained via a sequential application of VC.Disagg on all pairs (I, I_j). This however can be costly if the number of partitions approaches the size of I, e.g., if we want to disaggregate to all the elements of I.

In Fig. 6.1, we show an algorithm, VC.DisaggOneToMany, we show a divide-and-conquer algorithm for disaggregating an opening for a set I of size m into $m' = m/B$ openings, each for a partition of size B. For simplicity, we assume that m is a power of 2, and $B \mid m$.

Let $f_d(|I|)$ be the complexity of VC.Disagg. Then the complexity of VC.DisaggOneToMany is given by the following recurrence relation:

$$T(m) = 2T\left(\frac{m}{2}\right) + 2f_d(m/2)$$

which for example solves to $\Theta(m \log(m/B))$ if $f_d(n) \in \Theta(n)$, or to $\Theta(m \log m \log(m/B))$ if $f_d(n) \in \Theta(n \log n)$.

6.4.2 Committing and Opening with Precomputation

We present a construction of committing and opening algorithms (denoted VC.PPCom and VC.FastOpen respectively) that works generically for any SVC with incremental aggregation

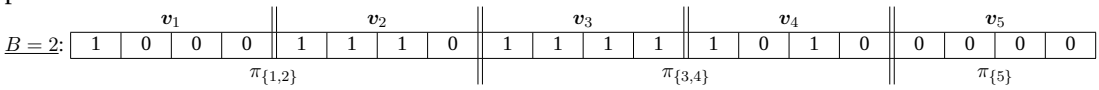
and that, by relying on preprocessing, can achieve fast opening time.

Our preprocessing method works with a flexible choice of a parameter B that allows for different time-memory tradeoffs. In a nutshell, ranging from 1 to n, a larger B reduces memory but increases opening time while a smaller B (e.g., $B = 1$) requires larger storage overhead but gives the fastest opening time.

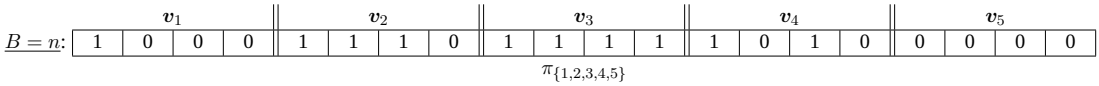
Let B be an integer that divides n, and let $n' = n/B$. The core of our idea is that, during the commitment stage, one can create openings for $n' = n/B$ subvectors of $\boldsymbol{v}$ that cover the all vector (e.g., B contiguous positions). Let $\pi_{P_1}, \ldots, \pi_{P_{n'}}$ be such openings; these elements are stored as advice information.

Next, in the opening phase, in order to compute the opening for a subvector $\boldsymbol{v}_I$ of m positions, one should: (i) fetch the subset of openings π_{P_j} such that, for some S, $I \subseteq \cup_{j \in S} P_j$, (ii) possibly disaggregate some of them and then aggregate in order to compute π_I.

To give a very general example of the above process, assume one has stored $\pi_{\{1,2\}}$ and $\pi_{\{3,4,5\}}$ and is asked for $\pi_{\{2,3\}}$, then she has to compute first π_2 and π_3 by disaggregating $\pi_{\{1,2\}}$ and $\pi_{\{3,4,5\}}$ respectively, and then aggregate them to $\pi_{\{2,3\}}$. Below are two more examples in picture:



$\approx p(\lambda) \cdot n/2$ bits in opening advice



$p(\lambda)$ bits in opening advice

The two algorithms VC.PPCom and VC.FastOpen are described in detail in Fig. 6.2.

In terms of auxiliary storage, in addition to the vector $\boldsymbol{v}$ itself, one needs at most $(n/B)p(\lambda)$ bits, where $p(\lambda)$ is the polynomial bounding the conciseness of the SVC scheme. In terms of time complexity, VC.PPCom requires one execution of Com, one execution of Open, and one execution of VC.DisaggOneToMany, which in turn depends on the complexity of VC.Disagg; VC.FastOpen requires to perform (at most) $|S|$ disaggregations (each with a set $|I_j|$ such that their sum is $|I|$)[41], and one execution of VC.AggManyToOne on $|S|$ openings. Note that VC.FastOpen's running time depends only on the size m of the set I and size B of the buckets P_j, and thus offers various tradeoffs by adjusting B.

More specific running times depend on the complexity of Com, Open, VC.Agg, and VC.Disagg of the given SVC scheme. See Appendix 6.10 for these results for our construction.

6.5 Our Realizations of Incrementally Aggregatable Vector Commitments

In this section we describe our new SVC realizations.

[41] Note that for $B = 1$ the disaggregation step can be skipped.

$\mathsf{VC.PPCom}(\mathsf{pp}, B, \boldsymbol{v})$

1 : $(C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, \boldsymbol{v})$
2 : $\pi^* \leftarrow \mathsf{Open}(\mathsf{pp}, [n], \boldsymbol{v}, \mathsf{aux})$
3 : $\boldsymbol{\pi} \leftarrow \mathsf{VC.DisaggOneToMany}(\mathsf{pp}, B, [n], \boldsymbol{v}, \pi^*)$
4 : $\mathsf{aux}^* := (\pi_1, \ldots, \pi_{n'}, \boldsymbol{v})$
5 : **return** C, aux^*

$\mathsf{VC.FastOpen}(\mathsf{pp}, B, \mathsf{aux}^*, I)$

1 : **Let** $P_j := \{(j-1)B + i : i \in [B]\}, \forall j \in [n']$
2 : **Let** $I := \{i_1, \ldots, i_m\}$
3 : **Let** S minimal set s.t. $\bigcup_{j \in S} P_j \supseteq I$
4 : **for** $j \in S$ **do** :
5 : $\quad I_j \leftarrow I \cap P_j$
6 : $\quad \pi'_j \leftarrow \mathsf{VC.Disagg}(\mathsf{pp}, P_j, \boldsymbol{v}_{P_j}, \pi_j, I_j)$
7 : **endfor**
8 : $\pi_I \leftarrow \mathsf{VC.AggManyToOne}(\mathsf{pp}, ((I_j, \boldsymbol{v}_{I_j}, \pi'_j))_{j \in S})$
9 : **return** π_I

Figure 6.2: Generic algorithms for committing and opening with precomputation.

6.5.1 Our First SVC Construction

An overview of our techniques. The basic idea underlying our VC can be described as a generic construction from any accumulator with union proofs. Consider a vector of bits $\boldsymbol{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$. In order to commit to this vector we produce two accumulator, Acc_0 and Acc_1, on two partitions of the set $S = \{1, \ldots, n\}$. Each accumulator Acc_b compresses the set of positions i such that $v_i = b$. In other words, Acc_b compresses the set $S_{=b} := \{i \in S : v_i = b\}$ with $b \in \{0,1\}$. In order to open to bit b at position i, one can create an accumulator membership proof for the statement $i \in \tilde{S}_b$ where we denote by $\tilde{S}_b$ the alleged set of positions that have value b.

However, if the commitment to $\boldsymbol{v}$ is simply the pair of accumulators $(\mathsf{Acc}_0, \mathsf{Acc}_1)$ we do not achieve position binding as an adversary could for example include the same element i in both accumulators. To solve this issue we set the commitment to be the pair of accumulators plus a succinct non-interactive proof π_S that the two sets $\tilde{S}_0, \tilde{S}_1$ they compress constitute together a *partition* of S. Notably, this proof π_S guarantees that each index i is in either $\tilde{S}_0$ or $\tilde{S}_1$, and thus prevents an adversary from also opening the position i to the complement bit $1 - b$.

The construction described above could be instantiated with any accumulator scheme that admits an efficient and succinct proof of union. We, though, directly present an efficient construction based on RSA accumulators [**?**, 18, 57, 152, 40] as this is efficient and has some nice extra properties like aggregation and constant-size parameters. Also, part of our technical contribution to construct this VC scheme is the construction of efficient and succinct protocols for proving the union of two RSA accumulators built with different generators.

$\mathsf{Setup}(1^\lambda)$: run $\mathbb{G} \leftarrow_\$ \mathsf{Ggen}(1^\lambda)$, $g_1, g_2, g_3 \leftarrow_\$ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g_1, g_2, g_3)$.
Prover's input: $(\mathsf{crs}, (Y, C), (a, b))$. Verifier's input: $(\mathsf{crs}, (Y, C))$.

$\mathsf{V} \to \mathsf{P}$: $\ell \leftarrow_\$ \mathbb{P}(1, 2^\lambda)$

$\mathsf{P} \to \mathsf{V}$: $\pi := ((Q_Y, Q_C), r_a, r_b)$ computed as follows

- $(q_a, q_b, q_c) \leftarrow (\lfloor a/\ell \rfloor, \lfloor b/\ell \rfloor, \lfloor ab/\ell \rfloor)$
- $(r_a, r_b) \leftarrow (a \mod \ell, b \mod \ell)$
- $(Q_Y, Q_C) := (g_1^{q_a} g_2^{q_b}, g_3^{q_c})$

$\mathsf{V}(\mathsf{crs}, (Y, C), \ell, \pi)$:

- Compute $r_c \leftarrow r_a \cdot r_b \mod \ell$
- Output 1 iff $r_a, r_b \in [\ell] \ \wedge \ Q_Y^\ell g_1^{r_a} g_2^{r_b} = Y \ \wedge \ Q_C^\ell g_3^{r_c} = C$

Figure 6.3: PoProd_2 protocol

6.5.1.1 Succinct AoK Protocols for Union of RSA Accumulators

Let $\mathbb{G}$ be a hidden order group as generated by Ggen, and let $g_1, g_2, g_3 \in \mathbb{G}$ be three honestly sampled random generators. We propose a succinct argument of knowledge for the following relation

$$R_{\mathsf{PoProd}_2} = \left\{ ((Y, C), (a, b)) \in \mathbb{G}^2 \times \mathbb{Z}^2 \ : \ Y = g_1^a g_2^b \wedge C = g_3^{a \cdot b} \quad \right\}$$

Our protocol (described in Fig. 6.3) is inspired by a similar protocol of Boneh et al. [40], PoDDH, for a similar relation in which there is only one generator (i.e., $g_1 = g_2 = g_3$, namely for DDH tuples (g^a, g^b, g^{ab})). Their protocol has a proof consisting of 3 groups elements and 2 integers of λ bits.

As we argue later PoProd_2 is still sufficient for our construction, i.e., for the goal of proving that $C = g_3^c$ is an accumulator to a set that is the union of sets represented by two accumulators $A = g_1^a$ and $B = g_2^b$ respectively. The idea is to invoke PoProd_2 on (Y, C) with $Y = A \cdot B$.

To prove the security of our protocol we rely on the adaptive root assumption and, in a non-black-box way, on the knowledge extractability of the PoKRep and PoKE protocols from [40]. The latter is proven in the generic group model for hidden order groups (where also the adaptive root assumption holds), therefore we state the following theorem.

Theorem 20. *The PoProd_2 protocol is an argument of knowledge for R_{PoProd_2} in the generic group model.*

Proof. For ease of exposition we show a security proof for a slight variant of the protocol PoProd_2. Then, towards the end of this proof we show that security of this variant implies security for our protocol. We let $\mathsf{PoProd}_2{}'$ be the same protocol as PoProd_2 with only difference that the prover computes also $r_c \leftarrow r_a \cdot r_b \pmod \ell$ and sends r_c in the proof, and the verifier $\mathcal{V}$ checks in the verification if $r_c = r_a \cdot r_b \pmod \ell$.

Let $\mathcal{A}' = (\mathcal{A}'_0, \mathcal{A}'_1)$ be an adversary of the Knowledge Extractability of $\mathsf{PoProd}_2{}'$ such that: $((Y, C), \mathsf{state}) \leftarrow \mathcal{A}'_0(\mathsf{crs})$, $\mathcal{A}'_1(\mathsf{crs}, (Y, C), \mathsf{state})$ executes with $\mathcal{V}(\mathsf{crs}, (Y, C))$ the protocol $\mathsf{PoProd}_2{}'$ and the verifier accepts with a non-negligible probability ϵ. We will construct an

extractor $\mathcal{E}'$ that having access to the internal state of $\mathcal{A}'_1$ and on input $(\mathsf{crs}, (Y, C), \mathsf{state})$, outputs a witness (a, b) of $R_{\mathsf{PoProd}_2{}'}$ with overwhelming probability and runs in (expected) polynomial time.

To prove knowledge extractability of $\mathsf{PoProd}_2{}'$ we rely on the knowledge extractability of the protocol PoKRep from [40], which is indeed implicit in our protocol. More precisely, given a $\mathsf{PoProd}_2{}'$ execution between $\mathcal{A}'$ and $\mathcal{V}$, $(\ell, Q_Y, Q_C, r_a, r_b, r_c)$, $\mathcal{E}'$ constructs an adversary $\mathcal{A}_Y = (\mathcal{A}_{Y,0}, \mathcal{A}_{Y,1})$ of PoKRep Knowledge Extractability and, by using the input and internal state of $\mathcal{A}'_1$, simulates an execution between $\mathcal{A}_Y$ and $\mathcal{V}$: $\mathcal{A}_{Y,0}$ outputs $(\mathsf{crs}_Y, Y, \mathsf{state}) := ((\mathbb{G}, g_1, g_2), Y, \mathsf{state})$, $\mathcal{A}_{Y,1}$ outputs (Q_Y, r_a, r_b). It is obvious that if the initial execution is accepted by $\mathcal{V}$ so is the PoKRep execution. From Knowledge Extractability of PoKRep we know that there exists an extractor $\mathcal{E}_Y$ corresponding to $\mathcal{A}_{Y,1}$ that outputs (a, b) such that $g_1^a g_2^b = Y$. Additionally, it is implicit from the extraction that $a = r_a \pmod{\ell}$ and $b = r_b \pmod{\ell}$ (for more details we refer to the Knowledge Extractability proof of PoKRep in [40]). So, $\mathcal{E}'$ uses $\mathcal{E}_Y$ and gets (a, b). Similarly, it simulates PoKE for $g_3^c = C$, uses the extractor $\mathcal{E}_c$ and gets c.

As one can see, the expected running time of $\mathcal{E}'$ is the (expected) time to obtain a successful execution of the protocol plus the running time of the 2 extractors: $\frac{1}{\epsilon} + t_{\mathcal{E}_Y} + t_{\mathcal{E}_c} = \mathsf{poly}(\lambda)$.

Now what is left to prove to conclude our theorem is to show that the extracted a, b, c are such that $a \cdot b = c$ with all but negligible probability. To this end, we observe that we could run $\mathcal{E}'$ a second time using a different random challenge ℓ'; by using again $\mathcal{E}_Y, \mathcal{E}_c$ (after simulating the corresponding PoKRep and PoKE executions) we would get a', b', c' such that $g_1^{a'} g_2^{b'} = Y = g_1^a g_2^b$, $g_3^{c'} = C = g_3^c$. We argue that $a = a'$, $b = b'$ and $c = c'$ holds over the integers with overwhelming probability under the assumption that computing a multiple of the order of the group $\mathbb{G}$ is hard (such assumption is in turn implied by the adaptive root assumption). If such event does not hold one can make a straightforward reduction to this problem. Therefore, we proceed by assuming that from the two executions we have $a = a'$, $b = b'$, and $c = c'$ over the integers. Moreover, since both executions are accepted we have $r'_c = r'_a \cdot r'_b \pmod{\ell'} \Rightarrow c' = a' \cdot b' \pmod{\ell'} \Rightarrow c = a \cdot b \pmod{\ell'}$, but ℓ' was sampled uniformly at random from $\mathbb{P}(\lambda)$ after a, b, c were determined. So $a \cdot b = c$ over the integers, unless with a negligible probability $\leq \frac{\#\{\text{factors of } ab-c\}}{|\mathbb{P}(\lambda)|} \leq \frac{\mathsf{poly}(\lambda)}{|\mathbb{P}(\lambda)|} = \mathsf{negl}(\lambda)$.

Finally, it is trivial to reduce the Knowledge Extractability of PoProd_2 to Knowledge Extractability of $\mathsf{PoProd}_2{}'$. Let a generic adversary $\mathcal{A}$ against the Knowledge Extractability of protocol PoProd_2 such that the verifier accepts with a non-negligible probability ϵ, we can construct a generic adversary $\mathcal{A}'$ against Knowledge Extractability of $\mathsf{PoProd}_2{}'$, so that the verifier accepts with the same probability. $\mathcal{A}'$ runs the $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ algorithm and sends crs to $\mathcal{A}$. The adversary $\mathcal{A}$ outputs $((Y, C), \mathsf{state}) \leftarrow \mathcal{A}_0(\mathsf{crs})$ and sends it to $\mathcal{A}'_0$, which outputs as it is. Then $\mathcal{A}'_1$ interacts with $\mathcal{V}$ in the protocol $\mathsf{PoProd}_2{}'$ (as a prover) and at the same time with $\mathcal{A}_1$ in PoProd_2 (as a verifier). After receiving ℓ from $\mathcal{V}$ it forwards it to $\mathcal{A}_1$. $\mathcal{A}_1$ answers with $\pi := ((Q_Y, Q_C), r_a, r_b)$. $\mathcal{A}'_1$ computes $r_c \leftarrow r_a r_b \mod \ell$ and sends $\pi' := ((Q_Y, Q_C), r_a, r_b, r_c)$ to $\mathcal{V}$. The verifier $\mathcal{V}$ accepts π' with the same probability that a verifier of PoProd_2 would accept π since $r_c = r_a r_b \mod \ell$ in both cases. From Knowledge Extractability of $\mathsf{PoProd}_2{}'$ we know that there is an extractor $\mathcal{E}'$ that outputs a witness (a, b). Then $\mathcal{E} = \mathcal{E}'$ is a valid extractor for PoProd_2. □

In Appendix 6.9 we give a protocol PoProd that proves $g_1^a = A \wedge g_2^b = B$ instead of $g_1^a g_2^b = Y$ (i.e., a version of PoDDH with different generators). Despite being conceptually

simpler, it is slightly less efficient than PoProd_2, and thus use the latter in our VC construction.

Hash to prime function and non-interactive PoProd_2. Our protocols can be made non-interactive by applying the Fiat-Shamir transform. For this we need an hash function that can be modeled as a random oracle and that maps arbitrary strings to prime numbers, i.e., $\mathsf{Hprime} : \{0,1\}^* \to \mathbb{P}(1, 2^{2\lambda})$[42]. A simple way to achieve such a function is to apply a standard hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ to an input $\boldsymbol{y}$ together with a counter i, and if $p_{y,i} = \mathsf{H}(\boldsymbol{y}, i)$ is prime then output $p_{y,i}$, otherwise continue to $\mathsf{H}(\boldsymbol{y}, i+1)$ and so on, until a prime is found. Due to the distribution of primes, the expected running time of this method is $O(\lambda)$, assuming that H's outputs are uniformly distributed. We do not insist, though, in the previous or any other specific instantiation of Hprime in this work. For more discussion on hash-to-prime functions we refer to [115, 55, 84, 40, 168].

6.5.1.2 Our First SVC Construction

Now we are ready to describe our SVC scheme. For an intuition we refer the reader to the beginning of this section. Also, we note that while the intuition was given for the case of committing to a vector of bits, our actual VC construction generalizes this idea to vectors where each item is a *block of k bits*. This is done by creating $2k$ accumulators, each of them holding sets of indices i for specific positions inside each block v_j.

Notation and Building Blocks. To describe our scheme we use the notation below:

- Our message space is $\mathcal{M} = \{0,1\}^k$. Then for a vector $\boldsymbol{v} \in \mathcal{M}^n$, we denote with $i \in [n]$ the vector's position, i.e., $v_i \in \mathcal{M}$, and with $j \in [k]$ the position of its j'th bit. So v_{ij} denotes the j-th bit in position i.
- We make use of a deterministic collision resistant function $\mathsf{PrimeGen}$ that maps integers to primes. In our construction we do not need its outputs to be random (see e.g., [40] for possible instantiations).
- As a building block, we use the PoProd_2 AoK from the previous section.
- $\mathsf{PartndPrimeProd}(I, \boldsymbol{y}) \to ((a_{I,1}, b_{I,1}), \ldots, (a_{I,k}, b_{I,k}))$: given a set of indices $I = \{i_1, \ldots, i_m\} \subseteq [n]$ and a vector $\boldsymbol{y} \in \mathcal{M}^m$, this function computes

$$(a_{I,j}, b_{I,j}) := \left(\prod_{l=1:y_{lj}=0}^{m} p_{i_l}, \prod_{l=1:y_{lj}=1}^{m} p_{i_l} \right) \quad \text{for } j = 1, \ldots, k$$

  where $p_i \leftarrow \mathsf{PrimeGen}(i)$ for all i.

  Basically, for every bit position $j \in [k]$, the function computes the products of primes that correspond to, respectively, 0-bits and 1-bits.

  In the special case where $I = [n]$, we omit the set of indices from the notation of the outputs, i.e., $\mathsf{PartndPrimeProd}([n], \boldsymbol{v})$ outputs a_j and b_j.

[42] As pointed out in [39], although for the interactive version of such protocols the prime can be of size λ, the non-interactive version requires at least a double-sized prime 2λ, as an explicit square root attack was presented. Notably, even in the interactive version a $2^{\lambda/2}$-attacker would still be able to succeed in breaking knowledge-soundness with $2^{-\lambda/2}$ probability, with a λ-sized prime.

- $\mathsf{PrimeProd}(I) \to u_I$: given a set of indices I, this function outputs the product of all primes corresponding to indices in I. Namely, it returns $u_I \coloneqq \prod_{i \in I} p_i$. In the special case $I = [n]$, we denote the output of $\mathsf{PrimeProd}([n])$ as u_n.

  Notice that by construction, for any I and $\boldsymbol{y}$, it always holds $a_{I,j} \cdot b_{I,j} = u_I$.

SVC Scheme. Below we describe our SVC scheme and then we show its incremental aggregation.

$\mathsf{Setup}(1^\lambda, \{0,1\}^k) \to \mathsf{pp}$ generates a hidden order group $\mathbb{G} \leftarrow \mathsf{Ggen}(1^\lambda)$ and samples three generators $g, g_0, g_1 \leftarrow \mathbb{G}$. It also determines a deterministic collision resistant function $\mathsf{PrimeGen}$ that maps integers to primes.
Returns $\mathsf{pp} = (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen})$

$\mathsf{Specialize}(\mathsf{pp}, n) \to \mathsf{pp}_n$ computes $u_n \leftarrow \mathsf{PrimeProd}([n])$ and $U_n = g^{u_n}$, and returns $\mathsf{pp}_n \leftarrow (\mathsf{pp}, U_n)$. One can think of U_n as an accumulator to the set $[n]$.

$\mathsf{Com}^\star(\mathsf{pp}_n, \boldsymbol{v}) \to (C^\star, \mathsf{aux}^\star)$ does the following:

1. Compute $((a_1, b_1), \ldots, (a_k, b_k)) \leftarrow \mathsf{PartndPrimeProd}([n], \boldsymbol{v})$; next,

   $$\text{for all } j \in [k] \text{ compute } A_j = g_0^{a_j} \text{ and } B_j = g_1^{b_j}$$

   One can think of each (A_j, B_j) as a pair of RSA accumulators for two sets that constitute a partition of $[n]$ done according to the bits of $v_{1j}, \ldots, v_{nj}$. Namely A_j and B_j accumulate the sets $\{i \in [n] : v_{ij} = 0\}$ and $\{i \in [n] : v_{ij} = 1\}$ respectively.
2. For all $j \in [k]$, compute $C_j = A_j \cdot B_j \in \mathbb{G}$ and a proof $\pi^{(j)}_{\mathsf{prod}} \leftarrow \mathsf{PoProd}_2.\mathsf{P}(\mathsf{pp}, (C_j, U_n), (a_j, b_j))$. Such proof ensures that the sets represented by A_j and B_j are a partition of the set represented by U_n. Since U_n is part of the CRS (i.e., it is trusted), this ensures the well-formedness of A_j and B_j.

Return $C^\star := \left(\{A_1, B_1, \ldots, A_k, B_k\}, \left\{\pi^{(1)}_{\mathsf{prod}}, ..., \pi^{(k)}_{\mathsf{prod}}\right\}\right)$ and $\mathsf{aux}^\star := \boldsymbol{v}$.

$\mathsf{Open}^\star(\mathsf{pp}_n, I, \boldsymbol{y}, \mathsf{aux}^\star) \to \pi_I$ proceeds as follows:

- let $J = [n] \setminus I$ and compute $((a_{J,1}, b_{J,1}), \ldots, (a_{J,k}, b_{J,k})) \leftarrow \mathsf{PartndPrimeProd}(J, \boldsymbol{v}_J)$;
- for all $j \in [k]$ compute

  $$\Gamma_{I,j} := g_0^{a_{J,j}} \text{ and } \Delta_{I,j} = g_1^{b_{J,j}}$$

Notice that $a_{J,j} = a_j / a_{I,j}$ and $b_{J,j} = b_j / b_{I,j}$. Also $\Gamma_{I,j}$ is a membership witness for the set $\{i_l \in I : y_{lj} = 0\}$ in the accumulator A_j, and similarly for $\Delta_{I,j}$.

Return $\pi_I \coloneqq \{\pi_{I,1}, \ldots, \pi_{I,k}\} \leftarrow \{(\Gamma_{I,1}, \Delta_{I,1}), \ldots, (\Gamma_{I,k}, \Delta_{I,k})\}$

$\mathsf{Ver}^\star(\mathsf{pp}_n, C^\star, I, \boldsymbol{y}, \pi_I) \to b$ computes $((a_{I,1}, b_{I,1}), \ldots, (a_{I,k}, b_{I,k}))$ using $\mathsf{PartndPrimeProd}(I, \boldsymbol{y})$, and then returns $b \leftarrow b_{acc} \wedge b_{prod}$ where:

$$b_{acc} \leftarrow \bigwedge_{j=1}^{k} \left(\Gamma_{I,j}^{a_{I,j}} = A_j \wedge \Delta_{I,j}^{b_{I,j}} = B_j\right) \tag{6.1}$$

$$b_{prod} \leftarrow \bigwedge_{j=1}^{k} \left(\mathsf{PoProd}_2.\mathsf{V}(\mathsf{pp}, (A_j \cdot B_j, U_n), \pi^{(j)}_{\mathsf{prod}})\right) \tag{6.2}$$

Remark 17. *For more efficient verification,* $\mathsf{Open}^\star$ *can be changed to include* $2k$ *(non-interactive) proofs of exponentiation* PoE *(which using the PoKCR aggregation from [40] add only* k *elements of* $\mathbb{G}$*). This reduces the exponentiations cost in* $\mathsf{Ver}^\star$*. As noted in [40], although the asymptotic complexity is the same, the operations are in* $\mathbb{Z}_{2^{2\lambda}}$ *instead of* $\mathbb{G}$*, which concretely makes up an improvement.*

The correctness of the vector commitment scheme described above is obvious by inspection (assuming correctness of PoProd_2).

Incremental Aggregation. Here we show that our SVC scheme is incrementally aggregatable.

$\mathsf{VC.Disagg}(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, K) \to \pi_K$**.** Let $L := I \setminus K$, and $\boldsymbol{v}_L$ be the subvector of $\boldsymbol{v}_I$ at positions in L. Then compute $\{a_{L,j}, b_{L,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(L, \boldsymbol{v}_L)$, and for each $j \in [k]$ set:

$$\Gamma_{K,j} \leftarrow \Gamma_{I,j}^{a_{L,j}}, \quad \Delta_{K,j} \leftarrow \Delta_{I,j}^{b_{L,j}}$$

and return $\pi_K := \{\pi_{K,1}, \ldots, \pi_{K,k}\} := \{(\Gamma_{K,1}, \Delta_{K,1}), \ldots, (\Gamma_{K,k}, \Delta_{K,k})\}$

$\mathsf{VC.Agg}(\mathsf{pp}, (I, \boldsymbol{v}_I, \pi_I), (J, \boldsymbol{v}_J, \pi_J)) \to \pi_K := \{(\Gamma_{K,1}, \Delta_{K,1}), \ldots, (\Gamma_{K,k}, \Delta_{K,k})\}$**.**

1. Let $L := I \cap J$. If $L \neq \emptyset$, set $I' := I \setminus L$ and compute $\pi_{I'} \leftarrow \mathsf{VC.Disagg}(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, I')$; otherwise let $\pi_{I'} = \pi_I$.
2. Compute $\{a_{I',j}, b_{I',j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I', \boldsymbol{v}_{I'})$ and $\{a_{J,j}, b_{J,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(J, \boldsymbol{v}_J)$.
3. Parse $\pi_{I'} := \{(\Gamma_{I',j}, \Delta_{I',j})\}_{j=1}^k$, $\pi_J := \{(\Gamma_{J,j}, \Delta_{J,j})\}_{j=1}^k$, and for all $j \in [k]$, compute

$$\Gamma_{K,j} \leftarrow \mathbf{ShamirTrick}(\Gamma_{I',j}, \Gamma_{J,j}, a_{I',j}, a_{J,j}),$$
$$\Delta_{Kj} \leftarrow \mathbf{ShamirTrick}(\Delta_{I',j}, \Delta_{J,j}, b_{I',j}, b_{J,j}).$$

Note that our algorithms above can work directly with the universal CRS pp, and do not need the specialized one pp_n.

Aggregation Correctness. The second property of aggregation correctness (the one about VC.Disagg) is straightforward by construction:
if we let $\{a_{K,j}, b_{K,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(K, \boldsymbol{v}_K)$, then $a_{I,j} = a_{L,j} \cdot a_{K,j}$, and thus $A_j = \Gamma_{I,j}^{a_{I,j}} = \Gamma_{I,j}^{a_{L,j}\cdot a_{K,j}} = \Gamma_{K,j}^{a_{K,j}}$ (and similarly for $\Delta_{K,j}$).

The first property instead follows from the correctness of Shamir's trick if the integer values provided as input are coprime; however since $I' \cap J = \emptyset$, $a_{I',j}$ and $a_{J,j}$ (resp. $b_{I',j}$ and $b_{J,j}$) are coprime unless a collision occurs in PrimeGen.

Efficiency. We summarize the efficiency of our construction in terms of both the computational cost of the algorithms and the communication (CRS, commitment and openings size). For this analysis we consider an instantiation of PrimeGen with a deterministic function that maps every integer in $[n]$ into a unique prime number, which can be of $\alpha = \log n$ bits.

Our scheme is presented in order to support vectors of length n of k-bits-long strings. We summarize efficiency in terms of k and n. However, we note that k is actually only a parameter and our scheme can work with any setting of vectors $\boldsymbol{v}$ of length N of ℓ-bits long strings. In this case, it is sufficient to fix an arbitrary k that divides ℓ and to spread each $v_i \in \{0,1\}^\ell$ over ℓ/k positions. For example, for $k = 1$ with have $n = N\ell$ and thus the prime size is $\alpha = \log(N\ell)$.

Setup. In terms of computation, Setup generates the group description and samples 3 generators, while Specialize computes one exponentiation in $\mathbb{G}$ with an $(n\alpha)$-long integer. The CRS consists of 3 elements of $\mathbb{G}$, and the specialized CRS (for any n) is one group element.

Committing. Committing to a vector $\boldsymbol{v} \in (\{0,1\}^k)^n$ requires about k exponentiations with an $(n\alpha)$-long integer each. A commitment consists of $4k$ elements of $\mathbb{G}$ and $2k$ integers in $\mathbb{Z}_{2^{2\lambda}}$.

Opening. Creating an opening for a set I of m positions has about the same cost of committing, and the opening consists of $2k$ group elements. Using the PoE to make verification more efficient (see Remark 17) would (naively) result to $4k$ elements. However, as described in [40], many PoE's for coprime exponents can be aggregated into a single group element. In our case, applying this optimization would result to k group elements for all the PoE's, which totally gives $3k$ group elements for an opening.

Verification. Verifying an opening for set I requires about k exponentiations with $(m \cdot \alpha)$-bit integers (resp. $4k$ exponentiations with λ-bit integers, $2k$ multiplications in $\mathbb{G}$ and $O(km\alpha)$ multiplications in $\mathbb{Z}_{2^{2\lambda}}$, when using PoE) to check equation (6.1), plus $5k$ exponentiations with 2λ-bit integers and $3k$ multiplications in $\mathbb{G}$ to verify PoProd_2 proofs in equation (6.2).

Aggregation and Disaggregation. Disaggregation requires $2k$ exponentiations with $((|I| - |K|)\alpha)$-bit integers, while aggregation requires $2k$ computations of $\mathbf{ShamirTrick}$ that amount to $O(k(|I| + |J|)\alpha)$ operations in $\mathbb{G}$. From this, we obtain that VC.AggManyToOne and VC.DisaggOneToMany take time $O(ksm \log m\alpha)$ $\mathbb{G}$ and $O(km \log(m/B)\alpha)$ $\mathbb{G}$, respectively.

Commitment and Opening with Precomputation. Finally, let us summarize the costs of committing and opening with preprocessing obtained by instantiating our method of Section 6.4.2. The preprocessing VC.PPCom takes time $O(kn\alpha \log(n/B))$. The opening requires computing at most $|S| \leq m$ disaggregation, each taking time $O(k\alpha(|P_j| - |I_j|))$, for a total of $O(k\alpha(|S|B - |I|))$, followed by the aggregation step that counts $O(k\alpha|S| \log |S|)$. So, in the worst case VC.FastOpen takes $O(k \cdot m \cdot \alpha(\log(m) + B - 1))$ operations of $\mathbb{G}$.

Security. The security of our SVC scheme, i.e., position binding, can be reduced to the Strong RSA and Adaptive root assumptions in the hidden order group $\mathbb{G}$ used in the construction and to the knowledge extractability of PoProd_2.

A bit more in detail the steps of the proof are as follows. Let an adversary to the position binding output $(C, I, \boldsymbol{y}, \pi, \boldsymbol{y}', \pi')$. First from knowledge extractability of PoProd_2 it comes that $A_j B_j = g_1^{a_j} g_2^{b_j}$ and $g^{a_j b_j} = U_n = g^{u_n}$. However, this does not necessarily means that $a_j b_j = u_n$ over the integers and to prove it we need the Low Order assumptions, under which it holds. Afterwards we prove that since $A_j B_j = g_1^{a_j} g_2^{b_j}$ no different proofs π, π' for the same positions can pass the verification under the strong RSA assumption, which is the core of our proof. The main caveat of the proof is that instead of knowing that $A_j = g_1^{a_j}$ and $B_j = g_2^{b_j}$ we know only that $A_j B_j = g_1^{a_j} g_2^{b_j}$. The former case would directly reduce to RSA Accumulator's security (strong RSA assumption). For this we first need to prove an intermediate lemma (lemma 14) which shows that specifically for our case $A_j B_j = g_1^{a_j} g_2^{b_j}$ is enough, since the choice of the primes p_i in the exponent is restricted to a polynomially bounded set.

Theorem 21 (Position-Binding)**.** *Let* Ggen *be the generator of hidden order groups where the Strong RSA and Low Order assumptions hold, and let* PoProd_2 *be an argument of knowledge for* R_{PoProd_2}*. Then the subVector Commitment scheme defined above is position binding.*

Proof. To prove the theorem we use a hybrid argument. We start by defining the game G_0 as the actual position binding game of Definition 12, and our goal is to prove that for any PPT $\mathcal{A}$, $\Pr[G_0 = 1] \in \mathsf{negl}(\lambda)$.

Game G_0:

$G_0 = \mathsf{PosBind}^{\mathcal{A}}_{\mathsf{VC}}(\lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$

$(C, I, \boldsymbol{y}, \pi, \boldsymbol{y}', \pi') \leftarrow \mathcal{A}(\mathsf{pp})$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}, \pi) = 1 \wedge \boldsymbol{y} \neq \boldsymbol{y}' \wedge \mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}', \pi') = 1$

return b

Lemma 10. *For any PPT $\mathcal{A}$ in game G_0 there exists an algorithm $\mathcal{E}$ and an experiment G_1 such that*

$$\Pr[G_0 = 1] \leq \Pr[G_1 = 1] + \mathsf{negl}(\lambda)$$

Proof. By construction of Com, the commitment C returned by the adversary $\mathcal{A}$ in game G_0 contains k proofs of PoProd_2, and by construction of Ver if G_0 returns 1 all these proofs verify. It is not hard to argue that for any adversary $\mathcal{A}$ playing in game G_0 there is an extractor $\mathcal{E}$ that outputs the k witnesses $\{a_j, b_j\}_{j\in[k]}$.

Game G_1: is the same as G_0 except that we also execute $\mathcal{E}$, which outputs $\{a_j, b_j\}_{j\in[k]}$, and we additionally check that $U_n = g^{a_j b_j}$ for all $j \in [k]$. Below is a detailed description of G_1 in which we "open the box" of the VC algorithms.

G_1

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M}); \mathsf{bad}_1 \leftarrow \mathsf{false}$

$(\{A_j, B_j, \pi^{(j)}_{\mathsf{prod}}\}_{j\in[k]}, n), I, \boldsymbol{y}, \{\Gamma_{I,j}, \Delta_{I,j}\}_{j\in[k]}, \boldsymbol{y}', \{\Gamma'_{I,j}, \Delta'_{I,j}\}_{j\in[k]}) \leftarrow \mathcal{A}(\mathsf{pp})$

$\{a_j, b_j\}_{j\in[k]} \leftarrow \mathcal{E}(\mathsf{pp})$

$u_n \leftarrow \mathsf{PrimeProd}(n); U_n \leftarrow g^{u_n}$

$b_{prod} \leftarrow \bigwedge_{j=1}^{k} \left(\mathsf{PoProd}_2.\mathsf{V}(\mathsf{pp}, (A_j \cdot B_j, U_n), \pi^{(j)}_{\mathsf{prod}})\right)$

$b_{wit} \leftarrow \bigwedge_{j=1}^{k} A_j \cdot B_j = g_0^{a_j} g_j^{b_j} \wedge U_n = g^{a_j \cdot b_j}$

if $b_{prod} = 1 \wedge b_{wit} = 0$ **then** $\mathsf{bad}_1 \leftarrow \mathsf{true}$

$\{a_{I,j}, b_{I,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y}); \ \{a'_{I,j}, b'_{I,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y}')$

$$b \leftarrow b_{prod} \wedge \bigwedge_{j=1}^{k} \left({\Gamma_{I,j}}^{a_{I,j}} = A_j \wedge {\Delta_{I,j}}^{b_{I,j}} = B_j\right) \wedge \boldsymbol{y} \neq \boldsymbol{y}' \wedge \bigwedge_{j=1}^{k} \left({\Gamma'_{I,j}}^{a'_{I,j}} = A_j \wedge {\Delta'_{I,j}}^{b'_{I,j}} = B_j\right)$$

if $\mathsf{bad}_1 = \mathsf{true}$ **then** $b \leftarrow 0$

return b

Clearly, the games G_0 and G_1 are identical except if the flag bad_1 is raised true, i.e., $\Pr[G_0 = 1] - \Pr[G_1 = 1] \leq \Pr[\mathsf{bad}_1 = \mathsf{true}]$. However, the event in which bad_1 is set true is

the event in which one of the witnesses returned by the extractor is not correct. By the knowledge extractability of PoProd_2 we immediately get that $\Pr[\mathsf{bad}_1 = \mathsf{true}] \in \mathsf{negl}(\lambda)$. □

Game G_2: is the same as G_1 except that G_2 outputs 0 if there is an index j such that $U_n = g^{a_j \cdot b_j}$ but $u_n \neq a_j \cdot b_j$. Precisely, if this happens a flag bad_2 is set true and the outcome of the experiment is 0. See below for the detailed description of G_2.

G_2

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M}); \mathsf{bad}_1, \mathsf{bad}_2 \leftarrow \mathsf{false}$

$(\{A_j, B_j, \pi^{(j)}_{\mathsf{prod}}\}_{j\in[k]}, n), I, \boldsymbol{y}, \{\Gamma_{I,j}, \Delta_{I,j}\}_{j\in[k]}, \boldsymbol{y}', \{\Gamma'_{I,j}, \Delta'_{I,j}\}_{j\in[k]}) \leftarrow \mathcal{A}(\mathsf{pp})$

$\{a_j, b_j\}_{j\in[k]} \leftarrow \mathcal{E}(\mathsf{pp})$

$u_n \leftarrow \mathsf{PrimeProd}(n); U_n \leftarrow g^{u_n}$

$$b_{prod} \leftarrow \bigwedge_{j=1}^{k} \left(\mathsf{PoProd}_2.\mathsf{V}(\mathsf{pp}, (A_j \cdot B_j, U_n), \pi^{(j)}_{\mathsf{prod}})\right)$$

$$b_{wit} \leftarrow \bigwedge_{j=1}^{k} A_j \cdot B_j = g_0^{a_j} g_j^{b_j} \wedge U_n = g^{a_j \cdot b_j}$$

if $b_{prod} = 1 \wedge b_{wit} = 0$ **then** $\mathsf{bad}_1 \leftarrow \mathsf{true}$

$$b_{col} \leftarrow \bigwedge_{j=1}^{k} u_n = a_j \cdot b_j$$

if $b_{prod} = 1 \wedge b_{col} = 0$ **then** $\mathsf{bad}_2 \leftarrow \mathsf{true}$

$\{a_{I,j}, b_{I,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y}); \ \{a'_{I,j}, b'_{I,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y}')$

$$b \leftarrow b_{prod} \wedge \bigwedge_{j=1}^{k} \left({\Gamma_{I,j}}^{a_{I,j}} = A_j \wedge {\Delta_{I,j}}^{b_{I,j}} = B_j\right) \wedge \boldsymbol{y} \neq \boldsymbol{y}' \wedge \bigwedge_{j=1}^{k} \left({\Gamma'_{I,j}}^{a'_{I,j}} = A_j \wedge {\Delta'_{Ij}}^{b'_{I,j}} = B_j\right)$$

if $\mathsf{bad}_1 = \mathsf{true} \vee \mathsf{bad}_2 = \mathsf{true}$ **then** $b \leftarrow 0$

return b

Lemma 11. *If the Low Order assumption holds for* Ggen*, then* $\Pr[G_1 = 1] - \Pr[G_2 = 1] \leq \mathsf{negl}(\lambda)$.

Proof. Clearly, G_1 and G_2 proceed identically except if bad_2 is set true. We claim that $\Pr[\mathsf{bad}_2 = \mathsf{true}]$ is negligible for any $\mathcal{A}, \mathcal{E}$ running in G_2. If this event happens, one indeed obtains an integer $v = u_n - a_j \cdot b_j$ such that $g^v = 1 \in \mathbb{G}$, where $g \neq 1$ and $1 < v < 2^{\mathsf{poly}(\lambda)}$, and solves the Low Order problem. □

Game G_3**:** is an experiment that can be seen as a simplification of G_2.

G_3

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$

$(\boldsymbol{v}, \{A_j, B_j\}_{j\in[k]}, I, \boldsymbol{y}, \{\Gamma_{I,j}, \Delta_{I,j}\}_{j\in[k]}, \boldsymbol{y}', \{\Gamma'_{I,j}, \Delta'_{I,j}\}_{j\in[k]}) \leftarrow \mathcal{A}'(\mathsf{pp})$

$\{a_j, b_j\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}([n], \boldsymbol{v})$

$\{a_{I,j}, b_{I,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y})$; $\{a'_{I,j}, b'_{I,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y}')$

$$b \leftarrow \bigwedge_{j=1}^{k} (A_j \cdot B_j = g_0^{a_j} \cdot g_1^{b_j}) \bigwedge_{j=1}^{k} \left(\Gamma_{I,j}{}^{a_{I,j}} = A_j \wedge \Delta_{I,j}{}^{b_{I,j}} = B_j\right) \wedge \boldsymbol{y} \neq \boldsymbol{y}' \wedge$$
$$\bigwedge_{j=1}^{k} \left(\Gamma'_{I,j}{}^{a'_{I,j}} = A_j \wedge \Delta'_{I,j}{}^{b'_{I,j}} = B_j\right)$$

return b

First, we show the following lemma that relates the probability of winning in G_3 with that of winning in G_2.

Lemma 12. *For any $(\mathcal{A}, \mathcal{E})$ running in G_2 there is an $\mathcal{A}'$ running in G_3 such that* $\Pr[G_2 = 1] = \Pr[G_3 = 1]$.

Proof. We build $\mathcal{A}'$ from $(\mathcal{A}, \mathcal{E})$ as follows. On input pp, $\mathcal{A}'$ executes $(\{A_j, B_j, \pi^{(j)}_{\mathsf{prod}}\}_{j\in[k]}, n), I, \boldsymbol{y}, \{\Gamma_{I,j}, \Delta_{I,j}\}_{j\in[k]}, \boldsymbol{y}', \{\Gamma'_{I,j}, \Delta'_{I,j}\}_{j\in[k]}) \leftarrow \mathcal{A}(\mathsf{pp})$ and $\{a_j, b_j\}_{j\in[k]} \leftarrow \mathcal{E}(\mathsf{pp})$. Next, $\mathcal{A}'$ reconstructs a vector $\boldsymbol{v} \in (\{0,1\}^k)^n$ from the set $\{a_j, b_j\}_{j\in[k]}$. This can be done by setting $v_{ij} = 0$ if $p_i \mid a_j$ and $v_{ij} = 1$ if $p_i \mid b_j$, where $p_i \leftarrow \mathsf{PrimeGen}(i)$ (in case both or neither cases occur, abort). Finally, $\mathcal{A}'$ runs all the checks as in game G_2, and if G_2 would output 1, then $\mathcal{A}'$ outputs $(\boldsymbol{v}, \{A_j, B_j\}_{j\in[k]}, I, \boldsymbol{y}, \{\Gamma_{I,j}, \Delta_{I,j}\}_{j\in[k]}, \boldsymbol{y}', \{\Gamma'_{I,j}, \Delta'_{I,j}\}_{j\in[k]})$, otherwise $\mathcal{A}'$ aborts.

To claim that $\Pr[G_2 = 1] = \Pr[G_3 = 1]$, we observe that whenever G_2 returns 1 it is the case that $a_j \cdot b_j = u_n = \prod_{i=1}^{n} p_i$ for all $j \in [k]$; therefore $\mathcal{A}'$ never aborts. □

Game G_4: this is the same as game G_3 except that the game outputs 0 if during any computation of lines 3 and 4 it happens that $\mathsf{PrimeGen}(i) = \mathsf{PrimeGen}(i')$ for distinct $i \neq i'$. It is straightforward to show that the probability of this event is bounded by the probability of finding collisions in PrimeGen, i.e., that under the collision resistance of PrimeGen it holds $\Pr[G_3 = 1] - \Pr[G_4 = 1] \in \mathsf{negl}(\lambda)$.

To conclude the proof of our Theorem, we prove that any PPT adversary can win in G_4 with only negligible probability assuming that the strong RSA assumption holds in $\mathbb{G}$.

Lemma 13. *If the* strong RSA assumption *holds for* Ggen*, then for every PPT adversary $\mathcal{A}'$ running in game G_4 we have that $Pr[G_4 = 1] \in \mathsf{negl}(\lambda)$.*

Proof. For the proof, we rely on the following lemma that defines a computational problem that we prove it is implied by the Strong RSA assumption.

Lemma 14. *Let* Ggen *be a hidden order group generation algorithm where the* strong RSA assumption *holds and* PrimeGen *a deterministic collision resistant function that maps integers to*

primes. Then for any PPT adversary $\mathcal{A}$ and any $n = \mathsf{poly}(\lambda)$, the probability below is negligible:

$$\Pr\left[\begin{array}{c} u^p = g_0^a \cdot g_1^b \\ \wedge (p \nmid a \vee p \nmid b) \\ \wedge u \in \mathbb{G} \wedge (a,b) \in \mathbb{Z}^2 \wedge p \in S \end{array} : \begin{array}{l} \mathbb{G} \leftarrow \mathsf{Ggen}(\lambda) \\ g_0, g_1 \leftarrow \mathbb{G} \\ S = \{p_i \leftarrow \mathsf{PrimeGen}(i)\}_{i=1}^n \\ (u,p,a,b) \leftarrow \mathcal{A}(\mathbb{G}, g_0, g_1, S) \end{array}\right] \in \mathsf{negl}(\lambda)$$

We proceed assuming that the lemma holds; its proof is deferred to the end.

Suppose by contradiction the existence of a PPT adversary $\mathcal{A}'$ such that $\Pr[G_4 = 1] = \epsilon$ with ϵ non-negligible. Below we show how to construct an adversary $\mathcal{B}$ that uses $\mathcal{A}'$ in order to solve the problem of Lemma 14 with probability ϵ.

- $\mathcal{B}(\mathbb{G}, g_0, g_1)$ samples a random $g \leftarrow_\$ \mathbb{G}$, determines a $\mathsf{PrimeGen}$ as in Setup, sets $\mathsf{pp} \leftarrow (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen})$, and runs $\mathcal{A}$ on input pp.
- $\mathcal{A}(\mathsf{pp})$ responds with a tuple $(\boldsymbol{v}, \{A_j, B_j\}_{j\in[k]}, I, \boldsymbol{y}, \pi, \boldsymbol{y}', \pi')$.
- $\mathcal{B}$ computes $\{a_j, b_j\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}([n], \boldsymbol{v})$, $\{a_{I,j}, b_{I,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y})$ and $\{a'_{I,j}, b'_{I,j}\}_{j\in[k]} \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y}')$ as in game G_3.
- If $\mathcal{A}'$ wins the game then we have that all the following conditions holds:

$$\boldsymbol{y} \neq \boldsymbol{y}', \bigwedge_{j=1}^{k} \left(\Gamma_{I,j}{}^{a_{I,j}} = A_j \wedge \Delta_{I,j}{}^{b_{I,j}} = B_j\right) = 1, \bigwedge_{j=1}^{k} \left(\Gamma'_{I,j}{}^{a'_{I,j}} = A_j \wedge \Delta'_{I,j}{}^{b'_{I,j}} = B_j\right) = 1$$

,

$$\bigwedge_{j=1}^{k} (A_j \cdot B_j = g_0^{a_j} \cdot g_1^{b_j}).$$

From $\boldsymbol{y} \neq \boldsymbol{y}'$ we get that there is at least one pair of indices $l \in [m]$ and $j \in [k]$ such that $y_{lj} \neq y'_{lj}$. Say wlog that $y_{lj} = 0$ and $y'_{lj} = 1$. Also, if we parse $I = \{i_1, \ldots, i_m\}$, we let $i = i_l \in [m]$. So we fix these indices i and j, and let $p_i = \mathsf{PrimeGen}(i)$ be the corresponding prime.

Notice that by construction of $\mathsf{PartndPrimeProd}$ (and since we assumed no collision occurs in $\mathsf{PrimeGen}$) we have that either $p_i \nmid a_j$ or $p_i \nmid b_j$ holds. Additionally, by our assumption that $y_{lj} = 0$ and $y'_{lj} = 1$, the following holds: $p_i \mid a_{I,j}$, $p_i \nmid b_{I,j}$, $p_i \nmid a'_{I,j}$, $p_i \mid b'_{I,j}$.

From the other condition on the validity of the proofs, $\mathcal{B}$ can compute two group elements $\hat{\Gamma}, \hat{\Delta}$ such that $\hat{\Gamma}^{p_i} = A_j$ and $\hat{\Delta}^{p_i} = B_j$.

Combining this with the condition $A_j \cdot B_j = g_0^{a_j} \cdot g_1^{b_j}$, we have that $(\hat{\Gamma} \cdot \hat{\Delta})^{p_i} = g_0^{a_j} \cdot g_1^{b_j}$.

- $\mathcal{B}$ sets $w = \hat{\Gamma} \cdot \hat{\Delta}$ and outputs the tuple (w, p_i, a_j, b_j).

From all the above observations, if $\mathcal{A}'$ makes game G_4 return 1, then the tuple returned by $\mathcal{B}$ is a suitable solution for the problem of Lemma 14, which in turn reduces to the Strong RSA assumption. □

By combining all the lemmas we have that any PPT adversary has at most negligible probability of breaking the position binding of our SVC scheme. □

Proof of Lemma 14. Suppose that for a PPT adversary $\mathcal{A}$ the above probability is a non-negligible value ϵ. We will construct an adversary $\mathcal{B}$ that breaks strong RSA assumption with a non-negligible probability. $\mathcal{B}$ takes as input $(\mathbb{G}, g)$. We denote as $G_{\mathcal{A}}$ the game defined in lemma (parametrized by an adversary $\mathcal{A}$). We define two different reductions:

Reduction 1. In reduction 1 the adversary $\mathcal{B}$ breaks strong RSA assumption only in case where the adversary $\mathcal{A}$ outputs a tuple (u, p, a, b) such that $p \mid a$ (and thus from assumption $p \nmid b$) and fails otherwise. $\mathcal{B}$ proceeds as follows.

$\mathcal{B}(\mathbb{G}, g)$ samples $\gamma \leftarrow_{\$} [1, 2^{\lambda}\mathsf{maxord}(\mathbb{G})]$, where $\mathsf{maxord}(\mathbb{G})$ is the upper bound of the order of $\mathbb{G}$ outputted by $\mathsf{Ggen}(1^{\lambda})$ (see Section 3.3), and sets $g_0 \leftarrow g^{\gamma}, g_1 \leftarrow g$. $\mathcal{B}$ runs $\mathcal{A}$ on input $(\mathbb{G}, g_0, g_1)$. γ is sampled from a large enough domain so that g^{γ} is statistically close to a uniformly distributed g_0 from $\mathbb{G}$ hence g_0, g_1 are indistinguishable to two uniformly random elements of $\mathbb{G}$. $\mathcal{A}(\mathbb{G}, g_0, g_1, S)$ responds with a tuple (u, p, a, b) and sends it to $\mathcal{B}$. We condition our analysis on the event $p \mid a$, meaning that $\mathcal{B}$ stops in case $p \nmid a$.

Assume that $u^p = g_0^a \cdot g_1^b \wedge (p \mid a \wedge p \nmid b) \wedge u \in \mathbb{G} \wedge (a, b) \in \mathbb{Z}^2 \wedge p \in S$ then we will show that $\mathcal{B}$ can break the strong RSA assumption. We argue that $p \mid a$ leads to $\gcd(p, \gamma a + b) = 1$. Let $\gcd(p, \gamma a + b) \neq 1$, meaning that $\gcd(p, \gamma a + b) = p$, then $p \mid \gamma a + b \Rightarrow \gamma a + b = 0 \pmod p$. However, $p \mid a \Rightarrow a = 0 \pmod p$. From the two previous facts we infer that $b = 0 \pmod p \Rightarrow p \mid b$, hence $p \mid a \wedge p \mid b$, which is a contradiction. Therefore, assuming that $\gcd(p, \gamma a + b) = 1$, $\mathcal{B}$ uses the extended Euclidean algorithm to compute (α, β) such that $\alpha p + \beta(a\gamma + b) = 1$. We know that $u^p = g_0^a g_1^b = g^{a\gamma + b} \Rightarrow u = g^{\frac{a\gamma+b}{p}}$ hence it follows that $g^{1/p} = g^{\frac{\alpha p + \beta(a\gamma+b)=1}{p}} = g^{\alpha + \beta\frac{a\gamma+b}{p}} = g^{\alpha} \cdot u^{\beta}$. Finally, $\mathcal{B}$ outputs $(g^{\alpha} \cdot u^{\beta}, p)$ which is a valid strong-RSA solution.

Reduction 2. In reduction 2 the adversary $\mathcal{B}$ breaks strong RSA assumption only in case where the adversary $\mathcal{A}$ outputs a tuple (u, p, a, b) such that $p \nmid a$ and fails otherwise.

$\mathcal{B}(\mathbb{G}, g)$ samples $\gamma \leftarrow_{\$} [1, 2^{\lambda}\mathsf{maxord}(\mathbb{G})]$, where $\mathsf{maxord}(\mathbb{G})$ is the upper bound of the order of $\mathbb{G}$ outputted by $\mathsf{Ggen}(1^{\lambda})$ (see Section 3.3), defines $S := \{p_i \leftarrow \mathsf{PrimeGen}(i)\}_{i=1}^{n}$ and $\mathsf{prod} \leftarrow \prod_{i=1}^{n} p_i$ and sets $g_0 \leftarrow g, g_1 \leftarrow g^{\gamma \cdot \mathsf{prod}}$. $\mathcal{B}$ sends $(\mathbb{G}, g_0, g_1)$ to $\mathcal{A}$. γ is sampled from a large enough domain so that g^{γ} is statistically close to a uniformly distributed g_1 from $\mathbb{G}$ hence g_0, g_1 are indistinguishable to two uniformly random elements of $\mathbb{G}$. $\mathcal{A}(\mathbb{G}, g_0, g_1, S)$ responds with a tuple (u, p, a, b) and sends it to $\mathcal{B}$. We condition our analysis on the event $p \nmid a$, meaning that $\mathcal{B}$ stops in case $p \mid a$.

Assume that $u^p = g_0^a \cdot g_1^b \wedge p \nmid a \wedge u \in \mathbb{G} \wedge (a, b) \in \mathbb{Z}^2 \wedge p \in S$ then we will show that $\mathcal{B}$ can break the strong RSA assumption. We argue that $\gcd(p, a + b\gamma\mathsf{prod}) = 1$. Let $\gcd(p, a + b\gamma\mathsf{prod}) \neq 1$, meaning that $\gcd(p, a + b\gamma\mathsf{prod}) = p$, then $p \mid a + b\gamma\mathsf{prod} \Rightarrow a + b\gamma\mathsf{prod} = 0 \pmod p$. However, prod includes p ($p \in S$) we know that $p \mid b\gamma\mathsf{prod} \Rightarrow b\gamma\mathsf{prod} = 0 \pmod p$. From the two previous facts we infer that $a = 0 \pmod p \Rightarrow p \mid a$ which is a contradiction. $\mathcal{B}$ uses the extended Euclidean algorithm to compute (α, β) such that $\alpha p + \beta(a + b\gamma\mathsf{prod}) = 1$. We know that $u^p = g_0^a g_1^b = g^{a + b\gamma\mathsf{prod}} \Rightarrow u = g^{\frac{a+b\gamma\mathsf{prod}}{p}}$ hence it follows that $g^{1/p} = g^{\frac{\alpha p + \beta(a+b\gamma\mathsf{prod})=1}{p}} = g^{\alpha + \beta\frac{a+b\gamma\mathsf{prod}}{p}} = g^{\alpha} \cdot u^{\beta}$. Finally, $\mathcal{B}$ outputs $(g^{\alpha} \cdot u^{\beta}, p)$ which is a valid strong-RSA solution.

To conclude the proof, notice that:

$$\begin{aligned}\Pr[G_{\mathcal{A}}=1] &= \Pr[G_{\mathcal{A}}=1|p \mid a]\Pr[p \mid a] + \Pr[G_{\mathcal{A}}=1|p \nmid a]\Pr[p \nmid a] \\ &\leq \Pr[G_{\mathcal{A}}=1|p \mid a] + \Pr[G_{\mathcal{A}}=1|p \nmid a]\end{aligned}$$

The reductions 1 and 2 described above show that under the strong RSA assumption $\Pr[G_{\mathcal{A}}=1|p \mid a]$ and $\Pr[G_{\mathcal{A}}=1|p \nmid a]$ respectively are negligible. Hence, we have that $\Pr[G_{\mathcal{A}}=1] \in \mathsf{negl}(\lambda)$, which concludes the proof. □

On concrete instantiation. Our SVC construction is described generically from a hidden order group $\mathbb{G}$, an AoK PoProd_2, and a mapping to primes PrimeGen. The concrete scheme we analyze is the one where PoProd_2 is instantiated with the non-interactive version of the PoProd_2 protocol described in Sec. 6.5.1.1. The non-interactive version needs a hash-to-prime function Hprime. We note that the same function can be used to instantiate PrimeGen, though for the sake of PrimeGen we do not need its randomness properties. One can choose a different mapping to primes for PrimeGen and even just a bijective mapping (which is inherently collision resistant) would be enough: this is actually the instantiation we consider in our efficiency analysis. Finally, see Section 3.3 for a discussion on possible instantiations of $\mathbb{G}$.

We note that by using the specific PoProd_2 protocol given in Sec. 6.5.1.1 we are assuming adversaries that are generic with respect to the group $\mathbb{G}$. Therefore, our SVC is ultimately position binding in the generic group model.

6.5.2 Our Second SVC Construction

In this section we propose another SVC scheme with constant-size parameters and incremental aggregation. This scheme builds on the SVC of [145] based on the RSA assumption, which in turn extends the VC of [66] to support subvector openings. Our technical contribution is twofold. First, we show that the SVC of [66, 145] can be modified in order to have public parameters and verification time independent of the vector's length. Second, we propose new algorithms for (incremental) aggregation and disaggregation for this SVC.

Our second SVC Construction. Let us start by giving a brief overview of the [66] VC scheme and of the basic idea to turn it into one with succinct parameters and verification time. In brief, in [66] a commitment to a vector $\boldsymbol{v}$ is $C = S_1^{v_1} \cdots S_n^{v_n}$, where each $S_i := g^{\prod_{j\in[n]\setminus\{i\}} e_j}$ with $g \in \mathbb{G}$ a random generator and e_j being distinct prime numbers (which can be deterministically generated using a suitable map-to-primes). The opening for position i is an element Λ_i such that $\Lambda_i^{e_i} \cdot S_i^{v_i} = C$ and the key idea is that such Λ_i is an e_i-th root that can be publicly computed as long as one does it for the correct position i and value v_i. Also, as it can be seen, the element S_i is necessary to verify an opening of position i, and thus $(S_1, \ldots, S_n)$ were included in the public parameters. Catalano and Fiore observed that it might be possible to remove the S_i-s from pp if the verifier opts for recomputing S_i at verification time *at the price of linear-time verification*.

Our goal is to obtain constant-size parameters *and* constant-time verification. To do that we let the prover compute S_i and include it in the opening for position i. To prevent adversaries from providing false S_i's, we store in the public parameters $U_n = g^{\prod_{i\in[n]} e_i}$ (i.e., an accumulator to all positions) so that the verifier can verify the correctness of S_i in constant-time by checking $S_i^{e_i} = U_n$. This technique easily generalizes to subvector openings.

In the following, we describe the scheme in details and then propose our incremental aggregation algorithms. To simplify our exposition, we use the following notation: for a set of indices $I \subseteq [n]$, $e_I := \prod_{i \in I} e_i$ denotes the product of all primes corresponding to the elements of I, and $S_I := g^{\prod_{i \in [n] \setminus I} e_i} = g^{e_{[n]\setminus I}} = U_n^{1/e_I}$ (which is a generalization of the former S_i), where, we recall, the e_i's are defined from the pp.

$\mathsf{Setup}(1^\lambda, \ell, n) \to \mathsf{pp}$ generates a hidden order group $\mathbb{G} \leftarrow \mathsf{Ggen}(1^\lambda)$ and samples a generator $g \leftarrow\!\!\$\, \mathbb{G}$. It also determines a deterministic collision resistant function $\mathsf{PrimeGen}$ that maps integers to primes.
Returns $\mathsf{pp} = (\mathbb{G}, g, \mathsf{PrimeGen})$

$\mathsf{Specialize}(\mathsf{pp}, n) \to \mathsf{pp}_n$ computes n $(\ell+1)$-bit primes $e_1, \ldots, e_n$, $e_i \leftarrow \mathsf{PrimeGen}(i)$ for each $i \in [n]$, and $U_n = g^{e_{[n]}}$ and returns $\mathsf{pp}_n \leftarrow (\mathsf{pp}, U_n)$. One can think of U_n as an accumulator to the set $[n]$.

$\mathsf{Com}(\mathsf{pp}, \boldsymbol{v}) \to (C, \mathsf{aux})$ Computes for each $i \in [n]$, $S_i \leftarrow g^{e_{[n]\setminus\{i\}}}$ and then $C \leftarrow S_1^{v_1} \ldots S_n^{v_n}$ and $\mathsf{aux} \leftarrow (v_1, \ldots, v_n)$.

$\mathsf{Open}(\mathsf{pp}, I, \boldsymbol{y}, \mathsf{aux}) \to \pi_I$ Computes for each $j \in [n] \setminus I$, $S_j^{1/e_I} \leftarrow g^{e_{[n]\setminus(I\cup\{j\})}}$ and $S_I \leftarrow g^{e_{[n]\setminus I}}$ and then

$$\Lambda_I \leftarrow \prod_{j=1, j\notin I}^{n} \left(S_j^{1/e_I}\right)^{y_j} = \left(\prod_{j=1, j\notin I}^{n} S_j^{y_j}\right)^{1/e_I}$$

Returns $\pi_I := (S_I, \Lambda_I)$

$\mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}, \pi_I) \to b$ Parse $\pi_I := (S_I, \Lambda_I)$, and compute $S_i = S_I^{e_{I\setminus\{i\}}} = U_n^{1/e_i}$ for every $i \in I$. Return 1 (accept) if both the following checks hold, and 0 (reject) otherwise:

$$S_I^{e_I} = U_n \ \wedge \ C = \Lambda_I^{e_I} \prod_{i \in I} S_i^{y_i}$$

The correctness of the above construction holds essentially the same as the one of the SVC of [66, 145] with the addition of the S_I elements of the openings, whose correctness can be seen by inspection (and is the same as for RSA accumulators).

Incremental Aggregation. Let us now show that the SVC above has incremental aggregation. Note that our algorithms also implicitly show that the RSA-based SVC of [145] is incrementally aggregatable.

$\mathsf{VC.Disagg}(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, K) \to \pi_K$ Parse $\pi_I := (S_I, \Lambda_I)$. First compute S_K from S_I, $S_K \leftarrow S_I^{e_{I\setminus K}}$, and then, for every $j \in I \setminus K$, $\chi_j = S_K^{1/e_j}$, e.g., by computing $\chi_j \leftarrow S_I^{e_{I\setminus(K\cup\{j\})}}$.
Return $\pi_K := (S_K, \Lambda_K)$ where

$$\Lambda_K \leftarrow \Lambda_I^{e_{I\setminus K}} \cdot \prod_{j \in I\setminus K} \chi_j^{v_j}$$

$\mathsf{VC.Agg}(\mathsf{pp}, (I, \boldsymbol{v}_I, \pi_I), (J, \boldsymbol{v}_J, \pi_J)) \to \pi_K$ Parse $\pi_I := (S_I, \Lambda_I)$ and similarly π_J. Also, let $K = I \cup J$, and assume for simplicity that $I \cap J = \emptyset$ (if this is not the case, one could simply disaggregate π_I (or π_J) to $\pi_{I \setminus J}$ (or $\pi_{J \setminus I}$)).

First, compute $S_K \leftarrow \mathbf{ShamirTrick}(S_I, S_J, e_I, e_J)$. Next, compute $\phi_j \leftarrow S_K^{e_{J\setminus\{j\}}} = S_I^{1/e_j}$ for every $j \in J$, and similarly $\psi_i \leftarrow S_K^{e_{I\setminus\{i\}}} = S_J^{1/e_i}$ for every $i \in I$. Then compute

$$\rho_I \leftarrow \frac{\Lambda_I}{\prod_{j\in J} \phi_j^{v_j}} \qquad \text{and} \qquad \sigma_J \leftarrow \frac{\Lambda_J}{\prod_{i\in I} \psi_i^{v_i}}$$

Return $\pi_K := (S_K, \Lambda_K)$ where $\Lambda_K \leftarrow \mathbf{ShamirTrick}(\rho_I, \sigma_J, e_I, e_J)$.

Aggregation Correctness. It follows from the correctness of Shamir's trick and by construction. In Aggregation and disaggregation S_K's correctness is straightforward, so we emphasize on Λ_K. For the disaggregation algorithm:

$$\begin{aligned}
\Lambda_K := \Lambda_I^{e_{I\setminus K}} \cdot \prod_{j\in I\setminus K} \chi_j^{v_j} &= \left(\prod_{j=1, j\notin I}^{n} S_j^{v_j}\right)^{\frac{1}{e_I} e_{I\setminus K}} \cdot \prod_{j\in I\setminus K} \left(S_j^{1/e_K}\right)^{v_j} \\
&= \left(\prod_{j=1, j\notin I}^{n} S_j^{v_j}\right)^{\frac{1}{e_K}} \cdot \left(\prod_{j\in I\setminus K} S_j^{v_j}\right)^{1/e_K} \\
&= \left(\prod_{j=1, j\notin K}^{n} S_j^{v_j}\right)^{1/e_K}
\end{aligned}$$

which is a valid opening for the K-subvector. And for the aggregation algorithm:

$$\rho_I := \frac{\Lambda_I}{\prod_{j\in J} \phi_j^{v_j}} = \left(\prod_{j=1, j\notin I\cup J}^{n} S_j^{v_j}\right)^{1/e_I} \quad \text{and} \quad \sigma_J := \frac{\Lambda_J}{\prod_{j\in I} \psi_j^{v_j}} = \left(\prod_{j=1, j\notin J\cup I}^{n} S_j^{v_j}\right)^{1/e_J}$$

so

$$\begin{aligned}
\Lambda_K &:= \mathbf{ShamirTrick}(\rho_I, \sigma_J, e_I, e_J) \\
&= \mathbf{ShamirTrick}\left(\left(\prod_{j=1, j\notin I\cup J}^{n} S_j^{v_j}\right)^{1/e_I}, \left(\prod_{j=1, j\notin J\cup I}^{n} S_j^{v_j}\right)^{1/e_J}, e_I, e_J\right) \\
&= \left(\prod_{j=1, j\notin I\cup J}^{n} S_j^{v_j}\right)^{\frac{1}{e_I e_J}} = \left(\prod_{j=1, j\notin I\cup J}^{n} S_j^{v_j}\right)^{\frac{1}{e_{I\cup J}}}
\end{aligned}$$

which is a valid opening for the $(I \cup J)$-subvector.

Efficiency. We summarize the efficiency of this construction in terms of both the computational cost of each algorithm and the communication. For the analysis we consider an instantiation of $\mathsf{PrimeGen}$ with a deterministic function that maps every integers in $[n]$ into an ℓ-bit prime number.

Also, we observe that the algorithms described above may have different implementations: while straightforward instantiations may lead to a complexity quadratic in the (sub)vector's length, in what follows we discuss more efficient ways that keeps the complexity quasilinear. For this, we often rely on the $\mathbf{MultiExp}$ algorithm described in [40]. On input an integer n, and two vectors $\boldsymbol{\alpha} \in \mathbb{G}^n$ and $\boldsymbol{x} \in \mathbb{Z}^n$, $\mathbf{MultiExp}(n, \boldsymbol{\alpha}, \boldsymbol{x})$ is a divide-and-conquer algorithm that computes $\prod_{i=1}^{n} \alpha_i^{x^*/x_i}$ where $x^* = \prod_{i=1}^{n} x_i$, and it does it in time $O(n \log n)$, instead of a naive $O(n^2)$.

Setup. Setup generates a group description and samples one random group element, while $\mathsf{Specialize}$ computes one exponentiation with an $(\ell \cdot n)$-bits integer. Both the universal and the specialized CRS consist each of 1 element of $\mathbb{G}$.

Committing. Committing to a vector $\boldsymbol{v} \in (\{0,1\}^{\ell})^n$ can be done in time $O(\ell\ n \log n)$ by using the $\mathbf{MultiExp}$ algorithm from [40], i.e., $C \leftarrow \mathbf{MultiExp}(n, \boldsymbol{\alpha}, \boldsymbol{e})$ where $\alpha_i = g^{v_i}$ and $e_i = \mathsf{PrimeGen}(i)$. The commitment is a single element of $\mathbb{G}$.

Opening. An opening for a set I of m positions consists of two group elements, and it can be computed as follows. First, compute S_I through the exponentiation $g^{e_{[n]\setminus I}}$ which requires $O(\ell(n-m))$ group operations, and then compute Λ_I in a way similar to committing, i.e., $\Lambda_I \leftarrow \mathbf{MultiExp}(n', \boldsymbol{\alpha}, \boldsymbol{x})$, where $n' = n - m$, $\boldsymbol{\alpha} = (g^{v_j})_{j \in [n]\setminus I}$, $\boldsymbol{x} = (e_j)_{j \in [n]\setminus I}$, which takes time $O(\ell(n-m)\log(n-m))$.

Verification. Verifying an opening for I of size m requires two exponentiations with an (ℓm)-bits long integer ($S_I^{e_I}$ and $\Lambda_I^{e_I}$), and the computation of $\prod_{i \in I} S_i^{y_i}$ can be done in time $O(\ell m \log m)$ by running $\mathbf{MultiExp}(m, \boldsymbol{\alpha}, \boldsymbol{x})$ with $\boldsymbol{\alpha} = (S_I^{y_1}, \ldots, S_I^{y_m})$ and $\boldsymbol{x} = (e_i)_{i \in I}$.

Aggregation and Disaggregation. Disaggregation can be computed in time $O(\ell(|I|-|K|)\log(|I|-|K|))$ in a way similar to verification: two exponentiations with an $\ell(|I|-|K|)$-bits long integer each, and an invocation of $\mathbf{MultiExp}((|I|-|K|), \boldsymbol{\alpha}, \boldsymbol{x})$, with $\boldsymbol{\alpha} = (S_I^{v_j})_{j \in I \setminus K}$ and $\boldsymbol{x} = (e_j)_{j \in I \setminus K}$, to compute $\prod_{j \in I \setminus K} S_I^{e_{I \setminus (K \cup \{j\})} \cdot v_j}$.

Aggregation can be computed in time $O(\ell m \log m)$ where $m = \max(|I|, |J|)$ as follows. Two invocations of $\mathbf{ShamirTrick}$, each requiring two exponentiations with (ℓm)-bits long integers, to compute S_K and Λ_K, and two invocations of $\mathbf{MultiExp}$ to compute $\prod_{j \in J} \phi_j^{v_j}$ and $\prod_{i \in I} \psi_i^{v_i}$ respectively. From this, we obtain that $\mathsf{VC.AggManyToOne}$ and $\mathsf{VC.DisaggOneToMany}$ take time $O(\ell m \log^2 m)$ $\mathbb{G}$ and $O(\ell m \log m \log(m/B))$ $\mathbb{G}$, respectively.

Commitment and Opening with Precomputation. Finally, let us summarize the costs of committing and opening with preprocessing obtained by instantiating our method of Section 6.4.2. The preprocessing $\mathsf{VC.PPCom}$, with parameter B, requires $O(\ell n \log n \log(n/B))$ operations of $\mathbb{G}$ and produces a storage advice of $2n/B$ group elements. The opening requires computing at most $|S| \leq m$ disaggregation, each taking time $O(\ell(|P_j| - |I_j|)\log((|P_j| - |I_j|)))$, for a total of $O(\ell(|S|B - |I|)\log(|S|))$, followed by the aggregation step that counts $O(\ell|S|\log^2|S|)$. So, in the worst case $\mathsf{VC.FastOpen}$ takes $O(\ell \cdot m \cdot (\log^2(m) + B - 1))$ operations of $\mathbb{G}$.

Security. For the security of the above SVC scheme we observe that the difference with the corresponding [145] lies in the generation of S_i's. In [145] they are generated in the trusted setup phase, thus they are considered "well-formed" in the security proof. In our case, the S_i's are reconstructed during verification time from the S_I that comes in the opening π_I which can (possibly) be generated in an adversarial way. However, in the verification it is checked that $S_I^{e_I} = U$, where $U = g^{e_{[n]}}$ is computed in the trusted setup. So under the Low Order assumption

we get that S_I has the correct form, $S_I = g^{e_{[n]}/e_I} = g^{e_{[n]\setminus I}}$, with overwhelming probability. Except for this change, the rest reduces to the position binding of the [145] SVC.

Theorem 22 (Position-Binding)**.** *Let* Ggen *be the generator of hidden order groups where the Low Order assumption holds and the [145] SVC is position binding. Then the SVC scheme defined above is position binding.*

Proof. We start by defining the game G_0 as the actual position binding game of Definition 12, and our goal is to prove that for any PPT $\mathcal{A}$, $\Pr[G_0 = 1] \in \mathsf{negl}(\lambda)$:

Game G_0**:**

$G_0 = \mathsf{PosBind}^{\mathcal{A}}_{\mathsf{VC}}(\lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$

$(C, I, \boldsymbol{y}, \pi, \boldsymbol{y}', \pi') \leftarrow \mathcal{A}(\mathsf{pp})$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}, \pi) = 1 \wedge \boldsymbol{y} \neq \boldsymbol{y}' \wedge \mathsf{Ver}(\mathsf{pp}, C, I, \boldsymbol{y}', \pi') = 1$

return b

More specifically $\mathsf{pp} := (\mathbb{G}, g, \mathsf{PrimeGen})$, $\pi := (S_I, \Lambda_I)$, $\pi' := (S'_I, \Lambda'_I)$ and

$$b = S_I^{e_I} = U_n \wedge C = \Lambda_I^{e_I} \prod_{i \in I} S_i^{y_i} \wedge \boldsymbol{y} = \boldsymbol{y}' \wedge S_I'^{e_I} = U_n \wedge C = \Lambda_I'^{e_I} \prod_{i \in I} S_i'^{y'_i}$$

where $S_i = S_I^{e_I \setminus \{i\}}$ and $S'_i = S_I'^{e_I \setminus \{i\}}$ for each $i \in I$.

Now let G_1 be the same as above except for the outputted by the adversary S_I and S'_I it holds that $S_I = g^{e_{[n]\setminus I}} = S'_I$. The $S_I^{e_I} = U_n = S_I'^{e_I}$ checks are not done in the verification (as they are redundant):

Game G_1**:**

G_1

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$

$(C, I, \boldsymbol{y}, (S_I, \Lambda_I), \boldsymbol{y}', (S'_I, \Lambda'_I)) \leftarrow \mathcal{A}(\mathsf{pp})$

if $S_I \neq g^{e_{[n]\setminus I}}$ or $S'_I \neq g^{e_{[n]\setminus I}}$ **then** abort

$b \leftarrow C = \Lambda_I^{e_I} \prod_{i \in I} \left(S_I^{e_I \setminus \{i\}}\right)^{y_i} \wedge \boldsymbol{y} = \boldsymbol{y}' \wedge C = \Lambda_I'^{e_I} \prod_{i \in I} \left(S_I'^{e_I \setminus \{i\}}\right)^{y'_i}$

return b

Then $\Pr[G_0 = 1] \leq \Pr[G_1 = 1] + \mathsf{negl}(\lambda)$. In G_0, $S_I^{e_I} = U_n = g^{e_{[n]}}$. Assume that $S_I \neq g^{e_{[n]\setminus I}}$ then $g^{e_{[n]\setminus I}} = S_I^*$, hence $S_I^{e_I} = S_I^{*e_I} \Rightarrow \left(S_I^{-1} S_I^*\right)^{e_I} = 1$. Since S_I^* is efficiently computable and $e_I < 2^{\mathsf{poly}(\lambda)}$ this constitutes a solution to the Low Order problem for the hidden order group. The previous happens only with negligible probability under the Low Order assumption. The same holds for S'_I. Notice that it follows that $S_i = S'_i = g^{e_{[n]\setminus\{i\}}}$.

Let G_2 be the same as above except the adversary receives $e_i \leftarrow \mathsf{PrimeGen}(i)$ and $S_i = g^{e_{[n]\setminus\{i\}}}$ for each $i \in [n]$, together with the parameters:

Game G_2**:**

G_2

$(\mathbb{G}, g, \mathsf{PrimeGen}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$

$e_i \leftarrow \mathsf{PrimeGen}(i); S_i = g^{\prod_{i \in [n] \setminus \{i\}} e_i}$ for each $i \in [n]$

$(C, I, \boldsymbol{y}, \Lambda_I, \boldsymbol{y}', \Lambda'_I) \leftarrow \mathcal{A}\left(\mathbb{G}, g, \mathsf{PrimeGen}, \{S_i\}_{i \in [n]}\right)$

$b \leftarrow C = \Lambda_I^{e_I} \prod_{i \in I} S_i^{y_i} \wedge \boldsymbol{y} = \boldsymbol{y}' \wedge C = \Lambda_I'^{e_I} \prod_{i \in I} S_i^{y'_i}$

return b

It is straightforward that $\Pr[G_1 = 1] = \Pr[G_2 = 1]$ and furthermore G_2 is identical to the position binding game of the [145] SVC scheme and according to the hypothesis $\Pr[G_2 = 1] = \mathsf{negl}(\lambda)$. □

As showed in [145], their SVC is position binding under the strong Distinct-Prime-Product Root assumption in the standard model. We conclude that the above SVC is position binding in hidden order groups where the Low Order and the Strong Distinct-Prime-Product Root assumptions hold.

6.5.3 Comparison with Related Work

We compare our two SVC schemes with the recent scheme proposed by Boneh et al. [40] and the one by Lai and Malavolta [145], which extends [66] to support subvector openings.[43] We present a detailed comparison in Table 6.1, considering to work with vectors of length N of ℓ-bit elements and security parameter λ. In particular we consider an instantiation of our first SVC with $k = 1$ (and thus $n = N \cdot \ell$).

Setup Model. [40] works with a fully universal CRS, whereas our schemes have both a universal CRS with deterministic specialization, which however, in comparison to [66, 145], outputs *constant-size* parameters instead of linear.

Aggregation. The VC of [40] supports aggregation only on openings created by Open (i.e., it is one-hop) and does not have disaggregatable proofs (unless in a different model where one works linearly in the length of the vector or knows the full vector). In contrast, we show the first schemes that satisfy incremental aggregation (also, our second one immediately yields a method for the incremental aggregation of [145]). As we mention later, incremental aggregation can be very useful to precompute openings for a certain number of vector blocks allowing for interesting time-space tradeoffs that can speedup the running time of Open.

Efficiency. From the table, one can see that our first SVC has: slightly worse commitments size than all the other schemes, computational asymptotic performances similar to [40], and opening size slightly better than [40]. Our second SVC is the most efficient among the schemes with constant-size parameters; in particular, it has faster asymptotics than our first SVC and [40] for having a smaller logarithmic factor (e.g., $\log(N - m)$ vs. $\log(\ell N)$), which is due to the avoidance of using one prime per bit of the vector. In some cases, [66, 145] is slightly better, but this is essentially a benefit of the linear-size parameters, namely the improvement is due to having the S_i's elements already precomputed.

[43] We refer to [40] to see how these schemes compare with Merkle trees.

When considering applications in which a user creates the commitment to a vector and (at some later points in time) is requested to produce openings for various subvectors, *our incremental aggregation property leads to use preprocessing to achieve more favorable time and memory costs*. In a nutshell, The idea of preprocessing is that one can precompute and store information that allows to speedup the generation of openings, in particular by making opening time less dependent on the total length of the vector. Our method in Section 6.4.2 works generically for any SVC that has incremental aggregation. A similar preprocessing solution can also be designed for the SVC of [40] by using its one-hop aggregation; we provide a detailed description of the method in Appendix 6.10. The preprocessing for [40] however has no flexibility in choosing how much auxiliary storage can be used, and one must store (a portion of) a non-membership witness *for every bit* of the vector.

Even in the simplest case of $B = 1$ (shown in Table 6.1) both our SVCs save a factor ℓ in storage, which concretely turns into $3\times$ less storage.

Furthermore we support flexible choices of B thus allowing to tune the amount of auxiliary storage. For instance, we can choose $B = \sqrt{N}$ so as to get $2\sqrt{N}|\mathbb{G}|$ bits of storage, and opening time about $O(\ell m \log n(\sqrt{n} + \log m))$ and $O(m(\sqrt{n} + \log^2 m))$ in the first and second scheme respectively. Our flexibility may also allow one to choose the buckets size B and their distribution according to applications-dependent heuristics; investigating its benefit may be an interesting direction for future work.

Metric	**Our First SVC**	**[40]**	**[66, 145]**
	Setup		
Setup	$O(1)$	$O(1)$	$O(1)$
$\lvert\mathsf{pp}\rvert$	$3\,\lvert\mathbb{G}\rvert$	$1\,\lvert\mathbb{G}\rvert$	$1\,\lvert\mathbb{G}\rvert$
Specialize	$O(\ell \cdot N \cdot \log(\ell N))\ \mathbb{G}$	—	$O(\ell \cdot N \cdot \log N)\ \mathbb{G}$
$\lvert\mathsf{pp}_N\rvert$	$1\,\lvert\mathbb{G}\rvert$	—	$N\,\lvert\mathbb{G}\rvert$
	Commit a vector $\boldsymbol{v} \in (\{0,1\}^\ell)^N$		
Com	$O(\ell \cdot N \cdot \log(\ell N))\ \mathbb{G}$	$O(\ell \cdot N \cdot \log(\ell N))\ \mathbb{G}$	$O(\ell \cdot N)\ \mathbb{G}$
$\lvert C\rvert$	$4\,\lvert\mathbb{G}\rvert + 2\,\lvert\mathbb{Z}_{2^k}\rvert$	$1\,\lvert\mathbb{G}\rvert$	$1\,\lvert\mathbb{G}\rvert$
	Opening and Verification for $\boldsymbol{v}_I$ with $\lvert I\rvert = m$		
Open	$O(\ell \cdot (N-m) \cdot \log(\ell N))\ \mathbb{G}$	$O(\ell \cdot (N-m) \cdot \log(\ell N))\ \mathbb{G}$	$O(\ell \cdot (N-m) \cdot m \log m)\ \mathbb{G}$
$\lvert\pi_I\rvert$	$4\,\lvert\mathbb{G}\rvert$	$5\,\lvert\mathbb{G}\rvert + 1\,\lvert\mathbb{Z}_{2^k}\rvert$	$1\,\lvert\mathbb{G}\rvert$
Ver	$O(m \cdot \ell \cdot \log(\ell N))\ \mathbb{Z}_{2^k} + O(\lambda)\ \mathbb{G}$	$O(m \cdot \ell \cdot \log(\ell N))\ \mathbb{Z}_{2^k} + O(\lambda)\ \mathbb{G}$	$O(\ell \cdot m)\ \mathbb{G}$
	Commitment and Opening with Precomputation		
Com	$O(\ell \cdot N \cdot \log(\ell \cdot N) \cdot \log(N))\ \mathbb{G}$	$O(\ell \cdot N \cdot \log(\ell \cdot N) \cdot \log(N))\ \mathbb{G}$	$O(\ell \cdot N^2)$
$\lvert\mathsf{aux}\rvert$	$O(N)\ \lvert\mathbb{G}\rvert$	$O(N)\ \lvert\mathbb{G}\rvert + O(\ell \cdot N)\lvert\mathbb{Z}_{2^k}\rvert$	$O(N)\ \lvert\mathbb{G}\rvert$
Open	$O(m \cdot \ell \cdot \log(m) \log(\ell N))\ \mathbb{G}$	$O(m \cdot \ell \cdot \log(m) \log(\ell N))\ \mathbb{G}$	$O(m \cdot \ell \cdot \log(m))\ \mathbb{G}$
Aggregation	Incremental	One-hop	Incremental
Disaggregation	Yes	No	Yes

Table 6.1: Comparison between the SVC's of [40], [145] and this work (including our incremental aggregation for [145]); our contributions highlighted in gray. We consider committing to a vector $\boldsymbol{v} \in (\{0,1\}^\ell)^N$ of length N, and opening and verifying for a set I of m positions. By '$O(x)\ \mathbb{G}$' we mean $O(x)$ group operations in $\mathbb{G}$; $|\mathbb{G}|$ denotes the bit length of an element of $\mathbb{G}$. An alternative algorithm for Open in [145] costs $O(\ell \cdot (N-m) \cdot \log(N-m))$.

6.6 Arguments of Knowledge for Our First SVC

We propose three Arguments of Knowledge (AoK) related to our vector commitment scheme presented in section 6.5.1. More specifically, the first AoK allows one to prove knowledge of

$\mathsf{Setup}(1^\lambda)$: run $\mathbb{G} \leftarrow_\$ \mathsf{Ggen}(1^\lambda)$, $g \leftarrow_\$ \mathbb{G}$, set $\mathsf{crs}^* := (\mathbb{G}, g)$.
Prover's input: $(\mathsf{crs}^*, (A, B, C, \Gamma, \Delta), (a, b))$. Verifier's input: $(\mathsf{crs}^*, (A, B, C, \Gamma, \Delta))$.

$\mathsf{V} \to \mathsf{P}$: $h \leftarrow_\$ \mathbb{G}$

$\mathsf{P} \to \mathsf{V}$: $z := (z_a, z_b)$ computed as $z_a \leftarrow h^a, z_b \leftarrow h^b$

$\mathsf{V} \to \mathsf{P}$: $\ell \leftarrow_\$ \mathbb{P}(1, 2^\lambda)$ and $\alpha \leftarrow_\$ [0, 2^\lambda)$

$\mathsf{P} \to \mathsf{V}$: $\pi := ((Q_A, Q_B, Q_C), r_a, r_b)$ computed as follows

- $(q_a, q_b, q_{ab}) \leftarrow (\lfloor a/\ell \rfloor, \lfloor b/\ell \rfloor, \lfloor ab/\ell \rfloor)$
- $(r_a, r_b) \leftarrow (a \mod \ell, b \mod \ell)$
- $(Q_A, Q_B, Q_C) := (\Gamma^{q_a} h^{\alpha q_a}, \Delta^{q_b} h^{\alpha q_b}, g^{q_{ab}})$

$\mathsf{V}(\mathsf{crs}, (A, B, C), z_a, z_b, \ell, \alpha, \pi)$:

- Compute $r_c \leftarrow r_a \cdot r_b \mod \ell$
- Output 1 iff $r_a, r_b \in [\ell] \ \wedge \ Q_A^\ell \Gamma^{r_a} h^{\alpha r_a} = A z_a^\alpha \ \wedge \ Q_B^\ell \Delta^{r_b} h^{\alpha r_b} = B z_b^\alpha \ \wedge \ Q_C^\ell g^{r_c} = C$

Figure 6.4: PoProd^* protocol

an opening of a subvector. The second AoK, is a direct outcome of the first and allows one to prove that two given commitments share a common subvector. Finally, the third protocol allows one to commit to a prefix-subvector of a vector and prove the knowledge of it succinctly.

Similarly to section 6.5.1.1, our protocols build on the techniques for succinct proofs in groups of unknown order from [40]. Furthermore, these arguments of knowledge are not zero knowledge and they serve efficiency purposes. Interestingly, one can prove knowledge of a portion of a vector committed *without having to send the actual vector values*. The proofs are constant-size which leads to an improvement of communication complexity linear in the size of the opening.

6.6.1 Building block: A Stronger Proof of Product

Before proceeding to describing the main protocols, we introduce another one that is used as building block. This is an argument of knowledge, called PoProd^*, for the relation R_{PoProd^*} described below, which uses a common reference string consisting of a hidden order group $\mathbb{G} \leftarrow \mathsf{Ggen}(1^\lambda)$ and a random generator $g \in \mathbb{G}$:

$$R_{\mathsf{PoProd}^*} = \left\{ ((A, B, C, \Gamma, \Delta), (a, b)) \in \mathbb{G}^5 \times \mathbb{Z}^2 \ : \ A = \Gamma^a \wedge B = \Delta^b \wedge C = g^{a \cdot b} \quad \right\}$$

The relation R_{PoProd^*} is similar to R_{PoProd} defined in Section 6.5.1.1 with the difference that now the first two bases Γ and Δ are not part of the common reference string, but part of the statement instead. As argued in [40] the PoKE protocol is not secure anymore for adversarially chosen bases, therefore we cannot use PoProd protocol which assumes knowledge extractability of PoKE. To deal with this problem, we thus modify the protocol by using the protocol PoKE2, which is secure for arbitrary bases. This comes with some cost: in our PoProd^* a proof consists of 5 group elements and 2 field elements, that is 2 group elements more comparing to proofs of PoProd. The protocol is in Figure 6.4.

Theorem 23. *The* PoProd^* *protocol in Fig. 6.4 is an argument of knowledge for* R_{PoProd^*} *in the generic group model.*

The proof of the theorem above is similar to the proof of Theorem 20, except that we use the extractor $\mathcal{E}_{\mathsf{PoKE2}}$ of the protocol PoKE2 from [40] in order to extract integers a and b and $\mathcal{E}_{\mathsf{PoKE}}$ in order to extract the exponent of C.

6.6.2 A Succinct AoK of Opening for our VC Construction

We show an argument of knowledge of an I-opening with respect to a commitment C to a vector, where I is a set of positions. We emphasize that the goal of this protocol is not to keep the opening secret (i.e., the protocol is not zero knowledge, also our vector commitment scheme is not hiding). The goal is to reduce the communication complexity of an opening by proving knowledge of the subvector at positions I without having to actually send the values $\boldsymbol{v}_I$. Even though the argument of knowledge itself adds an overhead it is independent of the number of the positions. Hence, the protocol makes more sense for large sets of positions I as for a small number of positions the overhead of the AoK would exceed the size of the opening values.

Let $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Specialize}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be our SVC scheme from Section 6.5.1, and let us define the following relation

$$R_{\mathsf{PoKOpen}} = \{(\,(c, I), (\boldsymbol{y}, \pi_I)\,) : \mathsf{Ver}(\mathsf{pp}, c, I, \boldsymbol{y}, \pi_I) = 1\}$$

that is parametrized by a CRS $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$, and where the statement consists of a commitment c and a set of indices $I \subseteq [n]$, and the witness consists of a vector $\boldsymbol{y} \in \mathcal{M}^{|I|}$ and an opening π_I.

For simplicity we present a protocol PoKOpen for the case when $k = 1$ in our VC (see section 6.5.1); extension to larger k is immediate. The idea of our protocol is that, given a commitment $C := ((A, B), \pi_{\mathsf{prod}})$ and a set of indices I, the prover, holding $\pi_I := (\Gamma_I, \Delta_I)$, first sends π_I to the verifier and then provides an AoK of (a_I, b_I) such that $\Gamma_I^{a_I} = A \wedge \Delta_I^{b_I} = B \wedge g^{a_I \cdot b_I} = U_I$, where $U_I \leftarrow g^{u_I}$ with $u_I \leftarrow \mathsf{PrimeProd}(I)$. This can be proven by using the PoProd^* protocol presented above. Finally the verifier should also verify the π_{prod} proof as in the normal verification of an opening algorithm.

We state the following theorem.

Theorem 24. *If* PoProd^* *is a succinct argument of knowledge for* R_{PoProd^*}*, then protocol* PoKOpen *is a succinct argument of knowledge for relation* R_{PoKOpen} *with respect to algorithm* Ver *of our construction of Section 6.5.1.*

Proof. Let $\mathcal{A}$ be an adversary of the Knowledge Extractability of PoKOpen such that: $((C, I), \mathsf{state}) \leftarrow \mathcal{A}_0(\mathsf{pp})$, $\mathcal{A}_1(\mathsf{pp}, (C, I), \mathsf{state})$ executes with $\mathcal{V}(\mathsf{pp}, (C, I))$ the protocol PoKOpen and the verifier accepts with a non-negligible probability ϵ. We will construct an extractor $\mathcal{E}$ that having access to the internal state of $\mathcal{A}_1$ and on input $(\mathsf{pp}, (C, I), \mathsf{state})$, outputs a witness $(\boldsymbol{y}, \pi_I)$ of R_{PoKOpen} with overwhelming probability and runs in (expected) polynomial time.

To prove knowledge extractability of PoKOpen we rely on the knowledge extractability of PoProd^*. More precisely, given a PoKOpen execution between $\mathcal{A}$ and $\mathcal{V}$, $(\Gamma_I, \Delta_I, \pi_{\mathsf{PoProd}'})$, $\mathcal{E}$ constructs an adversary $\mathcal{A}' = (\mathcal{A}'_0, \mathcal{A}'_1)$ of PoProd^* Knowledge Extractability and, by using the input and internal state of $\mathcal{A}_1$, simulates an execution between $\mathcal{A}'$ and $\mathcal{V}$: $\mathcal{A}'_0$ outputs $(((\mathbb{G}, g), (A, B, U_I, \Gamma_I, \Delta_I)), \mathsf{state})$, $\mathcal{A}'_1$ outputs tuple

PoKOpen protocol

Prover's input: $(\mathsf{pp}, (c, I), (\boldsymbol{y}, \pi_I)$. Verifier's input: $(\mathsf{pp}, (c, I))$.

$\underline{\mathsf{V}}$ Compute $u_I \leftarrow \mathsf{PrimeProd}(I)$ and then $U_I \leftarrow g^{u_I}$.

Similarly compute $u_n \leftarrow \mathsf{PrimeProd}([n])$ and then $U_n \leftarrow g^{u_n}$

$\underline{\mathsf{P}}$: Parse $\mathsf{pp} := (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen}, U_n)$, $c := (\{A, B\}, \pi_{\mathsf{prod}})$, $\pi_I := (\Gamma_I, \Delta_I)$. Compute $(a_I, b_I) \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y})$ and then $u_I \leftarrow \mathsf{PrimeProd}(I)$ and $U_I \leftarrow g^{u_I}$.

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: (Γ_I, Δ_I)

Finally a PoProd^* protocol (with an additional check of the commitment) between $\mathsf{P}((\mathbb{G}, g), (A, B, U_I, \Gamma_I, \Delta_I), (a_I, b_I))$ and $\mathsf{V}((\mathbb{G}, g), (A, B, U_I, \Gamma_I, \Delta_I))$ is executed:

$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: $h \leftarrow_\$ \mathbb{G}$

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: $z := (z_a, z_b)$ computed as $z_a \leftarrow h^{a_I}, z_b \leftarrow h^{b_I}$

$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: $\ell \leftarrow_\$ \mathbb{P}(1, 2^\lambda)$ and $\alpha \leftarrow_\$ [0, 2^\lambda)$

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: $\pi := ((Q_A, Q_B, Q_C), r_a, r_b)$ computed as follows

- $(q_a, q_b, q_{ab}) \leftarrow (\lfloor a_I/\ell \rfloor, \lfloor b_I/\ell \rfloor, \lfloor a_I b_I/\ell \rfloor)$
- $(r_a, r_b) \leftarrow (a_I \mod \ell, b_I \mod \ell)$
- $(Q_A, Q_B, Q_C) := \left(\Gamma_I^{q_a} h^{\alpha q_a}, \Delta_I^{q_b} h^{\alpha q_b}, g^{q_{ab}}\right)$

$\underline{\mathsf{V}}$: Parse $\mathsf{pp} := (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen}, U_n)$ and $C := (\{A, B\}, \pi_{\mathsf{prod}})$.

- Compute $r_c \leftarrow r_a \cdot r_b \mod \ell$
- Output 1 iff $r_a, r_b \in [\ell] \ \wedge \ Q_A^\ell \Gamma_I^{r_a} h^{\alpha r_a} = A z_a^\alpha \ \wedge \ Q_B^\ell \Delta_I^{r_b} h^{\alpha r_b} = B z_b^\alpha \ \wedge \ Q_C^\ell g^{r_c} = U_I \wedge \mathsf{PoProd}_2.\mathsf{V}(\mathsf{pp}, (A \cdot B, U_n), \pi_{\mathsf{prod}})$

Figure 6.5: PoKOpen protocol

$(z_a, z_b, (Q_A, Q_B, Q_C), r_a, r_b)$. It is obvious that if the initial execution is accepted by $\mathcal{V}$ so is the PoProd^* execution. From Knowledge Extractability of PoProd we know that there exists an extractor $\mathcal{E}'$ corresponding to $\mathcal{A}'_1$ that outputs (a_I, b_I) such that $A = \Gamma_I^{a_I} \wedge B = \Delta_I^{b_I} \wedge U_I = g^{a_I \cdot b_I}$. Since U_I is also computed from $\mathcal{V}$ it holds that $U_I = g^{u_I}$, unless with a negligible probability that $\mathcal{A}'$ can find an $x \neq u_I$ such that $g^x = U_I = g^{u_I}$ (which implies finding a multiple of the order of $\mathbb{G}$). Therefore $g^{u_I} = U_I = g^{a_I \cdot b_I}$ and using the same argument we know that $u_I = a_I \cdot b_I$ (unless with negligible probability).

So, $\mathcal{E}$ uses $\mathcal{E}'$ and gets a (a_I, b_I) such that $A = \Gamma_I^{a_I} \wedge B = \Delta_I^{b_I} \wedge U_I = g^{a_I \cdot b_I}$. Then computes $u_I \leftarrow \mathsf{PrimeProd}(I)$ and works as follows: for each $i \in I$ computes $p_i \leftarrow \mathsf{PrimeGen}(i)$ and if $p_i \mid a_I$ then sets $y_i = 0$, otherwise if $p_i \mid a_I$ then sets $y_i = 1$. It is clear that p_i divides exactly one of a_I, b_I since $a_I \cdot b_I = u_I = \prod_{i \in I} p_i := \prod_{i \in I} \mathsf{PrimeGen}(i)$ (unless with a negligible probability that a collision happened in PrimeGen). Finally sets the subvector $\boldsymbol{y} = (y_i)_{i \in I}$ and $\pi_I = (\Gamma_I, \Delta_I)$. As stated above $\Gamma_I^{a_I} = A \wedge \Delta_I^{b_I} = B$ and also since $\mathcal{V}$ verifies the PoKOpen protocol it holds that $\mathsf{PoProd}_2.\mathsf{V}(\mathsf{pp}, (A \cdot B, U_n), \pi_{\mathsf{prod}})$ which means that $\mathsf{Ver}(\mathsf{pp}, c, I, \boldsymbol{y}, \pi_I) = 1$.

As one can see, the expected running time of $\mathcal{E}$ is the (expected) time to obtain a successful execution of the protocol plus the running time to obtain $\boldsymbol{y}$ plus the running time of $\mathcal{E}'$. To obtain $\boldsymbol{y}$ it will need to make $|I|$ divisibility checks which takes time $\tilde{O}(|I|)$ plus $|I|$ calls of $\mathsf{PrimeGen}$, which takes $\mathsf{poly}(\lambda)$ time. So overall the expected time is $\frac{1}{\epsilon} + t_{\mathcal{E}'} + \tilde{O}(|I|) + \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda)$. □

Non-interactive PoKOpen**.** A non-interactive version of the protocol PoKOpen after applying the generalized Fiat-Shamir transform [26] is shortly presented below:

$\mathsf{PoKOpen.P}(\mathsf{pp}, (c, I), (\boldsymbol{y}, \pi_I)) \to \pi$**:** Parse $\mathsf{pp} := (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen}, U_n)$, $c := (\{A, B\}, \pi_{\mathsf{prod}})$, $\pi_I := (\Gamma_I, \Delta_I)$. Compute $(a_I, b_I) \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y})$ and then $u_I \leftarrow \mathsf{PrimeProd}(I)$ and $U_I \leftarrow g^{u_I}$. Finally compute a proof $\pi_{\mathsf{PoProd}^*} \leftarrow \mathsf{PoProd}^*.\mathsf{P}((\mathbb{G}, g), (A, B, U_I, \Gamma_I, \Delta_I), (a_I, b_I))$.

Return $\pi \leftarrow (\Gamma_I, \Delta_I, \pi_{\mathsf{PoProd}^*})$

$\mathsf{PoKOpen.V}(\mathsf{pp}, (c, I), \pi_{\mathsf{PoProd}^*}) \to b$**:** Parse $\mathsf{pp} := (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen}, U_n)$, $C := (\{A, B\}, \pi_{\mathsf{prod}})$ and $\pi := (\Gamma_I, \Delta_I, \pi_{\mathsf{PoProd}^*})$. Compute $u_I \leftarrow \mathsf{PrimeProd}(I)$ and then $U_I \leftarrow g^{u_I}$.

Return 1 if both $\mathsf{PoProd_2.V}(\mathsf{pp}, (A \cdot B, U_n), \pi_{\mathsf{prod}})$ and $\mathsf{PoProd}^*.\mathsf{V}((\mathbb{G}, g), (A, B, \Gamma_I, \Delta_I, U_I), \pi_{\mathsf{PoProd}^*})$ output 1, and 0 otherwise.

Remark 18 (Achieving sub-linear verification time)**.** *For ease of exposition we presented the case of $k = 1$ in the above. For the case of arbitrary k one should prove knowledge of (a_{Ij}, b_{Ij}) such that $\bigwedge_{j=1}^{k} \Gamma_{Ij}^{a_{Ij}} = A_j \bigwedge_{j=1}^{k} \Delta_{Ij}^{b_{Ij}} = B \wedge g^{a_{Ij} \cdot b_{Ij}} = U_I$, where $U_I \leftarrow g^{u_I}$ and $u_I \leftarrow \mathsf{PrimeProd}(I)$. Using the same technique as above the size of the AoK is $O(k)$ (as is the commitment and the opening proof). However, since the U_I is the same for each j, the verification is done in $O(|I|/k + \lambda \cdot k)$ time. Interestingly, if $k = \sqrt{|I|}$ the verification time gets $O(\sqrt{|I|})$, which is sublinear in the size of the opening. Essentially, in cases where the opening queries are (approximately) fixed, one can trade a larger commitment size $O(\sqrt{|I|})$ in order to achieve an argument of knowledge of subvectors that has sublinear size and sublinear verification time $O(\sqrt{|I|})$.*

Applications to Compact Proofs of Storage. We observe that the protocol PoKOpen for our VC immediately implies a *keyless* proof of storage, or more precisely a proof of retrievable commitment (PoRC) [105] with non-black-box extraction. In a nutshell, a PoRC is a proof of retrievability [135] of a committed file. In [105] Fisch defines PoRC and proposes a construction based on vector commitments – called VC-PoRC – which abstracts away a classical proof of retrievability based on Merkle trees. A bit more in detail, in the VC-PoRC scheme the prover uses a VC to commit to a file (seen as a vector of blocks); then at every audit the verifier chooses a challenge by picking a set of λ_{pos} randomly chosen positions $I = \{i \leftarrow\!\!\$\, [n]\}$, and the prover responds by sending the subvector $\boldsymbol{v}_I$ and an opening π_I. Here λ_{pos} is a statistical parameter that governs the probability of catching an adversary that deletes (or corrupts) a fraction of the file. For example, if the file is first encoded using an erasure code with constant rate μ (i.e., one where a μ-fraction of blocks suffices to decode and such that the encoded file has size roughly $\mu^{-1} \cdot |F|$), then an erasing adversary has probability at most $\mu^{\lambda_{\mathsf{pos}}}$ of passing an audit.

Our PoRC scheme is obtained by modifying the VC-PoRC of [105] in such a way that the VC opening is replaced by a PoKOpen AoK. This change saves the cost of sending the λ_{pos}

vector values, which gives us *proofs of fixed size*, 7 elements of $\mathbb{G}$ and 2 values of $\mathbb{Z}_{2^{2\lambda}}$. As drawback, our scheme is not black-box extractable; strictly speaking, this means it is not a PoR in the sense of [135] since the extractor does not exist in the real world.[44]

We note that another solution with fixed-size proofs can be achieved by using a SNARK to prove knowledge of the VC openings so that the VC-PoRC verifier would accept. For the Merkle tree VC, this means proving knowledge of λ_{pos} Merkle tree openings, which amounts to proving correctness of about $\lambda_{\mathsf{pos}} \log n$ hash computations. On a file of 2^{20} bits with 128 spot-checks, this solution would reduce proof size from 80KB to less than 1KB. But its concrete proving costs are high (more than 20 minutes and hundreds of GB of RAM).

In contrast we can estimate our AoK to be generated in less than 20 seconds and of size roughly 2KB.

Since our PoRC scheme is a straightforward modification of Fisch's VC-PoRC construction, a complete description is omitted. We stress that our technical contribution here is the design of the AoK.

Finally, we note that we can apply the observation of the previous remark in order to also achieve verification time sub-linear in the size $|I|$ of the challenged subvector at the expense of slightly larger commitments (of size $\sqrt{|I|}$).

6.6.3 An AoK for commitments with common subvector

We note that a simple AND composition of two $\mathsf{PoKOpen}$ arguments of knowledge on two different vector commitments can serve as a protocol proving knowledge of a common subvector of the two vectors committed. More specifically given two vector commitments, C_1, C_2 on two different vector $\boldsymbol{v}_1, \boldsymbol{v}_2$ respectively, one can prove knowledge of a common subvector $\boldsymbol{v}_I$ with a succinct (constant sized) argument without having to send the actual subvector. The two commitments should share the same CRS $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$ though they can have distinct specialized CRSs pp_{n_1} and pp_{n_2} respectively (i.e., $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ may have different length). The underlying relation is:

$$\begin{aligned} R_{\mathsf{PoKComSub}} \quad = \quad & \{((C_1, C_2, I), (\boldsymbol{v}_I, \pi_{I,1}, \pi_{I,2})) : \mathsf{Ver}^\star(\mathsf{pp}_{n_1}, C_1, I, \boldsymbol{v}_I, \pi_{I,1}) = 1 \\ & \wedge \mathsf{Ver}^\star(\mathsf{pp}_{n_2}, C_2, I, \boldsymbol{v}_I, \pi_{I,2}) = 1\} \end{aligned}$$

As mentioned above, it is straightforward to show that an AND composition of $\mathsf{PoKOpen}$ on different vector commitments C_1 and C_2 is a protocol for the above relation. That is the prover, holding $\pi_{I,1} := (\Gamma_{I,1}, \Delta_{I,1})$ and $\pi_{I,2} := (\Gamma_{I,2}, \Delta_{I,2})$, first sends $\pi_{I,1}, \pi_{I,2}$ to the verifier and then provides an argument of knowledge of (a_I, b_I) such that $\Gamma_{I,1}^{a_I} = A_1 \wedge \Delta_{I,1}^{b_I} = B_1 \wedge g^{a_I \cdot b_I} = U_I \wedge \Gamma_{I,2}^{a_I} = A_2 \wedge \Delta_{I,2}^{b_I} = B_2$, where $U_I \leftarrow g^{u_I}$ and $u_I \leftarrow \mathsf{PrimeProd}(I)$.

6.6.4 A Succinct AoK for Commitment on Subvector

Here we present a protocol which succinctly proves that a commitment C' opens to an I-subvector $\boldsymbol{v}_I$ of the opening $\boldsymbol{v}$ of another commitment C. Since C' is a vector commitment $\boldsymbol{v}_I$ should be a normal vector instead of a general subvector, i.e. I should be a set of consecutive positions starting from 1, $I = \{1, \ldots, n'\}$ for some $n' \in \mathbb{N}$. We note though that both commitments should share the same pp (but not the same specialized CRS). Below is the relation of the

[44] The notion of PoR with non-black-box extractability is close to that of robust proof of data possession [10, 9].

AoK that is parametrized by the two specialized CRSs $\mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$ and $\mathsf{pp}_{n'} \leftarrow \mathsf{Specialize}(\mathsf{pp}, n')$ where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{M})$ is common.

$$\begin{aligned} R_{\mathsf{PoKSubV}} \quad = \quad & \{(\,(C, C', I), (\boldsymbol{v}_I, \pi_I, \pi'_I)\,) : \mathsf{Ver}^\star(\mathsf{pp}_n, C, I, \boldsymbol{v}_I, \pi_I) = 1 \\ & \wedge\, \mathsf{Ver}^\star(\mathsf{pp}_{n'}, C', I, \boldsymbol{v}_I, \pi'_I) = 1 \wedge |\boldsymbol{v}_I| = n'\} \end{aligned}$$

The idea of our protocol is that since the opening $\boldsymbol{v}_I$ is the I-subvector of $\boldsymbol{v}$ one can provide a succinct proof of knowledge of the opening at these positions using the PoKOpen protocol presented above. However this is not enough as one should bind the opening proof with C'. This concretely can happen if one embeds a proof of product for the two components, A' and B', of C' inside the proof of opening. More specifically the prover provides an opening proof $\pi_I := (\Gamma_I, \Delta_I)$ then computes $(a_I, b_I) \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{v}_I)$ and proves that $g_0^{a_I} = A' \wedge g_1^{b_I} = B' \wedge U_{n'} = g^{a_I \cdot b_I} \wedge \Gamma_I^{a_I} = A \wedge \Delta_I^{b_I} = B$. Notice that the last three equalities correspond to the proof of opening protocol and the first three to the proof of product. So a conjunction of PoKOpen and PoProd protocol is sufficient. Lastly g, g_0, g_1 and $U_{n'}$ are part of $\mathsf{pp}_{n'}$ and (A, B), (A', B') part of the C and C' commitments respectively.

Prover input: $((\mathsf{pp}_n, \mathsf{pp}_{n'}), (C, C', I), (\boldsymbol{v}_I, \pi_I)$. Verifier input: $((\mathsf{pp}_n, \mathsf{pp}_{n'}), (C, C', I))$.

$\mathsf{P} \rightarrow \mathsf{V}$: $\pi_I := (\Gamma_I, \Delta_I)$

A conjuction of PoProd* and PoKOpen protocols between $\mathsf{P}(\mathsf{pp}_n, \mathsf{pp}_{n'}, (C, C', I), (\boldsymbol{v}_I, \pi_I))$ and $\mathsf{V}(\mathsf{pp}_n, \mathsf{pp}_{n'}, (C, C', I))$ is executed:

$\mathsf{V} \rightarrow \mathsf{P}$: $h \leftarrow_\$ \mathbb{G}$

$\mathsf{P} \rightarrow \mathsf{V}$: $z := (z_a, z_b)$ computed as $z_a \leftarrow h^{a_I}, z_b \leftarrow h^{b_I}$

$\mathsf{V} \rightarrow \mathsf{P}$: $\ell \leftarrow_\$ \mathbb{P}(1, 2^\lambda)$ and $\alpha \leftarrow_\$ [0, 2^\lambda)$

$\mathsf{P} \rightarrow \mathsf{V}$: $\pi := ((Q_A, Q_B, Q'_A, Q'_B, Q_C), r_a, r_b)$ computed as follows

- $(q_a, q_b, q_{ab}) \leftarrow (\lfloor a_I/\ell \rfloor, \lfloor b_I/\ell \rfloor, \lfloor a_I b_I/\ell \rfloor)$
- $(r_a, r_b) \leftarrow (a_I \mod \ell, b_I \mod \ell)$
- $(Q_{A'}, Q_{B'}, Q_A, Q_B, Q_C) := \left(g_0^{q_a}, g_1^{q_b}, \Gamma_I^{q_a} h^{\alpha q_a}, \Delta_I^{q_b} h^{\alpha q_b}, g^{q_{ab}}\right)$

V: Parse $\mathsf{pp}_n := (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen}, U_n)$, $\mathsf{pp}_{n'} := (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen}, U_{n'})$ and $C := (\{A, B\}, \pi_{\mathsf{prod}})$.

- Compute $r_c \leftarrow r_a \cdot r_b \mod \ell$
- Output 1 iff $r_a, r_b \in [\ell] \;\wedge\; Q_{A'}^\ell g_0^{r_a} = A' \;\wedge\; Q_{B'}^\ell g_1^{r_b} = B' \;\wedge\; Q_A^\ell \Gamma_I^{r_a} h^{\alpha r_a} = A z_a^\alpha \;\wedge\; Q_B^\ell \Delta_I^{r_b} h^{\alpha r_b} = B z_b^\alpha \;\wedge\; Q_C^\ell g^{r_c} = U_{n'} \wedge \mathsf{PoProd}_2.\mathsf{V}(\mathsf{pp}, (A \cdot B, U_n), \pi_{\mathsf{prod}})$

Figure 6.6: PoKSubV protocol

We state the following theorem for the security of the protocol above.

Theorem 25. *If* PoProd* *and* PoKOpen *are succinct arguments of knowledge for* R_{PoProd^*} *and* R_{PoKOpen}*, then protocol* PoKSubV *in Fig. 6.6 is a succinct argument of knowledge for relation* R_{PoKSubV} *with respect to algorithm* Ver *of our construction of Section 6.5.1.*

All Participating Nodes	
	Storage Nodes
Store current digest.	**Store** a portion of the file.
Can retrieve blocks of the file and verify responses.	**Can answer and certify** retrievals of subportions.
Can aggregate proofs they received.	**Can produce and publish updates** to their view.
Can update the digest following updates from other nodes	**Can apply updates** from other nodes efficiently.

Table 6.2: Roles in a decentralized verifiable database.

The intuition of the proof is that one proves knowledge of an opening I for C, namely that $\boldsymbol{v}_I$ is an I-subvector of C, where $(a_I, b_I) \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{v}_I)$, with a normal proof of subvector opening. This is equivalent to $\mathsf{Ver}^\star(\mathsf{pp}_n, C, I, \boldsymbol{v}_I, \pi_I) = 1$. Then in the same proof proves that the accumulators of C' are composed by the same (a_I, b_I) which results to proving that C' commits to $\boldsymbol{v}_I$. The last point is equivalent to $\mathsf{Ver}^\star(\mathsf{pp}_{n'}, C', I, \boldsymbol{v}_I, \pi'_I) = 1 \wedge |\boldsymbol{v}_I| = n'$.

6.7 Verifiable Decentralized Storage

In this section we introduce verifiable decentralized storage (VDS). We recall that in VDS there are two types of parties (called nodes): the generic *client nodes* and the more specialized *storage nodes* (a storage node can also act as a client node). The main goal of client nodes is to retrieve some blocks (i.e., a portion) of a given file. The role of a storage node is instead to store a portion of a file (or more files) and to answer to the retrieval queries of clients that are relevant to the portion it stores. In terms of security, VDS guarantees that malicious storage nodes cannot send to the clients blocks of the file that have been tampered with.

In Table 6.2 we summarize the main roles/capabilities of VDS nodes.

6.7.1 Syntax

Here we introduce the syntax of VDS. A VDS scheme is defined by a collection of algorithms that are to be executed by either storage nodes or client nodes. The only exception is the Bootstrap algorithm that is used to bootstrap the entire system and is assumed to be executed by a trusted party, or to be implemented in a distributed fashion (which is easy if it is public coin).

The syntax of VDS reflects its goal: guaranteeing data integrity in a highly dynamic and decentralized setting (the file can change and expend/shrink often and no single node stores it all). In VDS we create both parameters and an initial commitment for an empty file at the beginning (through the probabilistic Bootstrap algorithm, which requires a trusted execution). From then on this commitment is changed through incremental updates (of arbitrary size). Updating is divided in two parts. A node can carry out an update it and "push" it to all the other nodes, i.e. providing auxiliary information (that we call "update hint") other nodes can use to update their local certificates (if affected by the change) and a new digest[45]. These operations are done respectively trough StrgNode.PushUpdate and StrgNode.ApplyUpdate. Opening and verifying are where VC (with incremental aggregation) and VDS share the same mechanism. To respond to a query, a storage node can produce (possibly partial) proofs of opening via the

[45] One can also see this update hint as a certificate to check that a new digest is consistent with some changes. This issue does not arise in our context at all but the Bootstrap algorithms are deterministic.

StrgNode.Retrieve algorithm.. If these proofs need to be aggregated, any node can use algorithm AggregateCertificates. Anyone can verify a proof through ClntNode.VerRetrieve.

In VDS we model the files to be stored as vectors in some message space $\mathcal{M}$ (e.g., $\mathcal{M} = \{0,1\}$ or $\{0,1\}^{\ell}$), i.e., $\mathsf{F} = (\mathsf{F}_1, \ldots, \mathsf{F}_N)$. Given a file F, we define a *portion* of it as a pair (I, F_I) where F_I is essentially the I-subvector of F.

Definition 30 (Verifiable Decentralized Storage)**.** *Algorithm to bootstrap the system:*

$\mathsf{Bootstrap}(1^{\lambda}) \to (\mathsf{pp}, \delta_0, \mathsf{st}_0)$ *Given the security parameter λ, the probabilistic bootstrap algorithm outputs public parameters* pp*, initial digest δ_0 and state st_0. δ_0 and st_0 correspond to the digest and storage node's local state respectively for an empty file.*

All the algorithms below implicitly take as input the public parameters pp*.*

The algorithms for storage nodes are:

$\mathsf{StrgNode.AddStorage}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, Q, \mathsf{F}_Q, \pi_Q) \to (\mathsf{st}', J, \mathsf{F}_J)$ *This algorithm allows a storage node to add more blocks of a given file* F *to its local storage. Its first inputs are the local view of the storage node that is defined by a digest δ, a length n, a state* st*, and a file portion (I, F_I). Then it takes as input a file subportion (Q, F_Q) together with a valid retrieval certificate π_Q. The output is an updated view of the storage node, that is a new state st' and file portion $(J, \mathsf{F}_J) := (I, \mathsf{F}_I) \cup (Q, \mathsf{F}_Q)$.*

Note that this algorithm can be used to enable anyone who holds a valid retrieval certificate for a file portion F_Q to become a storage node of such portion.

$\mathsf{StrgNode.RmvStorage}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, K) \to (\mathsf{st}', J, \mathsf{F}_J)$ *This algorithm allows a storage node to remove blocks of a given file* F *from its local storage. Its first inputs are the local view of the storage node that is defined by a digest δ, a length n, a state* st*, and a file portion (I, F_I). Then it takes as input a set of positions $K \subseteq I$, and the output is an updated view of the storage node, that is a new state st' and file portion $(J, \mathsf{F}_J) := (I, \mathsf{F}_I) \setminus (K, \cdot)$.*

$\mathsf{StrgNode.CreateFrom}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, J) \to (\delta', n', \mathsf{st}', J, \mathsf{F}_J, \Upsilon_J)$ *This algorithm allows a storage node for a file subportion F_I to create a new file containing only a subset F_J of F_I along with the corresponding digest δ' and length n' and a hint to help other nodes generate their own digest. The algorithm takes as input the local view of the storage node, i.e., digest δ, length n, local state* st *and file portion (I, F_I), and a set of indices $J \subseteq I$. The algorithm returns a new digest δ', length n', a local state st', a file portion (J, F_J) and an advice Υ. This advice can be used by a client holding only the former digest δ to obtain the new digest δ', by using the* ClntNode.GetCreate *algorithm described below.*

$\mathsf{StrgNode.PushUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta) \to (\delta', n', \mathsf{st}', J, \mathsf{F}'_J, \Upsilon_\Delta)$ *This algorithm allow a storage node of a file subportion F_I to perform an update on the file and to generate a corresponding digest, length and local view, along with a hint other nodes can use to accordingly update their digests and local views. The inputs include the local view of the storage node, i.e., digest δ, length n, local state* st *and file portion (I, F_I), an update operation $\mathsf{op} \in \{\mathsf{mod}, \mathsf{add}, \mathsf{del}\}$ and an update description Δ. The outputs are a new digest δ' and length n', a new local state st', an updated file portion (J, F'_J) and an update hint Υ_Δ. If $\mathsf{op} = \mathsf{mod}$, then Δ contains a file portion (K, F'_K) such that $K \subseteq I$ and F'_K represents the new content to be written in positions K. If $\mathsf{op} = \mathsf{add}$, it is also $\Delta = (K, \mathsf{F}'_K)$ except that K is a set of new (sequential) positions $K \cap I = \emptyset$ that start from $n+1$ (and end to $n + |K|$). If $\mathsf{op} = \mathsf{del}$, then*

Δ only contains a set of positions $K \subseteq I$, which are the ones to be deleted (and are ought to be the $|K|$ last sequential positions). The proof Υ_Δ can be used by client nodes holding δ in order to check the validity of the new digest δ', and by other storage nodes, holding additionally the length n, in order to check the validity of the changes and to update their local views accordingly.

$\mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta, \Upsilon_\Delta) \rightarrow (b, \delta', n', \mathsf{st}', J, \mathsf{F}'_J)$ *This algorithm allows a storage node to incorporate changes in a file pushed by another node. The inputs include the local view of the storage node, i.e., digest δ, length n, local state st and file portion (I, F_I), an update operation $\mathsf{op} \in \{\mathsf{mod}, \mathsf{add}, \mathsf{del}\}$, an update description Δ and an update hint Υ_Δ. The algorithm returns a bit b (to accept/reject the update) and (if $b = 1$) a new digest δ', a new length n', a new (local) state st' and an updated file subportion (J, F'_J). If $\mathsf{op} \in \{\mathsf{mod}, \mathsf{add}\}$ we have that $J = I$, i.e., the node keeps storing the same indices; if $\mathsf{op} = \mathsf{del}$ then J is I minus the deleted indices.*

$\mathsf{StrgNode.Retrieve}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, Q) \rightarrow (\mathsf{F}_Q, \pi_Q)$ *This algorithm allows a storage node to answer a retrieval query for blocks with indices in Q and to create a certificate vouching for the correctness of the returned blocks. The inputs include the local view of the storage node, i.e., digest δ, length n local state st and file portion (I, F_I), and a set of indices Q. The output is a file portion F_Q and a retrieval certificate π_Q.*

The algorithms for clients nodes are:

$\mathsf{ClntNode.GetCreate}(\delta, J, \Upsilon_J) \rightarrow (b, \delta')$ *On input a digest δ, a set of indices J and a creation advice Υ_J, this algorithm returns a bit b (to accept/reject) and (if $b = 1$) a new digest δ' that corresponds to a file F' that is the prefix with indices J of the file represented by digest δ.*

$\mathsf{ClntNode.ApplyUpdate}(\delta, \mathsf{op}, \Delta, \Upsilon_\Delta) \rightarrow (b, \delta')$ *On input a digest δ, an update operation $\mathsf{op} \in \{\mathsf{mod}, \mathsf{add}, \mathsf{del}\}$, an update description Δ and an update hint Υ_Δ, it returns a bit b (to accept/reject update) and (if $b = 1$) a new digest δ'.*

$\mathsf{ClntNode.VerRetrieve}(\delta, Q, \mathsf{F}_Q, \pi_Q) \rightarrow b$ *On input a digest δ, a file portion (Q, F_Q) and a certificate π_Q, this algorithm accepts (i.e. it outputs 1) only if π_Q is a valid proof that δ corresponds to a file F with length n of which F_Q is the portion corresponding to indices Q.*

$\mathsf{AggregateCertificates}(\delta, (I, \mathsf{F}_I, \pi_I), (J, \mathsf{F}_J, \pi_J)) \rightarrow \pi_K$ *On input a digest δ and two certificated retrieval outputs (I, F_I, π_I) and (J, F_J, π_J), this algorithm aggregates their certificates into a single certificate π_K (with $K := I \cup J$). In a running VDS system, this algorithm can be used by any node to aggregate two (or more) incoming certified data blocks into a single certified data block.*

Remark 19 (On CreateFrom)**.** *For completeness, our VDS syntax also includes the functionalitis* $(\mathsf{StrgNode.CreateFrom}, \mathsf{ClntNode.GetCreate})$ *that allow a storage node to initialize storage (and corresponding digest) for a new file that is a subset of an existing one, and a client node to verify such resulting digest. Although this feature can be interesting in some application scenarios (see the Introduction), we still see it as an extra feature that may or may not be satisfied by a VDS construction.*

6.7.2 Correctness and Efficiency of VDS

Intuitively, we say that a VDS scheme is *efficient* if running VDS has a "small" overhead in terms of the storage required by all the nodes and the bandwidth to transmit certificates. More formally,

a VDS scheme is said efficient if there is a fixed polynomial $p(\cdot)$ such that $p(\lambda, \log n)$ (with λ the security parameter and n the length of the file) is a bound for all certificates and advices generated by the VDS algorithms as well as for digests δ and the local state st of storage nodes. Note that combining this bound with the requirement that all algorithms are polynomial time in their input, we also get that no VDS algorithm can run linearly in the size of the file (except in the trivial case that the file is processed in one shot, e.g., in the first StrgNode.AddStorage).

Efficiency essentially models that running VDS is cost-effective for all the nodes in the sense that it does not require them to store significantly more data then they would have to store without. Notice that by requiring certificates to have a fixed size implies that they do not grow with aggregation.

For correctness, intuitively speaking, we want that for any (valid) evolution of the system in which the VDS algorithms are honestly executed we get that any storage node storing a portion of a file F can successfully convince a client holding a digest of F about retrieval of any portion of F. And such (intuitive notion of) correctness is also preserved when updates, aggregations, or creations of new files are done.

Turning this intuition into a formal correctness definition turned out to be nontrivial. This is due to the distributed nature of this primitive and the fact that there could be many possible ways in which, at the time of answering a retrieval query, a storage node may have reached its state starting from the empty node state. The basic idea of our definition is that an empty node is "valid", and then any "valid" storage node that runs StrgNode.PushUpdate "transfers" such validity to both itself and to other nodes that apply such update. A bit more precisely, we model "validity" as the ability to correctly certify retrievals of any subsets of the stored portion. A formal definition correctness follows. To begin with, we define the notion of validity for the view of a storage node.

Definition 31 (Validity of storage node's view)**.** *Let* pp *be public parameters as generated by* Bootstrap*. We say that a local view* $(\delta, n, \mathsf{st}, I, \mathsf{F}_I)$ *of a storage node is* valid *if* $\forall Q \subseteq I$*:*

$$\mathsf{CIntNode.VerRetrieve}(\delta, Q, \mathsf{F}_Q, \pi_Q) = 1$$

where $(\mathsf{F}_Q, \pi_Q) \leftarrow \mathsf{StrgNode.Retrieve}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, Q)$

Remark 20. *By Definition 31 the output of a bootstrapping algorithm* $(\mathsf{pp}, \delta_0, \mathsf{st}_0) \leftarrow \mathsf{Bootstrap}(1^\lambda)$ *is always such that* $(\mathsf{pp}, \delta_0, 0, \mathsf{st}_0, \emptyset, \emptyset)$ *is valid. This provides a "base case" for Definition 33.*

Second, we define the notion of admissible update, which intuitively models when a given update can be meaningfully processed, locally, by a storage node.

Definition 32 (Admissible Update)**.** *An update* (op, Δ) *is* admissible *for* (n, I, F_I) *if:*

- *for* $\mathsf{op} = \mathsf{mod}$*,* $K \subseteq I$ *and* $|\mathsf{F}'_K| = |K|$*, where* $\Delta := (K, \mathsf{F}'_K)$*.*
- *for* $\mathsf{op} = \mathsf{add}$*,* $K \cap I = \emptyset$ *and* $|\mathsf{F}'_K| = |K|$ *and* $K = \{n+1, n+2, \ldots, n+|K|\}$*, where* $\Delta := (K, \mathsf{F}'_K)$*.*
- *for* $\mathsf{op} = \mathsf{del}$*,* $K \subseteq I$ *and* $K = \{n - |K| + 1, \ldots, n\}$*, where* $\Delta := K$*.*

In words, the above definition formalizes that: to push a modification at positions K, the storage node must store those positions; to push an addition, the new positions K must extend

the currently stored length of the file; to push a deletion of position K, the storage node must store data of the positions to be deleted and those positions must also be the last $|K|$ positions of the currently stored file (i.e., the file length is reduced).

Definition 33 (Correctness of VDS)**.** *A VDS scheme* VDS *is* correct *if for all honestly generated parameters* $(\mathsf{pp}, \delta_0, \mathsf{st}_0) \leftarrow \mathsf{Bootstrap}(1^\lambda)$ *and any storage node's local view* $(\delta, n, \mathsf{st}, I, \mathsf{F}_I)$ *that is valid, the following conditions hold.*
Update Correctness. *For any update* (op, Δ) *that is admissible for* (n, I, F_I) *and for any* $(\delta', n', \mathsf{st}', J, \mathsf{F}'_J, \Upsilon_\Delta) \leftarrow \mathsf{StrgNode.PushUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta)$*:*

1. $(\mathsf{pp}, \delta', n', \mathsf{st}', J, \mathsf{F}'_J)$ *is valid;*
2. *for any valid* $(\delta, n, \mathsf{st}_s, I_s, \mathsf{F}_{I_s})$*, if* $(b_s, \delta'_s, n', \mathsf{st}'_s, I'_s, \mathsf{F}'_s) \leftarrow \mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}_s, I_s, \mathsf{F}_{I_s}, \mathsf{op}, \Delta, \Upsilon_\Delta)$ *then we have:* $b_s = 1$*,* $\delta'_s = \delta'$*,* $n'_s = n'$*, and* $(\delta'_s, n'_s, \mathsf{st}'_s, I'_s, \mathsf{F}'_s)$ *is valid;*
3. *if* $(b_c, \delta'_c) \leftarrow \mathsf{ClntNode.ApplyUpdate}(\delta, \mathsf{op}, \Delta, \Upsilon_\Delta)$*, then* $\delta'_c = \delta'$ *and* $b_c = 1$*.*

Add-Storage Correctness. *For any* (Q, F_Q, π_Q) *such that*
$\mathsf{ClntNode.VerRetrieve}(\delta, Q, \mathsf{F}_Q, \pi_Q) = 1$*, if* $(\mathsf{st}', J, \mathsf{F}_J) \leftarrow \mathsf{StrgNode.AddStorage}(\delta, \mathsf{st}, I, \mathsf{F}, Q, \mathsf{F}_Q, \pi_Q)$ *then* $(\delta, n, \mathsf{st}', J, \mathsf{F}_J)$ *is valid.*
Remove-Storage Correctness. *For any* $K \subseteq I$*,*
if $(\mathsf{st}', J, \mathsf{F}_J) \leftarrow \mathsf{StrgNode.RmvStorage}(\delta, \mathsf{st}, I, \mathsf{F}, K)$ *then* $(\delta, n, \mathsf{st}', J, \mathsf{F}_J)$ *is valid.*
Create Correctness. *For any* $J \subseteq I$*, if* $(\delta', n', \mathsf{st}', J, \mathsf{F}_J, \Upsilon_J)$ *is output of* $\mathsf{StrgNode.CreateFrom}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, J)$ *and* $(b, \delta'') \leftarrow \mathsf{ClntNode.GetCreate}(\delta, J, \Upsilon_J)$*, then* $b = 1$*,* $n' = |J|$*,* $\delta'' = \delta'$ *and* $(\mathsf{pp}, \delta', n', \mathsf{st}', J, \mathsf{F}_J)$ *is valid.*
Aggregate Correctness. *For any pair of triples* (I, F_I, π_I) *and* (J, F_J, π_J) *such that*
$\mathsf{ClntNode.VerRetrieve}(\delta, I, \mathsf{F}_I, \pi_I) = 1$ *and* $\mathsf{ClntNode.VerRetrieve}(\delta, J, \mathsf{F}_J, \pi_J) = 1$*,*
if $\pi_K \leftarrow \mathsf{AggregateCertificates}((I, \mathsf{F}_I, \pi_I), (J, \mathsf{F}_J, \pi_J))$ *and* $(K, \mathsf{F}_K) := (I, \mathsf{F}_I) \cup (J, \mathsf{F}_J)$*, then*
$\mathsf{ClntNode.VerRetrieve}(\delta, K, \mathsf{F}_K, \pi_K) = 1$*.*

Remark 21 (Relation with Updatable VCs)**.** *Our notion of VDS is very close to the notion of updatable VCs [66] extended to support subvector openings and incremental aggregation. On a syntactical level, in comparison to updatable VCs, our VDS notion makes more evident the decentralized nature of the primitive, which is reflected in the definition of our algorithms where for example it is clear that no one ever needs to store/know the entire file. One major difference is that in VDS the public parameters must necessarily be* short *since no node can run linearly in the size of the file (nor it can afford such storage), whereas in VCs this may not be necessarily the case. Another difference is that in updatable VCs [66] updates can be received without any hint, which is instead the case in VDS. Finally, it is interesting to note that, as of today, there exists no VC scheme that is updatable, incrementally aggregatable and with subvector openings, that enjoys short parameters and has the required short verification time. So, in a way, our two VDS realizations show how to bypass this barrier of updatable VC by moving to a slightly different (and practically motivated) model.*

6.7.3 Security of VDS

In this section we define the security of VDS schemes. Intuitively speaking, we require that a malicious storage node (or a coalition of them) cannot convince a client of a false data block

in a retrieval query. To formalize this, we let the adversary fully choose a *history* of the VDS system that starts from the empty state and consists of a sequence of steps, where each step is either an update (addition, deletion, modification) or a creation (from an existing file) and is accompanied by an advice. A client's digest δ is updated following such history and using the adversarial advices, and similarly one gets a file F corresponding to such digest. At this point, the adversary's goal is to provide a tuple $(Q, \pi_Q, \mathsf{F}^*_Q)$ that is accepted by a client with digest δ but where $\mathsf{F}^*_Q \neq \mathsf{F}_Q$.

Definition 34 (History for Decentralized Storage)**.** *Let* VDS *be a verifiable decentralized storage scheme. A history for* VDS *is a sequence* $\mathcal{H} = (\mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)_{i\in[\ell]}$ *of tuples, where* op^i *is either in* $\{\mathsf{mod}, \mathsf{add}, \mathsf{del}\}$ *(i.e., it is an update of the file), or* $\mathsf{op}^i = \mathsf{cfrom}$ *(i.e., it is the creation of a new file related to the current one), in which case* Δ^i *is a set of indices. In order to define valid histories we define the function* $\mathsf{EvalHistory}(\mathsf{pp}, \delta_0, \mathsf{st}_0, \mathcal{H})$ *as follows*

$\mathsf{EvalHistory}(\mathsf{pp}, \delta_0, \mathsf{st}_0, \mathcal{H})$

$\mathsf{F}_0 \leftarrow \emptyset; b \leftarrow 1$
for $i \in [\ell]$
  $\mathsf{F}_i \leftarrow \mathsf{FileChange}(\mathsf{F}_{i-1}, \mathsf{op}^i, \Delta^i)$
  if $\mathsf{op}^i \in \{\mathsf{mod}, \mathsf{add}, \mathsf{del}\}$ **then**
    $(b_i, \delta_i) \leftarrow \mathsf{ClntNode.ApplyUpdate}($
      $\delta_{i-1}, \mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)$
  elseif $\mathsf{op}^i = \mathsf{cfrom}$ **then**
    $(b_i, \delta_i) \leftarrow \mathsf{ClntNode.GetCreate}($
      $\delta_{i-1}, \Delta^i, \Upsilon^i_\Delta)$
  endif
  $b \leftarrow b \wedge b_i$
endfor
return $(b, \delta_\ell, \mathsf{F}_\ell)$

$\mathsf{FileChange}(\mathsf{F}, \mathsf{op}, \Delta)$

if $\mathsf{op} \in \{\mathsf{mod}, \mathsf{add}\}$ *parse* $\Delta = (K, \mathsf{F}'_K)$
  $\forall i \in K : \mathsf{F}^*_i \leftarrow \mathsf{F}'_i; \forall i \in [|\mathsf{F}|] \setminus K : \mathsf{F}^*_i \leftarrow \mathsf{F}_i,$
elseif $\mathsf{op} = \mathsf{del}$ *parse* $\Delta = K$
  $\forall i \in [|\mathsf{F}|] \setminus K : \mathsf{F}^*_i \leftarrow \mathsf{F}_i,$
elseif $\mathsf{op} = \mathsf{cfrom}$ *parse* $\Delta = K$
  $\forall i \in K : \mathsf{F}^*_i \leftarrow \mathsf{F}_i,$
endif return F^*

We say that a history $\mathcal{H}$ *is valid w.r.t. public parameters* pp *and initial digest* δ_0 *and state* st_0 *if* $\mathsf{EvalHistory}(\mathsf{pp}, \delta_0, \mathsf{st}_0, \mathcal{H})$ *returns bit* $b = 1$.

Definition 35 (Security for Verifiable Decentralized Storage)**.** *Consider the experiment* $\mathsf{VDS\text{-}Security}^{\mathcal{A}}_{\mathsf{VDS}}(\lambda)$ *below. Then we say that a VDS scheme* VDS *is secure if for all PPT* $\mathcal{A}$ *we have* $\Pr[\mathsf{VDS\text{-}Security}^{\mathcal{A}}_{\mathsf{VDS}}(\lambda) = 1] \in \mathsf{negl}(\lambda)$.

$\mathsf{VDS\text{-}Security}^{\mathcal{A}}_{\mathsf{VDS}}(\lambda)$

$(\mathsf{pp}, \delta_0, \mathsf{st}_0) \leftarrow \mathsf{Bootstrap}(1^\lambda)$
$(\mathcal{H}, Q, \mathsf{F}^*_Q, \pi^*) \leftarrow \mathcal{A}(\mathsf{pp}, \delta_0, \mathsf{st}_0)$
$(b, \delta, \mathsf{F}) \leftarrow \mathsf{EvalHistory}(\mathsf{pp}, \delta_0, \mathsf{st}_0, \mathcal{H})$
$b \leftarrow b \wedge \mathsf{F}^*_Q \neq \mathsf{F}_Q \wedge$
  $\mathsf{ClntNode.VerRetrieve}(\mathsf{pp}, \delta, Q, \mathsf{F}^*_Q, \pi^*)$
return b

6.8 Our Realizations of VDS in Hidden-Order Groups

In this section, we present two constructions of VDS that work in hidden-order groups. The two schemes are presented in Sections 6.8.1 and 6.8.2 respectively, and we discuss a comparison in Section 6.8.3.

6.8.1 Our First VDS Construction

We build our first scheme by extending the techniques used to construct our first SVC scheme from Section 6.5.1. In particular, we start from a modified version of our SVC that achieves a weaker position binding property (in which the adversary reveals the full vector, yet its goal is to find two distinct openings for the same position) and then show how to make this scheme dynamic (i.e., to change vector values or its length) and fully distributed (i.e., updates can be performed without knowing the entire vector).

Preliminaries. We begin by describing the simplified version of our SVC, considering the case of $k = 1$, which fits best our VDS construction, regarding efficiency and communication complexity. For convenience of the reader we describe again shortly the algorithms and functions (and variations of them) from section 6.5.1 that are used in the scheme (for more details we refer to the corresponding section):

- $\mathsf{PrimeGen}$, a deterministic collision resistant function that maps integers to primes.
- $\mathsf{PartndPrimeProd}(I, \boldsymbol{y}) \to (a_I, b_I)$: given a set of indices $I = \{i_1, \ldots, i_m\} \subseteq [n]$ and a vector $\boldsymbol{y} \in \mathcal{M}^m$, the function computes $(a_I, b_I) := \left(\prod_{l=1:y_l=0}^{m} p_{i_l}, \prod_{l=1:y_l=1}^{m} p_{i_l}\right)$, where $p_i \leftarrow \mathsf{PrimeGen}(i)$ for all $i \in \mathbb{N}$.

$\mathsf{Setup}(1^\lambda, \{0,1\}^k) \to \mathsf{pp} := (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen})$.

$\mathsf{Com}'(\mathsf{pp}, \boldsymbol{v}) \to C$ compute $(a, b) \leftarrow \mathsf{PartndPrimeProd}([n], \boldsymbol{v})$, where $n \leftarrow |\boldsymbol{v}|$; next compute $A = g_0^a$ and $B = g_1^b$. Return $C := (C^\star, n) := ((A, B), |\boldsymbol{v}|)$.

$\mathsf{Ver}'(\mathsf{pp}, C, I, \boldsymbol{y}, \pi_I) \to b$ compute $(a_I, b_I) \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{y})$, and then parse $\pi_I := (\Gamma_I, \Delta_I)$ and return $b \leftarrow (\Gamma_I^{a_I} = A) \wedge (\Delta_I^{b_I} = B)$.

$\mathsf{VC.Disagg}'(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, K) \to \pi_K$ let $L := I \setminus K$, and $\boldsymbol{v}_L$ be the subvector of $\boldsymbol{v}_I$ at positions in L. Then compute $a_L, b_L \leftarrow \mathsf{PartndPrimeProd}(L, \boldsymbol{v}_L)$ parse $\pi_I := (\Gamma_I, \Delta_I)$ and set $(\Gamma_K, \Delta_K) \leftarrow (\Gamma_I^{a_L}, \Delta_I^{b_L})$. Return $\pi_K \leftarrow (\Gamma_K, \Delta_K)$.

$\mathsf{VC.Agg}'(\mathsf{pp}, (I, \boldsymbol{v}_I, \pi_I), (J, \boldsymbol{v}_J, \pi_J)) \to \pi_K$:

1. Let $L := I \cap J$. If $L \neq \emptyset$, set $I' := I \setminus L$ and compute $\pi_{I'} \leftarrow \mathsf{VC.Disagg}(\mathsf{pp}, I, \boldsymbol{v}_I, \pi_I, I')$; otherwise let $\pi_{I'} = \pi_I$.
2. Compute $(a_{I'}, b_{I'}) \leftarrow \mathsf{PartndPrimeProd}(I, \boldsymbol{v}_{I'})$ and $\{a_J, b_J\} \leftarrow \mathsf{PartndPrimeProd}(J, \boldsymbol{v}_J)$.
3. Parse $\pi_{I'} := (\Gamma_{I'}, \Delta_{I'})$, $\pi_J := (\Gamma_J, \Delta_J)$ and compute $\Gamma_K \leftarrow \mathbf{ShamirTrick}(\Gamma_{I'}, \Gamma_J, a_{I'}, a_J)$ and $\Delta_K \leftarrow \mathbf{ShamirTrick}(\Delta_{I'}, \Delta_J, b_{I'}, b_J)$
4. Return $\pi_K \leftarrow (\Gamma_K, \Delta_K)$

Finally, let $\mathsf{PoKSubV}'$ be the same protocol as in section 6.6 but adjusted according to the above algorithms. That is the CRS of is simply pp instead of the two specialized CRSs. Furthermore,

since C is not accompanied with PoProd_2 the verifier does not have to check the validity of it. The rest of the protocol remains the same and the underlying relation is:

$$\begin{aligned} R_{\mathsf{PoKSubV}'} \quad = \quad & \{((C, C', I), (\boldsymbol{v}_I, \pi_I, \pi'_I)) : \mathsf{Ver}'(\mathsf{pp}, C, I, \boldsymbol{v}_I, \pi_I) = 1 \\ & \wedge \mathsf{Ver}'(\mathsf{pp}, C', I, \boldsymbol{v}_I, \pi'_I) = 1 \wedge |\boldsymbol{v}_I| = n' \} \end{aligned}$$

Finally, we note that for simplicity in the following we abuse the notation for Shamir's trick by writing e.g. $(\Gamma'_I, \Delta'_I) \leftarrow \left(\mathbf{ShamirTrick}(\Gamma_I, \Gamma_K, \mathsf{F}_I, \mathsf{F}_K)^{a'_K}\ ,\ \mathbf{ShamirTrick}(\Delta_I, \Delta_K, \mathsf{F}_I, \mathsf{F}_K)^{b'_K}\right)$ instead of writing, more precisely,

$$(\Gamma'_I, \Delta'_I) \leftarrow \left(\mathbf{ShamirTrick}(\Gamma_I, \Gamma_K, a_I, a_K)^{a'_K}\ ,\ \mathbf{ShamirTrick}(\Delta_I, \Delta_K, b_I, b_K)^{b'_K}\right).$$

Our scheme VDS_1**.** The algorithms of the VDS scheme VDS_1 are the following:[46]

$\mathsf{Bootstrap}(1^\lambda) \to (\mathsf{pp}, \delta_0, n_0, \mathsf{st}_0)$ Execute $\mathsf{Setup}(1^\lambda, \{0,1\}^k)$ and get $\mathsf{pp} := (\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen})$. Set $n_0 \leftarrow 0$, $\delta_0 \leftarrow ((g_0, g_1), n_0)$ and $\mathsf{st}_0 \leftarrow (g_0, g_1)$.

The algorithms for storage nodes are:

$\mathsf{StrgNode.AddStorage}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, Q, \mathsf{F}_Q, \pi_Q) \to (\mathsf{st}', J, \mathsf{F}_J)$ If $I = \emptyset$ then set $\mathsf{st}' \leftarrow \pi_Q$, otherwise $\mathsf{st} := \pi_I$. Then compute $\mathsf{st}' \leftarrow \mathsf{VC.Agg}'(\mathsf{pp}, (I, \mathsf{F}_I, \pi_I), (Q, \mathsf{F}_Q, \pi_Q))$. The computation of J and F_J is straightforward: $(J, \mathsf{F}_J) \leftarrow (I \cup Q, \mathsf{F}_I \cup \mathsf{F}_Q)$.

$\mathsf{StrgNode.RmvStorage}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, K) \to (\mathsf{st}', J, \mathsf{F}_J)$ Compute $J \leftarrow I \setminus K$ and the corresponding F_J. Then $\pi_J \leftarrow \mathsf{VC.Disagg}'(\mathsf{pp}, I, \mathsf{F}_I, \pi_I, J)$ and set $\mathsf{st}' \leftarrow \pi_J$.

$\mathsf{StrgNode.CreateFrom}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, J) \to (\delta', n', \mathsf{st}', J, \mathsf{F}_J, \Upsilon_J)$ The new digest δ' of F_J is computed with the commitment algorithm $\delta' \leftarrow \mathsf{Com}'(\mathsf{pp}, \mathsf{F}_J)$. The new length gets $n' \leftarrow |J|$. The previous local state is $\mathsf{st} = \pi_I$ and the new local state gets $\mathsf{st}' \leftarrow \mathsf{VC.Disagg}(\mathsf{pp}, I, \mathsf{F}_I, \pi_I, J)$. Finally, for Υ_J it computes an argument of knowledge of subvector (see section 6.6), $\pi_{\mathsf{PoKSubV}'} \leftarrow \mathsf{PoKSubV}'.\mathsf{P}(\mathsf{pp}, (\delta, \delta', J), (\boldsymbol{v}_J, \pi_I))$ and sets $\Upsilon_J \leftarrow (\delta', \pi_{\mathsf{PoKSubV}'})$.

$\mathsf{StrgNode.PushUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta) \to (\delta', n', \mathsf{st}', J, \mathsf{F}'_J, \Upsilon_\Delta)$ The algorithm works according to the type of update operation op:

- $\mathsf{op} = \mathsf{mod}$: parse $\Delta := (K, \mathsf{F}'_K)$ and $\mathsf{st} := \pi_I$. Execute $\pi_K \leftarrow \mathsf{VC.Disagg}'(\mathsf{pp}, I, \mathsf{F}_I, \pi_I, K)$ and parse $\pi_K := (\Gamma_K, \Delta_K)$. Then compute $(a'_K, b'_K) \leftarrow \mathsf{PartndPrimeProd}(K, \mathsf{F}'_K)$ and set $\delta' \leftarrow ((\Gamma_K^{a'_K}, \Delta_K^{b'_K}), n)$ (i.e., $n' = n$ remains the same). st' is the new opening of I, $\pi'_I \leftarrow \pi_I$, which is the same so the local state does not change $\mathsf{st}' \leftarrow \mathsf{st}$. Since it is a modification operation $(J, \mathsf{F}'_J) \leftarrow (I, \mathsf{F}'_I)$, where F'_I is simply the modified file $\mathsf{F}'_I = (\mathsf{F}_I \setminus \mathsf{F}_K) \cup \mathsf{F}'_K$. Finally, set $\Upsilon_\Delta \leftarrow (\mathsf{F}_K, \pi_K)$.
- $\mathsf{op} = \mathsf{add}$: parse $\Delta := (K, \mathsf{F}'_K)$, $\mathsf{st} := \pi_I$, and the old digest $\delta := ((A, B), n)$. Then compute $(a'_K, b'_K) \leftarrow \mathsf{PartndPrimeProd}(K, \mathsf{F}'_K)$ and the new digest gets $\delta' \leftarrow ((A^{a'_K}, B^{b'_K}), n')$ where $n' \leftarrow n + |K|$. The new state refers to the new file subportion $(J, \mathsf{F}'_J) \leftarrow (I \cup K, \mathsf{F}_I \cup \mathsf{F}_K)$, $\mathsf{st}' := \pi'_J$, and is the same as the old one $\mathsf{st}' \leftarrow \mathsf{st}$ since $\pi_I = \pi'_J$. Finally, set $\Upsilon_\Delta \leftarrow \varnothing$.

[46] Since the scheme has several parts in common with the above VC algorithms, we use those algorithms as shorthands in the description.

- $\mathsf{op} = \mathsf{del}$: parse $\Delta := K$ and $\mathsf{st} := \pi_I$. Execute $\pi_K \leftarrow \mathsf{VC.Disagg}'(\mathsf{pp}, I, \mathsf{F}_I, \pi_I, K)$ and parse $\pi_K := (\Gamma_K, \Delta_K)$. Then the new digest is $\delta' \leftarrow ((\Gamma_K, \Delta_K), n')$ where $n' \leftarrow n - |K|$. The new state refers to the new file subportion $(J, \mathsf{F}'_J) \leftarrow (I \setminus K, \mathsf{F}_I \setminus \mathsf{F}_K))$ and is the same as the old one $\mathsf{st}' \leftarrow \mathsf{st}$ since $\pi_I = \pi'_J$. Finally set $\Upsilon_\Delta \leftarrow (\mathsf{F}_K, \pi_K)$.

$\mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta, \Upsilon_\Delta) \rightarrow (b, \delta', n', \mathsf{st}', J, \mathsf{F}'_J)$ Again, it works according to the type of update operation op:

- $\mathsf{op} = \mathsf{mod}$: parse $\Delta := (K, \mathsf{F}'_K)$, $\mathsf{st} := \pi_I$ and $\Upsilon_\Delta := (\mathsf{F}_K, \pi_K)$. Compute acceptance bit $b \leftarrow \mathsf{Ver}'(\mathsf{pp}, \delta, K, \mathsf{F}_K, \pi_K)$. Then, if $b = 1$ parse $\pi_K := (\Gamma_K, \Delta_K)$, compute $(a'_K, b'_K) \leftarrow \mathsf{PartndPrimeProd}(K, \mathsf{F}'_K)$ and set $\delta' \leftarrow ((\Gamma_K^{a'_K}, \Delta_K^{b'_K}), n')$ where $n' \leftarrow n$. It is clear that in the case of a modify operation $(J, \mathsf{F}'_J) \leftarrow (I, \mathsf{F}'_I)$, where F'_I is simply the modified file $\mathsf{F}'_I = (\mathsf{F}_I \setminus \mathsf{F}_K) \cup \mathsf{F}'_K$. For the new local state st' that we discern three cases:
  - $I \cap K = \emptyset$: then compute $(\Gamma'_I, \Delta'_I) \leftarrow \left(\mathbf{ShamirTrick}(\Gamma_I, \Gamma_K, \mathsf{F}_I, \mathsf{F}_K)^{a'_K}, \mathbf{ShamirTrick}(\Delta_I, \Delta_K, \mathsf{F}_I, \mathsf{F}_K)^{b'_K}\right)$ and set $\mathsf{st}' \leftarrow \pi'_I := (\Gamma'_I, \Delta'_I)$.
  - $I \cap K = K$: compute $(\Gamma'_I, \Delta'_I) \leftarrow (\Gamma_I, \Delta_I)$ and set $\mathsf{st}' \leftarrow \pi'_I := (\Gamma'_I, \Delta'_I)$.
  - For the case where neither $I \cap K = \emptyset$ nor $I \cap K = K$, i.e. $I \cap K = L \notin \{K, \emptyset\}$ we partition K as $K = L \cup \bar{L}$ and apply two sequential updates to π_I, one with L' (s.t. $I \cap \bar{L} = \emptyset$) and one with L (s.t. $I \cap L = L$). That is, compute $(a'_{\bar{L}}, b'_{\bar{L}}) \leftarrow \mathsf{PartndPrimeProd}(\bar{L}, \mathsf{F}'_{\bar{L}})$ and then $(\Gamma'_I, \Delta'_I) \leftarrow \left(\mathbf{ShamirTrick}(\Gamma_I, \Gamma_{\bar{L}}, \mathsf{F}_I, \mathsf{F}_{\bar{L}})^{a'_{\bar{L}}}, \mathbf{ShamirTrick}(\Delta_I, \Delta_{\bar{L}}, \mathsf{F}_I, \mathsf{F}_{\bar{L}})^{b'_{\bar{L}}}\right)$. Then $(\Gamma''_I, \Delta''_I) \leftarrow (\Gamma'_I, \Delta'_I)$. Finally, set $\mathsf{st}' \leftarrow (\Gamma''_I, \Delta''_I)$. Essentially, since the case of $I \cap L = L$ doesn't cause any change to the state, computationally it is as a single update.
- $\mathsf{op} = \mathsf{add}$: parse $\Delta := (K, \mathsf{F}'_K)$, $\mathsf{st} := \pi_I$ and the old digest as $\delta := ((A, B), n)$. Set $b = 1$ iff $K = \{n+1, \ldots, n+|K|\}$. Then if $b = 1$ compute $(a'_K, b'_K) \leftarrow \mathsf{PartndPrimeProd}(K, \mathsf{F}'_K)$ and the new digest becomes $\delta' \leftarrow ((A^{a'_K}, B^{b'_K}), n')$ where $n' \leftarrow n + |K|$. For the new local state, first parse the old one $\mathsf{st} := \pi_I := (\Gamma_I, \Delta_I)$ and the new one gets $\mathsf{st}' \leftarrow \pi'_I$ where $\pi'_I \leftarrow (\Gamma_I^{a'_K}, \Delta_I^{b'_K})$. Finally set $(J, \mathsf{F}'_J) \leftarrow (I, \mathsf{F}_I)$, i.e., the file remains unchanged.
- $\mathsf{op} = \mathsf{del}$: parse $\Delta := K$, $\mathsf{st} := \pi_I$, and $\Upsilon_\Delta := (\mathsf{F}_K, \pi_K)$. Set $b = 1$ iff $K = \{n - |K| + 1, \ldots, n\} \wedge \mathsf{Ver}'(\mathsf{pp}, \delta, K, \mathsf{F}_K, \pi_K) = 1$. Then if $b = 1$ sets $\delta' \leftarrow ((\Gamma_K, \Delta_K), n')$ where $n' \leftarrow n - |K|$. For the new local state, similarly to the modify operation, we discern three cases. If $I \cap K = \emptyset$ then $(\Gamma'_I, \Delta'_I) \leftarrow (\mathbf{ShamirTrick}(\Gamma_I, \Gamma_K, \mathsf{F}_I, \mathsf{F}_K), \mathbf{ShamirTrick}(\Delta_I, \Delta_K, \mathsf{F}_I, \mathsf{F}_K))$ and set $\mathsf{st}' \leftarrow \pi_K := (\Gamma'_I, \Delta'_I)$; else if $I \cap K = K$ $\mathsf{st}' = \mathsf{st}$, else if $I \cap K = L$ then (let $\bar{L} = K \setminus L$) $(\Gamma'_I, \Delta'_I) \leftarrow (\mathbf{ShamirTrick}(\Gamma_I, \Gamma_{\bar{L}}, \mathsf{F}_I, \mathsf{F}_{\bar{L}}), \mathbf{ShamirTrick}(\Delta_I, \Delta_{\bar{L}}, \mathsf{F}_I, \mathsf{F}_{\bar{L}}))$ and set $\mathsf{st}' \leftarrow \pi_I := (\Gamma'_I, \Delta'_I)$ (similarly to the $\mathsf{op} = \mathsf{mod}$ case). Finally $(J, \mathsf{F}'_J) \leftarrow (I \setminus L, \mathsf{F}_I \setminus \mathsf{F}_L)$.

$\mathsf{StrgNode.Retrieve}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, Q) \rightarrow (\mathsf{F}_Q, \pi_Q)$ Compute both portion $\mathsf{F}_Q \subseteq \mathsf{F}_I$ as well as proof $\pi_Q \leftarrow \mathsf{VC.Disagg}'(\mathsf{pp}, I, \mathsf{F}_I, \mathsf{st}, Q)$.

The algorithms for client nodes are:

$\mathsf{ClntNode.GetCreate}(\delta, J, \Upsilon_J) \rightarrow (b, \delta')$ Parse $\Upsilon_J := (\delta', \pi_{\mathsf{PoKSubV}'})$, set $n' = |J|$ and output $b \leftarrow \mathsf{PoKSubV}'.\mathsf{V}(\mathsf{pp}, (\delta, \delta', J), \pi_J) \wedge J = \{1, \ldots, |J|\}$ and δ'.

$\mathsf{ClntNode.VerRetrieve}(\delta, Q, \mathsf{F}_Q, \pi_Q) \to b$ Output $b \leftarrow \mathsf{Ver}'(\mathsf{pp}, \delta, Q, \mathsf{F}_Q, \pi_Q)$

$\mathsf{ClntNode.ApplyUpdate}(\delta, \mathsf{op}, \Delta, \Upsilon_\Delta) \to (b, \delta')$ This algorithm is almost identical to the first part of the Storage Node algorithm $\mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta, \Upsilon_\Delta)$. The difference is that it executes only the parts that are related to the output of b and δ.

$\mathsf{AggregateCertificates}(\delta, (I, \mathsf{F}_I, \pi_I), (J, \mathsf{F}_J, \pi_J)) \to \pi_K$
Return $\pi_K \leftarrow \mathsf{VC.Agg}'(\mathsf{pp}, (I, \mathbf{F}_I, \pi_I), (J, \mathbf{F}_J, \pi_J))$.

Correctness. Here we state and prove the correctness of VDS_1.

Theorem 26. *The scheme VDS_1 presented above is a correct verifiable decentralized storage scheme.*

Proof. In the following we will always assume that $\mathsf{st} := (\mathsf{st}_1, \mathsf{st}_2)$ and $\delta := (\delta^\star, n) := ((\delta_1, \delta_2), n)$. Furthermore, whenever (a_I, b_I) appear, we assume that they are the outputs of $\mathsf{PartndPrimeProd}(I, \mathsf{F}_I)$, for each set of indices I. Finally for each set of indices I we assume $\pi_I := (\Gamma_I, \Delta_I)$.

First we note that in our construction it is sufficient for a local view $(\mathsf{pp}, \delta, n, \mathsf{st}, I, \mathsf{F}_I)$ of a storage node to be valid that
$\mathsf{ClntNode.VerRetrieve}(\delta, I, \mathsf{StrgNode.Retrieve}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, I)) = 1$ holds. More concretely this translates to $\mathsf{st}_1^{a_I} = \delta_1 \wedge \mathsf{st}_2^{b_I} = \delta_2$ and due to the correctness of disaggregation property $\mathsf{st}_1'^{a_Q} = \delta_1 \wedge \mathsf{st}_2'^{b_Q} = \delta_2$ holds where $\mathsf{st}' \leftarrow \mathsf{StrgNode.Retrieve}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, Q)$ for each $Q \subseteq I$. To put things clear, a local view of a storage node $(\mathsf{pp}, \delta, n, \mathsf{st}, I, \mathsf{F}_I)$ is valid if $\mathsf{st}_1^{a_I} = \delta_1 \wedge \mathsf{st}_2^{b_I} = \delta_2$.

Let $(\mathsf{pp}, \delta, n, \mathsf{st}, I, \mathsf{F}_I)$ be a valid local view of a storage node:
Update Correctness. Let (op, Δ) be an admissible update for (I, F_I, n) and $(\delta', n', \mathsf{st}', J, \mathsf{F}'_J, \Upsilon_\Delta)$ be the output of $\mathsf{StrgNode.PushUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta)$. We discern three cases depending on the type of update:

- $\mathsf{op} = \mathsf{mod}$:

1. According to our construction $\delta^{\star\prime} = (\Gamma_K^{a'_K}, \Delta_K^{b'_K})$, where
   $(\Gamma_K, \Delta_K) = (\Gamma_I^{a_{I\setminus K}}, \Delta_I^{b_{I\setminus K}}) = (\mathsf{st}_1^{\frac{a_I}{a_K}}, \mathsf{st}_2^{\frac{b_I}{b_K}})$ (due to $\mathsf{VC.Disagg}'$). So $\delta' = (\mathsf{st}_1^{\frac{a_I}{a_K}a'_K}, \mathsf{st}_2^{\frac{b_I}{b_K}b'_K})$.
   Furthermore $\mathsf{st}' = \mathsf{st}$ and $J = I$, so

$$(\mathsf{st}_1'^{a'_J}, \mathsf{st}_1'^{b'_J}) = (\mathsf{st}_1^{\frac{a_I}{a_K}a'_K}, \mathsf{st}_2^{\frac{b_I}{b_K}b'_K}) = (\delta'_1, \delta'_2)$$

2. Let $(\delta, n, \mathsf{st}_s, I_s, \mathsf{F}_{I_s})$ be valid and $(b_s, \delta'_s, n'_s, \mathsf{st}'_s, J_s, \mathsf{F}'_{J_s})$ be the output of $\mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta, \Upsilon_\Delta)$. $b_s = 1$, $\delta'_s = \delta'$ and $n'_s = n'$ come from inspection.
   If $I \cap K = \emptyset$ then
   $(\mathsf{st}'_{s,1}, \mathsf{st}'_{s,2}) \leftarrow \left(\mathbf{ShamirTrick}(\mathsf{st}_{s,1}, \Gamma_K, \mathsf{F}_I, \mathsf{F}_K)^{a'_K}, \mathbf{ShamirTrick}(\mathsf{st}_{s,2}, \Delta_K, \mathsf{F}_I, \mathsf{F}_K)^{b'_K}\right) =$
   $= (\mathsf{st}_{s,1}^{\frac{a'_K}{a_K}}, \mathsf{st}_{s,2}^{\frac{b'_K}{b_K}})$ and $(a'_I, b'_I) = (a_I, b_I)$ remains the same. So

$$(\mathsf{st}_{s,1}'^{a'_I}, \mathsf{st}_{s,2}'^{b'_I}) = (\mathsf{st}_{s,1}^{\frac{a'_K}{a_K}a_I}, \mathsf{st}_{s,2}^{\frac{b'_K}{b_K}b_I}) = (\delta_{s,1}, \delta_{s,2})$$

If $I \cap K = K$ then st_s doesn't change and $(a'_I, b'_I) = (\frac{a_I}{a_K} a'_K, \frac{b_I}{b_K} b'_K)$, hence

$$(\mathsf{st}_{s,1}^{\prime a'_I}, \mathsf{st}_{s,2}^{\prime b'_I}) = (\delta'_{s,1}, \delta'_{s,2})$$

The validity of $(\mathsf{pp}, \delta'_s, n'_s, \mathsf{st}'_s, J_s, \mathsf{F}'_{J_s})$ in the case of $I \cap K = L \notin \{\emptyset, K\}$ is covered by the above two, since it essentially is a sequence of the two above cases.

3. Let (b_c, δ_c) be the output of $\mathsf{ClntNode.ApplyUpdate}(\delta, \mathsf{op}, \Delta, \Upsilon_\Delta)$. It follows directly from the definition of $\mathsf{ClntNode.ApplyUpdate}$ (and its similarity with $\mathsf{StrgNode.ApplyUpdate}$) that $b_c = b_s = 1$ and $\delta'_c = \delta'_s = \delta'$.

- $\mathsf{op} = \mathsf{add}$:

1. According to our construction $\delta^{\star\prime} = (\delta_1^{a'_K}, \delta_2^{b'_K})$ and $\mathsf{st}' = \mathsf{st}$. Also, $J = I \cup K$ and $(a'_J, b'_J) = (a_I a'_K, b_I b'_K)$ and so

$$(\mathsf{st}_1^{\prime a'_J}, \mathsf{st}_1^{\prime b'_J}) = (\mathsf{st}_1^{a_I a'_K}, \mathsf{st}_2^{b_I b'_K}) = (\delta_1^{a'_K}, \delta_2^{b'_K}) = (\delta'_1, \delta'_2)$$

2. Let $(\delta, n, \mathsf{st}_s, I_s, \mathsf{F}_{I_s})$ be valid and $(b_s, \delta'_s, n'_s, \mathsf{st}'_s, J_s, \mathsf{F}'_{J_s})$ be the output of $\mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta, \Upsilon_\Delta)$. $b_s = 1$, $\delta'_s = \delta'$ and $n'_s = n'$ come from inspection. Also $J = I$ so $(a'_J, b'_J) = (a_I, b_I)$. $\mathsf{st}' = (\mathsf{st}_1^{a'_K}, \mathsf{st}_2^{b'_K})$ and $\delta^{\star\prime} = (\delta_1^{a'_K}, \delta_2^{b'_K})$ so

$$(\mathsf{st}_1^{\prime a'_J}, \mathsf{st}_1^{\prime b'_J}) = (\mathsf{st}_1^{a'_K a_I}, \mathsf{st}_2^{b'_K b_I}) = (\delta'_1, \delta'_1)$$

3. Let (b_c, δ_c) be the output of $\mathsf{ClntNode.ApplyUpdate}(\delta, \mathsf{op}, \Delta, \Upsilon_\Delta)$. Again correctness comes directly from the definition of $\mathsf{ClntNode.ApplyUpdate}$.

- $\mathsf{op} = \mathsf{del}$:

1. According to our construction $(\delta'_1, \delta'_2) = (\Gamma_K, \Delta_K) = (\delta_1^{\frac{1}{a_K}}, \delta_2^{\frac{1}{b_K}})$, $\mathsf{st}' = \mathsf{st}$ and $J = I \setminus K$. Furthermore, $(a'_J, b'_J) = (\frac{a_I}{a_K}, \frac{b_I}{b_K})$

$$(\mathsf{st}_1^{\prime a'_J}, \mathsf{st}_1^{\prime b'_J}) = (\mathsf{st}_1^{\frac{a_I}{a_K}}, \mathsf{st}_2^{\frac{b_I}{b_K}}) = (\delta_1^{\frac{1}{a_K}}, \delta_2^{\frac{1}{b_K}}) = (\delta'_1, \delta'_2)$$

2. Let $(\delta, n, \mathsf{st}_s, I_s, \mathsf{F}_{I_s})$ be valid and $(b_s, \delta'_s, n'_s, \mathsf{st}'_s, J_s, \mathsf{F}'_{J_s})$ be the output of $\mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta, \Upsilon_\Delta)$. $b_s = 1$, $\delta'_s = \delta'$ and $n'_s = n'$ come from inspection. Also let $L = I \cap K$ then $J = I \setminus L$ and if $\bar{L} = K \setminus L$ then
$(\mathsf{st}'_1, \mathsf{st}'_2) \leftarrow (\mathbf{ShamirTrick}(\mathsf{st}_1, \Gamma_{\bar{L}}, \mathsf{F}_I, \mathsf{F}_{\bar{L}})\,,\, \mathbf{ShamirTrick}(\mathsf{st}_2, \Delta_{\bar{L}}, \mathsf{F}_I, \mathsf{F}_{\bar{L}})) = (\mathsf{st}_1^{\frac{1}{a_{\bar{L}}}}, \mathsf{st}_2^{\frac{1}{b_{\bar{L}}}})$

$$(\mathsf{st}_1^{\prime a'_J}, \mathsf{st}_1^{\prime b'_J}) = (\mathsf{st}_1^{\frac{a_J}{a_{\bar{L}}}}, \mathsf{st}_2^{\frac{b_J}{b_{\bar{L}}}}) = (\mathsf{st}_1^{\frac{a_I/a_L}{a_K/a_L}}, \mathsf{st}_1^{\frac{b_I/b_L}{b_K/b_L}}) = (\delta_1^{\frac{1}{a_K}}, \delta_2^{\frac{1}{b_K}}) = (\delta'_1, \delta'_2)$$

3. Let (b_c, δ_c) be the output of $\mathsf{ClntNode.ApplyUpdate}(\delta, \mathsf{op}, \Delta, \Upsilon_\Delta)$. $b_c = b_s = 1$ and $\delta'_c = \delta'_s = \delta'$ from inspection.

Add Storage Correctness. It comes directly from aggregation correctness of $\mathsf{VC.Agg}'$ (see section 6.5.1.2).

Remove Storage Correctness. It comes directly from disaggregation correctness of $\mathsf{VC.Disagg}'$ (see section 6.5.1.2).

Create Correctness. Let $J \subseteq I$ and $(\delta', n', \mathsf{st}', J, \mathsf{F}_J, \Upsilon_J)$ be the output of $\mathsf{StrgNode.CreateFrom}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, J)$ and (b, δ'') the output of $\mathsf{ClntNode.GetCreate}(\delta, J, \Upsilon_J)$, then $n' = |J|$ comes from inspection of $\mathsf{StrgNode.CreateFrom}$, $\delta'' = \delta'$ comes from inspection of $\mathsf{ClntNode.GetCreate}$ algorithm and validity of $(\mathsf{pp}, \delta', n', \mathsf{st}', J, \mathsf{F}_J)$ comes from correctness of Com' and $\mathsf{VC.Agg}$. Finally, $b = 1$ comes from correctness of $\mathsf{PoKSubV}'$ protocol.

Aggregate Correctness. It comes directly from aggregation correctness of $\mathsf{VC.Agg}'$ (see section 6.5.1.2).

□

Security. Below we state and prove the security of our VDS_1 scheme.

Theorem 27 (Security)**.** *Let $\mathbb{G} \leftarrow \mathsf{Ggen}(1^\lambda)$ be a hidden order group where the strong RSA assumption holds, then the scheme VDS_1 presented above is a secure Verifiable Decentralized Storage scheme in the generic group model.*

Proof. First we observe that in our scheme, for every valid history $\mathcal{H}$, with $\mathsf{Bootstrap}(1^\lambda) \rightarrow (\mathsf{pp}, \delta_0, \mathsf{st}_0) := ((\mathbb{G}, g, g_0, g_1, \mathsf{PrimeGen}), ((g_0, g_1), 0), (g_0, g_1))$, the digest that arises is the same as a commitment of the file with Com'. Concretely, let $(b, \delta, \mathsf{F}) \leftarrow \mathsf{EvalHistory}(\mathsf{pp}, \delta_0, \mathsf{st}_0, \mathcal{H})$ then if $b = 1$ it holds that $\delta = \mathsf{Com}'(\mathsf{pp}, \mathsf{F})$ or $\delta^\star = (\delta_1, \delta_2) = (g_0^a, g_1^b)$, where $(a, b) \leftarrow \mathsf{PartndPrimeProd}([|\mathsf{F}|], \mathsf{F})$. Particularly this is central to our construction and one can validate that it holds by inspecting all the algorithms that alter the digest.

To prove the theorem we use a hybrid argument. We start by defining the game G_0 as the actual VDS security game of Definition 35, and our goal is to prove that for any PPT $\mathcal{A}$, $\Pr[G_0 = 1] \in \mathsf{negl}(\lambda)$.

Game G_0:

$G_0 = \mathsf{VDS\text{-}Security}^{\mathcal{A}}_{\mathsf{VDS}}(\lambda)$

$(\mathsf{pp}, \delta_0, \mathsf{st}_0) \leftarrow \mathsf{Bootstrap}(1^\lambda)$
$(\mathcal{H}, Q, \mathsf{F}^*_Q, \pi^*) \leftarrow \mathcal{A}(\mathsf{pp}, \delta_0, \mathsf{st}_0)$
$(b, \delta, \mathsf{F}) \leftarrow \mathsf{EvalHistory}(\mathsf{pp}, \delta_0, \mathsf{st}_0, \mathcal{H})$
$b \leftarrow b \wedge \mathsf{F}^*_Q \neq \mathsf{F}_Q \wedge$
  $\mathsf{ClntNode.VerRetrieve}(\mathsf{pp}, \delta, Q, \mathsf{F}^*_Q, \pi^*)$
return b

$\mathsf{EvalHistory}(\mathsf{pp}, \delta_0, \mathsf{st}_0, \mathcal{H})$

$\mathsf{F}_0 \leftarrow \emptyset; b \leftarrow 1$
for $i \in [\ell]$
  $\mathsf{F}_i \leftarrow \mathsf{FileChange}(\mathsf{F}_{i-1}, \mathsf{op}^i, \Delta^i)$
  if $\mathsf{op}^i \in \{\mathsf{mod}, \mathsf{add}, \mathsf{del}\}$ **then**
    $(b_i, \delta_i) \leftarrow \mathsf{ClntNode.ApplyUpdate}(\delta_{i-1}, \mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)$
  elseif $\mathsf{op}^i = \mathsf{cfrom}$ **then**
    $(b_i, \delta_i) \leftarrow \mathsf{ClntNode.GetCreate}(\delta_{i-1}, \Delta^i, \Upsilon^i_\Delta)$
  endif
  $b \leftarrow b \wedge b_i$
endfor
return $(b, \delta_\ell, \mathsf{F}_\ell)$

Recall that $\mathcal{H} = (\mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)_{i \in [\ell]}$ where:

- for $\mathsf{op}^i = \mathsf{mod}$: $\Delta^i := (K^i, \mathsf{F}^i_{K^i})$, $\Upsilon^i_\Delta := (\mathsf{F}^{i-1}_{K^i}, \pi^{i-1}_{K^i})$ and $\mathsf{ClntNode.ApplyUpdate}(\delta^{i-1}, \mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)$ outputs $b^i = 1$ if $\mathsf{Ver}'(\mathsf{pp}, \delta^{i-1}, K^i, \mathsf{F}^{i-1}_{K^i}, \pi^{i-1}_{K^i}) = 1$ or $(\Gamma^{a_{K^i}}_{K^i} = \delta^{i-1}_1) \wedge (\Delta^{b_{K^i}}_{K^i} = \delta^{i-1}_2)$.
- for $\mathsf{op}^i = \mathsf{add}$: $\Delta^i := (K, \mathsf{F}^i_{K^i})$, $\Upsilon^i_\Delta := \varnothing$ and $\mathsf{ClntNode.ApplyUpdate}(\delta^{i-1}, \mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)$ outputs $b^i = 1$ if $K^i = \{n^{i-1} + 1, \ldots, n^{i-1} + |K^i|\}$.
- $\mathsf{op}^i = \mathsf{del}$: $\Delta^i := K^i$, $\Upsilon^i_\Delta := (\mathsf{F}^{i-1}_{K^i}, \pi^{i-1}_{K^i})$ and $\mathsf{ClntNode.ApplyUpdate}(\delta^{i-1}, \mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)$ outputs $b^i = 1$ if $(K^i = \{n^{i-1} - |K^i| + 1, \ldots, n^{i-1}\}) \wedge \mathsf{Ver}'(\mathsf{pp}, \delta^{i-1}, K^{i-1}, \mathsf{F}^{i-1}_{K^i}, \pi^{i-1}_{K^i}))$ or $(K^i = \{n^{i-1} - |K^i| + 1, \ldots, n^{i-1}\} \wedge (\Gamma^{a_{K^i}}_{K^i} = \delta^{i-1}_1) \wedge (\Delta^{b_{K^i}}_{K^i} = \delta^{i-1}_2)$.
- $\mathsf{op}^i = \mathsf{cfrom}$: $\Delta^i := K^i$, $\Upsilon^i_\Delta := (\delta^i, \pi^i_{\mathsf{PoKSubV}'})$ and $\mathsf{ClntNode.GetCreate}(\delta^{i-1}, \Delta^i, \Upsilon^i_\Delta)$ outputs $b^i = 1$ if $\mathsf{PoKSubV}'.\mathsf{V}(\mathsf{pp}, (\delta^{i-1}, \delta^i, |K^i|, K^i), \pi^i_{K^i}) = 1$.

Game G_i: define G_i be the same as G_{i-1} except for the update i:

- if $\mathsf{op}^i = \mathsf{mod}$: $\Delta^i := (K^i, \mathsf{F}^i_{K^i})$, $\Upsilon^i_\Delta := (\mathsf{F}^{i-1}_{K^i}, \pi^{i-1}_{K^i})$ but in the i-th step of $\mathsf{EvalHistory}$ b^i is instead output of:
$$b^i \leftarrow (a_{K^i}|a) \wedge (b_{K^i}|b)$$
where $(a, b) \leftarrow \mathsf{PartndPrimeProd}([|\mathsf{F}^{i-1}|], \mathsf{F}^{i-1})$
In case $b^i = 0$ aborts ($\mathsf{abort_i}$). Otherwise δ^i is computed normally from $\mathsf{ClntNode.ApplyUpdate}(\delta^{i-1}, \mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)$.
- for $\mathsf{op}^i = \mathsf{add}$: $\Delta^i := (K, \mathsf{F}^i_{K^i})$, $\Upsilon^i_\Delta := \varnothing$ and everything is the same as in G_{i-1}. I.e. (b^i, δ^i) is the output of $\mathsf{ClntNode.ApplyUpdate}(\delta^{i-1}, \mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)$.
- $\mathsf{op}^i = \mathsf{del}$: $\Delta^i := K^i$, $\Upsilon^i_\Delta := (\mathsf{F}^{i-1}_{K^i}, \pi^{i-1}_{K^i})$. Similarly to the mod case b^i is the output of:
$$b^i \leftarrow (a_{K^i}|a) \wedge (b_{K^i}|b) \wedge (K^i = \{n^{i-1} - |K^i| + 1, \ldots, n^{i-1}\})$$
where $(a, b) \leftarrow \mathsf{PartndPrimeProd}([|\mathsf{F}^{i-1}|], \mathsf{F}^{i-1})$
In case $b^i = 0$ aborts ($\mathsf{abort_i}$). Otherwise δ^i is computed normally from $\mathsf{ClntNode.ApplyUpdate}(\delta^{i-1}, \mathsf{op}^i, \Delta^i, \Upsilon^i_\Delta)$.
- $\mathsf{op}^i = \mathsf{cfrom}$: $\Delta^i := K^i$, $\Upsilon^i_\Delta := (\delta^i, \pi^i_{\mathsf{PoKSubV}'})$ but in the i-th step of $\mathsf{EvalHistory}$ b^i is instead:
$$b^i \leftarrow (\mathsf{F}^{i-1}_{K^i} \subseteq \mathsf{F}^{i-1}) \wedge \delta^i = \mathsf{Com}'(\mathsf{pp}, \mathsf{F}^{i-1}_{K^i}) \wedge J = \{1, \ldots, |J|\}$$
In case $b^i = 0$ aborts ($\mathsf{abort_i}$).

Lemma 15. *Let* $\mathsf{op}^i = \mathsf{mod}$ *then if the strong RSA assumption holds for* Ggen, $\Pr[G_{i-1} = 1] \leq \Pr[G_i = 1] + \mathsf{negl}(\lambda)$.

Proof. It is straightforward that the only difference between G_{i-1} and G_i is in the computation of b^i inside the $\mathsf{EvalHistory}$. That is in G_{i-1} : $b^i = (\Gamma^{a_{K^i}}_{K^i} = \delta^{i-1}_1) \wedge (\Delta^{b_{K^i}}_{K^i} = \delta^{i-1}_2)$ and in G_i : $b^i = (a_{K^i}|a) \wedge (b_{K^i}|b)$. Since $\mathsf{abort_1}, \mathsf{abort_2}, \ldots, \mathsf{abort_{i-2}}$ have not happen, from correctness of the VDS scheme it comes that $(\delta^{i-1}_1, \delta^{i-1}_2) = (g^a_0, g^b_1)$, where $(a, b) \leftarrow \mathsf{PartndPrimeProd}([|\mathsf{F}^{i-1}|], \mathsf{F}^{i-1})$.

$|\Pr[G_{i-1}=1]-\Pr[G_i=1]| = Pr[\mathsf{abort_i}] = \mathsf{Pr}[\mathsf{b^i}=0] = \mathsf{Pr}[(\mathsf{a_{K^i}}|\mathsf{a}) \wedge (\mathsf{b_{K^i}}|\mathsf{b})]$. But since $\mathsf{abort_{i-1}}$ didn't happen $(\Gamma_{K^i}^{a_{K^i}} = g_0^a) \wedge (\Delta_{K^i}^{b_{K^i}} = g_1^b)$. Therefore it is straightforward to $\mathsf{abort_i}$ to the strong RSA assumption, i.e. $Pr[\mathsf{abort_i}] = \mathsf{negl}(\lambda)$. □

Lemma 16. *Let* $\mathsf{op}^i = \mathsf{del}$ *then if the strong RSA assumption holds for* Ggen, $\Pr[G_{i-1}=1] \leq \Pr[G_i=1] + \mathsf{negl}(\lambda)$.

Proof. The same as the above case of $\mathsf{op}^i = \mathsf{mod}$ holds. □

Lemma 17. *Let* $\mathsf{op}^i = \mathsf{add}$ *then* $\Pr[G_{i-1}=1] = \Pr[G_i=1]$.

Proof. G_{i-1} and G_i are identical. □

Lemma 18. *Let* $\mathsf{op}^i = \mathsf{cfrom}$ *then for any PPT* $\mathcal{A}$ *in* G_i *there exists an algorithm* $\mathcal{E}$ *such that* $\Pr[G_{i-1}=1] \leq \Pr[G_i=1] + \mathsf{negl}(\lambda)$ *of the strong RSA assumption holds.*

Proof. Let $\mathcal{E}$ be the extractor of $\mathsf{PoKSubV'}$ protocol that corresponds to $\mathcal{A}$. Since $\mathsf{PoKSubV'}$ is knowledge sound, $\mathcal{E}$ outputs $(\mathbf{F}_{K^i}^{i-1}, \pi_{K^i}, \pi'_{K^i})$ such that $\mathsf{Ver'}(\mathsf{pp}, \delta^{i-1}, K^i, \mathbf{F}_{K^i}^{i-1}, \pi_{K^i}) = 1 \wedge \mathsf{Ver'}(\mathsf{pp}, \delta^i, K^i, \mathbf{F}_{K^i}^{i-1}, \pi'_{K^i}) = 1 \wedge |\mathbf{F}_{K^i}^{i-1}| = n'$, where $\delta^i = (\delta^{\star i}, n^i)$. Since $\mathsf{abort_1}$, $\mathsf{abort_2}, \ldots, \mathsf{abort_{i-2}}$ have not happen, from correctness of the VDS scheme it comes that $\delta^{i-1} = \mathsf{Com'}(\mathsf{pp}, \mathsf{F}^{i-1})$. From the first verification equation above we get that under strong RSA assumption $\mathsf{F}_{K_i}^{i-1} \subseteq \mathsf{F}^{i-1}$. From the second verification equation above we get that $\mathsf{F}_{K_i}^{i-1}$ is an opening of δ^i. From the third equation above we get that δ^i is a digest for a file of size $|\mathsf{F}_{K_i}^{i-1}|$. From the last two points we get that $\delta^i = \mathsf{Com'}(\mathsf{pp}, \mathsf{F}_{K^i}^{i-1})$.

So $\Pr[G_{i-1}=1] \leq \Pr[G_i=1] + \mathsf{negl}(\lambda)$. □

We conclude that in any case $\Pr[G_{i-1}=1] \leq \Pr[G_i=1] + \mathsf{negl}(\lambda)$. Since $|\mathcal{H}| = \ell = \mathsf{poly}(\lambda)$ with a hybrid argument we get that $\Pr[G_0=1] \leq \Pr[G_\ell=1] + \mathsf{negl}(\lambda)$. But clearly $G_\ell = 0$ always (since no abort has happened), and thus $\Pr[\mathsf{VDS\text{-}Security}_{\mathsf{VDS}}^{\mathcal{A}}(\lambda) = 1] = P[G_0=1] = \mathsf{negl}(\lambda)$. □

6.8.2 Our Second VDS Construction

To construct our second VDS scheme, denoted VDS_2, we build on our second SVC scheme from section 6.5.2. The main difficulty that we face in turning our SVC into a VDS is the specializtion phase of the CRS, i.e. the trusted generation of $U = g^{\prod_{i\in[n]} e_i}$. Although VDS schemes can support a trusted setup phase, it can only be done once by the Bootstrap algorithm. However, U depends on the current size of the file (though not on its content), meaning that normally at each addition (or deletion) to the file it should be updated[47]. To solve this problem, we attach U to the VDS's digest (together with n for technical reasons), $\delta = ((U, C), n)$.

Then, U can be built progressively while the file is extended or reduced. Namely, when adding new positions from the set K to the file, all e_i's in K are added to the accumulator, i.e. $U' \leftarrow U^{\prod_{i\in K} e_i}$. The definition of VDS security (def. 35) ensures that the digest is evaluated honestly which ensures that U has the correct form $U = g^{\prod_{i\in n} e_i}$.

Finally, we make use of the dynamic properties of the [66, 145] scheme (in which our SVC builds) and the RSA Accumulator, to construct the VDS scheme. The latter is important if one

[47] Another solution would be to recompute it at the verification time, but it would require linear work, which contradicts VDS requirements.

notice that $U = g^{\prod_{i\in[n]} e_i}, S_I = g^{\prod_{i\in[n]\setminus I} e_i}$ resemble an RSA Accumulator value and witness respectively.

Our scheme VDS_2**.** In the following $\delta := ((U, C), n)$, $\mathsf{st} := \pi_I$, where $\pi_I := (S_I, \Lambda_I)$. Also, each e_i is computed as $e_i \leftarrow \mathsf{PrimeGen}(i)$; so $\mathsf{PrimeGen}(i)$ is omitted for simplicity in the description. $\mathsf{VC.Agg}, \mathsf{VC.Disagg}$ are the aggregation and disaaggregation algorithms defined in section 6.5.2. We highlight that possession of S_I allows anyone to compute $S_J \leftarrow S_I^{\prod_{j\in I\setminus J} e_j}$ for each $J \subseteq I$, thus for simplicity we omit explicitly refer to the procedure of computing any such S_J.

$\mathsf{Bootstrap}(1^\lambda, \ell) \to (\mathsf{pp}, \delta_0, n_0, \mathsf{st}_0)$ generates a hidden order group $\mathbb{G} \leftarrow \mathsf{Ggen}(1^\lambda)$ and samples a generator $g \leftarrow_\$ \mathbb{G}$. It also determines a deterministic collision resistant function $\mathsf{PrimeGen}$ that maps integers to primes of $\ell + 1$ bits. Set $n_0 \leftarrow 0$, $\delta_0 \leftarrow ((1, g), n_0)$ and $\mathsf{st}_0 \leftarrow g$.

$\mathsf{StrgNode.AddStorage}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, Q, \mathsf{F}_Q, \pi_Q) \to (\mathsf{st}', J, \mathsf{F}_J)$ aggregates the parameters and the opening proofs

$$S_{I\cup Q} \leftarrow \mathbf{ShamirTrick}(S_I, S_Q, \prod_{i\in I} e_i, \prod_{i\in Q} e_i) \text{ and } \Lambda_{I\cup Q} \leftarrow \mathsf{VC.Agg}((S_I, S_J), (I, \mathsf{F}_I, \Lambda_I), (J, \mathsf{F}_J, \Lambda_J))$$

$\mathsf{StrgNode.RmvStorage}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, K) \to (\mathsf{st}', J, \mathsf{F}_J)$ disaggregates

$$S_J \leftarrow S_I^{\prod_{i\in I\cap K} e_i} \text{ and } \Lambda_J \leftarrow \mathsf{VC.Disagg}(S_J, I, \mathsf{F}_I, \Lambda_I, J)$$

$\mathsf{StrgNode.PushUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta) \to (\delta', n', \mathsf{st}', J, \mathsf{F}'_J, \Upsilon_\Delta)$ the algorithm works according to the type of update operation op:

- $\mathsf{op} = \mathsf{mod}$: $\Delta := (K, \mathsf{F}'_K)$.

$$C' \leftarrow C \cdot \prod_{i\in K} S_i^{\mathsf{F}'_i - \mathsf{F}_i}, \qquad U' \leftarrow U, \qquad \Lambda'_I \leftarrow \Lambda_I, \qquad S'_I \leftarrow S_I \qquad \Upsilon_\Delta \leftarrow (\mathsf{F}_K, S_K)$$

- $\mathsf{op} = \mathsf{add}$: $\Delta := (K, \mathsf{F}'_K)$.

$$C' \leftarrow C \cdot \prod_{j\in K} S_j^{\mathsf{F}_j}, \qquad U' \leftarrow U^{\prod_{i\in K} e_i}, \qquad \Lambda'_I \leftarrow \Lambda_I, \qquad S'_I \leftarrow S_I, \qquad \Upsilon_\Delta \leftarrow S_K$$

- $\mathsf{op} = \mathsf{del}$: $\Delta := K$.

$$C' \leftarrow \frac{C}{\prod_{j\in K} S_j^{\mathsf{F}_j}}, \quad U' \leftarrow S_I^{\prod_{i\in I\setminus K} e_i} = S_K, \quad \Lambda'_I \leftarrow \Lambda_I^{\prod_{j\in K} e_j}, \quad S'_I \leftarrow S_I, \quad \Upsilon_\Delta \leftarrow (\mathsf{F}_K, S_K)$$

$\mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta, \Upsilon_\Delta) \to (b, \delta', n', \mathsf{st}', J, \mathsf{F}'_J)$ Again, it works according to the type of update operation op:

- $\mathsf{op} = \mathsf{mod}$: $\Delta := (K, \mathsf{F}'_K)$ and $\Upsilon_\Delta := S_K$. Compute $b \leftarrow (S_K^{\prod_{j\in K} e_j} = U)$ and if $b = 1$:

$$C' \leftarrow C \cdot \prod_{i\in K} S_i^{\mathsf{F}'_i - \mathsf{F}_i}, \quad U' \leftarrow U, \quad \Lambda'_I \leftarrow \Lambda_I \cdot \prod_{j\in K\setminus I} \left(S_j^{1/\prod_{i\in I} e_i}\right)^{\mathsf{F}'_j - \mathsf{F}_j}, \; S'_I \leftarrow S_I$$

- $\mathsf{op} = \mathsf{add}$: $\Delta := (K, \mathsf{F}'_K)$ and $\Upsilon_\Delta := S_K$. Compute $b \leftarrow (S_K^{\prod_{j\in K} e_j} = U)$ and if $b = 1$:

$$C' \leftarrow C \cdot \prod_{j\in K} S_j^{\mathsf{F}_j}, \quad U' \leftarrow U^{\prod_{i\in K} e_i}, \quad \Lambda'_I \leftarrow \Lambda_I \cdot \prod_{j\in K} \left(S_j^{1/\prod_{i\in I} e_i}\right)^{\mathsf{F}_j}, \qquad S'_I \leftarrow S_I^{\prod_{i\in K} e_i}$$

  where $S_j^{1/\prod_{i\in I} e_i} = \mathbf{ShamirTrick}(S_I, S_j, \prod_{i\in I} e_i, e_j)$ for each $j \in K$.

- $\mathsf{op} = \mathsf{del}$: $\Delta := K$ and $\Upsilon_\Delta := (\mathsf{F}_K, S_K)$. Compute $b \leftarrow (S_K^{\prod_{j\in K} e_j} = U)$ and if $b = 1$:

$$C' \leftarrow \frac{C}{\prod_{j\in K} S_j^{\mathsf{F}_j}}, \qquad U' \leftarrow S_K,$$

$$\Lambda'_I \leftarrow \frac{\Lambda_I^{\prod_{i\in K\cap I} e_i}}{\prod_{j\in K\setminus I} \left(S_j^{1/\prod_{i\in K\cap I} e_i}\right)^{\mathsf{F}_j}}, \qquad S'_I \leftarrow \mathbf{ShamirTrick}(S_I, S_{K\setminus I}, \prod_{i\in I} e_i, \prod_{i\in K\setminus I} e_i)$$

$\mathsf{StrgNode.Retrieve}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, Q) \to (\mathsf{F}_Q, \pi_Q)$ disaggregates

$$S_Q \leftarrow S_I^{\prod_{i\in I\setminus Q} e_i} \text{ and } \Lambda_Q \leftarrow \mathsf{VC.Disagg}(S_Q, I, \mathsf{F}_I, \Lambda_I, Q)$$

The algorithms for client nodes are:

$\mathsf{ClntNode.VerRetrieve}(\delta, Q, \mathsf{F}_Q, \pi_Q) \to b$ output

$$b \leftarrow \mathsf{Ver}(\mathsf{pp}, C, Q, \mathsf{F}_Q, \Lambda_Q) \wedge S_Q^{\prod_{i\in Q} e_i} = U$$

$\mathsf{ClntNode.ApplyUpdate}(\delta, \mathsf{op}, \Delta, \Upsilon_\Delta) \to (b, \delta')$ This algorithm is almost identical to the first part of the Storage Node algorithm $\mathsf{StrgNode.ApplyUpdate}(\delta, n, \mathsf{st}, I, \mathsf{F}_I, \mathsf{op}, \Delta, \Upsilon_\Delta)$. The difference is that it executes only the parts that are related to the output of b and δ.

$\mathsf{AggregateCertificates}(\delta, (I, \mathsf{F}_I, \pi_I), (J, \mathsf{F}_J, \pi_J)) \to \pi_K$ return

$$S_{I\cup J} \leftarrow \mathbf{ShamirTrick}(S_I, S_J, \prod_{i\in I} e_i, \prod_{i\in J} e_i) \text{ and } \Lambda_K \leftarrow \mathsf{VC.Agg}\left((S_I, S_J), (I, \mathbf{F}_I, \Lambda_I), (J, \mathbf{F}_J, \Lambda_J)\right)$$

We note that we do not define an efficient $\mathsf{StrgNode.CreateFrom}$ operation for the VDS_2 construction. While general-purpose SNARKs would work to achieve this result, they would be extremely expensive. We leave it as an open problem to find an efficient arguments of knowledge of subvector opening for this scheme.

Theorem 28 (VDS_2)**.** *Let $\mathbb{G} \leftarrow \mathsf{Ggen}(1^\lambda)$ be a hidden order group where the strong Distinct-Prime-Product Root and the Low Order assumptions hold. Then the* VDS *scheme presented above is a correct and secure Verifiable Decentralized Storage scheme.*

The intuition of the above theorem is as follows: the VDS scheme can be seen as preserving and updating a vector commitment C and an RSA Accumulator U. So correctness of VDS comes from correctness of the updatable vector commitment SVC and correctness of updates of the RSA Accumulator (see [40]). Similarly, security comes from security of SVC and the RSA accumulator's security, which in turn rely on the strong distinct-prime-product root assumption

and the strong RSA assumption respectively. Note that strong Distinct-Prime-Product Root implies strong RSA (the opposite also holds in RSA groups).

Recall that U is an RSA accumulator of all e_i's and is used to verify S_I's. The RSA accumulator's security demands that the accumulated value U is honestly computed, which is ensured in the VDS setting since we assume a valid history. So given a valid history one knows that U is of correct form (i.e. $U = g^{\prod_{i\in[n]} e_i}$) and then can securely check that S_I is of correct form (by checking $S_I^{\prod_{i\in I} e_i} = U$), which is ensured from RSA Accumulator's security. After checking the validity of S_I it all boils down to position binding of the vector commitment. To conclude, the gap between position binding of the original VC and security of our VDS construction is to ensure that S_I is well formed, which in turn relies on the correct form of U.

6.8.3 Efficiency and Comparison

In Table 6.3 we provide a detailed efficiency analysis and comparison of the two VDS schemes, VDS_1 and VDS_2, proposed in the two earlier sections.

In terms of performances, the two schemes do similarly, though VDS_2 outperforms the first one by a logarithmic factor. Its efficiency advantage comes from the fact that operations are not bit-by-bit as in the first one. More in detail, in VDS_1 most of the operations require one exponentiation with an α-bit prime for each bit and each position of the subfile, roughly $O(\ell \cdot |I| \cdot \alpha)$ group operations. In VDS_2, the main overhead is related to handling the distributed parameters $\{S_i\}$. In fact, computing S_i for each $i \in I$, given S_I takes $O(I \log |I|)$ exponentiations with $(\ell+1)$-bit primes, roughly $O(\ell \cdot |I| \cdot \log |I|)$ group operations.

To compare the two methods, recall that α is at least $\log(\ell n)$ (since we need at least ℓn distinct primes), which means that VDS_1 has a (logarithmic) dependence on the size of the file. On the other hand, VDS_2's cost depends only on the size of the subfile that is processed. Hence, since $\alpha > \log(\ell n) > \log(n) \geq \log(|I|)$ the VDS_2 always outperforms VDS_1 (see Table 6.3).

Another notable difference regards the $\mathsf{StrgNode.PushUpdate}$ algorithm for $\mathsf{op} = \mathsf{mod}$. In VDS_2, the running time depends solely on the size of the update, whereas in VDS_1 it depends on the size of the entire subfile stored locally. This can be a huge difference for nodes that decide to store large portions, and it constitutes a major theoretical (and practical) improvement of VDS_2 over VDS_1.

In terms of security, VDS_1 is based on a weaker assumption[48], over groups of unknown order, than VDS_2 (although for the specific case of RSA groups the two assumptions are equivalent). Finally, in terms of functionality, VDS_1 is the only scheme that can support efficiently the $\mathsf{StrgNode.CreateFrom}$ functionality and the (compact) Proofs of Data Possession; this is thanks to its compatibility with the efficient succinct arguments of knowledge that we propose in section 6.6.

[48] This holds when considering the basic scheme without the $\mathsf{StrgNode.CreateFrom}$ functionality.

Metric		VDS_1	VDS_2
Bootstrap		$O(1)$	$O(1)$
$\lvert\mathsf{pp}\rvert$		$3\lvert\mathbb{G}\rvert$	$1\lvert\mathbb{G}\rvert$
Digest $\lvert\delta\rvert$		$2\lvert\mathbb{G}\rvert + \log\lvert\mathsf{F}\rvert$	$2\lvert\mathbb{G}\rvert + \log\lvert\mathsf{F}\rvert$
Storage Node storing (I, F_I)			
State $\lvert\mathsf{st}_I\rvert$		$2\lvert\mathbb{G}\rvert$	$2\lvert\mathbb{G}\rvert$
StrgNode.AddStorage (K)		$O(\ell\cdot\alpha\cdot(\lvert I\rvert+\lvert K\rvert))$	$O(\ell\cdot(\lvert I\rvert\log\lvert I\rvert+\lvert K\rvert\log\lvert K\rvert))$
StrgNode.RmvStorage (K)		$O(\ell\cdot\alpha\cdot\lvert K\rvert)$	$O(\ell\cdot\lvert K\rvert\log\lvert K\rvert)$
StrgNode.CreateFrom (J)		$O(\ell\cdot\alpha\cdot\lvert I\rvert)$	no [1]
$\lvert\Upsilon_J\rvert$		$9\lvert\mathbb{G}\rvert + 2\lvert\mathbb{Z}_{2^k}\rvert$	
StrgNode.PushUpdate (Δ)	mod	$O(\ell\cdot\alpha\cdot\lvert I\rvert)$	$O(\ell\cdot\lvert\Delta\rvert\log\lvert\Delta\rvert)$
	add	$O(\ell\cdot\alpha\cdot\lvert\Delta\rvert)$	$O(\ell\cdot\lvert\Delta\rvert\log\lvert\Delta\rvert)$
	del	$O(\ell\cdot\alpha\cdot(\lvert I\rvert-\lvert\Delta\rvert))$	$O(\ell\cdot(\lvert I\rvert-\lvert\Delta\rvert+\lvert\Delta\rvert\log\lvert\Delta\rvert))$
$\lvert\Upsilon_\Delta\rvert$	mod, del	$O(\lvert\Delta\rvert)+2\cdot\lvert\mathbb{G}\rvert$	$O(\lvert\Delta\rvert)+1\cdot\lvert\mathbb{G}\rvert$
	add	$\varnothing$	1
StrgNode.ApplyUpdate (Δ)	mod	$O(\ell\cdot\alpha\cdot(\lvert I\rvert+\lvert\Delta\rvert))$	$O(\ell\cdot(\lvert I\rvert+\lvert\Delta\rvert\log\lvert\Delta\rvert))$
	add	$O(\ell\cdot\alpha\cdot\lvert\Delta\rvert)$	$O(\ell\cdot(\lvert I\rvert+\lvert\Delta\rvert\log\lvert\Delta\rvert))$
	del	$O(\ell\cdot\alpha\cdot(\lvert I\rvert+\lvert\Delta\rvert))$	$O(\ell\cdot(\lvert I\rvert+\lvert\Delta\rvert\log\lvert\Delta\rvert))$
StrgNode.Retrieve (Q)		$O(\ell\cdot\alpha\cdot(\lvert I\rvert-\lvert Q\rvert))$	$O(\ell\cdot(\lvert I\rvert-\lvert Q\rvert)\log(\lvert I\rvert-\lvert Q\rvert))$
$\lvert\pi_Q\rvert$		$2\lvert\mathbb{G}\rvert$	$2\lvert\mathbb{G}\rvert$
Client Node			
ClntNode.GetCreate (J)		$O(\ell\cdot\alpha\cdot\lvert J\rvert)$	no [1]
ClntNode.VerRetrieve (Q)		$O(\ell\cdot\alpha\cdot\lvert Q\rvert)$	$O(\ell\cdot\lvert Q\rvert\log\lvert Q\rvert)$
ClntNode.ApplyUpdate (Δ) (mod, add, del)		$O(\ell\cdot\alpha\cdot\lvert\Delta\rvert)$	$O(\ell\cdot\lvert\Delta\rvert\log\lvert\Delta\rvert)$
AggregateCertificates (I, J)		$O(\ell\cdot\alpha\cdot(\lvert I\rvert+\lvert J\rvert))$	$O(\ell\cdot(\lvert I\rvert\log\lvert I\rvert+\lvert J\rvert\log\lvert J\rvert))$
PoR		yes	yes
PDP		yes	no [1]

Table 6.3: Comparison between our two VDS schemes. The running time is expressed in number of $\mathbb{G}$-group operations. Notation for the sets of positions: I are the ones held by the storage node, K the ones added or removed from local storage by the storage node, J the ones used to create the file in StrgNode.CreateFrom, Δ the updated ones, and Q the ones of a retrieval query. In VDS_1, α denotes the size of the primes (returned by PrimeGen); so $\alpha \geq \log(n\ell)$ where n is the size of the file and ℓ the bit-size of each position (i.e. $\mathsf{F} \in (\{0,1\}^\ell)^n$).

[1] Such a protocol exists but it is either inefficient for the prover (SNARKs) or it has a large overhead in communication complexity (Σ-protocols or PoKE-based ones).

6.9 PoProd protocol for Union of RSA Accumulators

Let $\mathbb{G}$ be a an hidden order group as generated by Ggen, and let $g_1, g_2, g_3 \in \mathbb{G}$ be three honestly sampled random generators. A more straightforward succinct argument of knowledge for the union of RSA Accumulators is for the following relation

$$R_{\mathsf{PoProd}} = \left\{((A,B,C),(a,b)) \in \mathbb{G}^3 \times \mathbb{Z}^2 \ : \ A = g_1^a \wedge B = g_2^b \wedge C = g_3^{a\cdot b} \quad \right\}$$

Our protocol PoProd is described below.

PoProd protocol

$\mathsf{Setup}(1^\lambda)$: run $\mathbb{G} \leftarrow_\$ \mathsf{Ggen}(1^\lambda)$, $g_1, g_2, g_3 \leftarrow_\$ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g_1, g_2, g_3)$.
Prover's input: $(\mathsf{crs}, (A, B, C), (a, b))$. Verifier's input: $(\mathsf{crs}, (A, B, C))$.

$\mathsf{V} \to \mathsf{P}$: $\ell \leftarrow_\$ \mathbb{P}(1, 2^\lambda)$

$\mathsf{P} \to \mathsf{V}$: $\pi := ((Q_A, Q_B, Q_C), r_a, r_b)$ computed as follows

- $(q_a, q_b, q_c) \leftarrow (\lfloor a/\ell \rfloor, \lfloor b/\ell \rfloor, \lfloor ab/\ell \rfloor)$
- $(r_a, r_b) \leftarrow (a \mod \ell, b \mod \ell)$
- $(Q_A, Q_B, Q_C) := (g_1^{q_a}, g_2^{q_b}, g_3^{q_c})$

$\mathsf{V}(\mathsf{crs}, (A, B, C), \ell, \pi)$:

- Compute $r_c \leftarrow r_a \cdot r_b \mod \ell$
- Output 1 iff $r_a, r_b \in [\ell] \ \wedge \ Q_A^\ell g_1^{r_a} = A \ \wedge \ Q_B^\ell g_2^{r_b} = B \ \wedge \ Q_C^\ell g_3^{r_c} = C$

To prove the security of our protocol we rely on the adaptive root assumption and, in a non-black-box way, on the knowledge extractability of the PoKE protocol from [40]. The latter is proven in the generic group model for hidden order groups (where also the adaptive root assumption holds).

Theorem 29. *The* PoProd *protocol is an argument of knowledge for* R_{PoProd} *in the generic group model.*

The proof is quite similar to the one of theorem 20 only instead of using the extractor if PoKRep protocol we use the extractors of two PoKE protocols (one for $g_1^a = A$ and one for $g_2^b = B$).

6.10 Comparison with the [40] SVC on Committing and Opening with Precomputation

We discuss how the preprocessing technique can also be applied to the SVC scheme of [40] (instantiated for binary vectors of length $n = N\ell$). In this case, however, we will not use the incremental disaggregation and aggregation but only one-hop aggregation.

Let us recall that in [40] a commitment to $\boldsymbol{v} \in \{0,1\}^n$ is $Acc = g^b$ with $b = \prod_{j \in [n], v_j = 1} p_j$. When asked for opening of some positions in the set I, the vector owner has to provide a batched membership proof for all $\{p_j : j = (i-1)\ell + l, i \in I, l \in [\ell], v_j = 1\}$ and a batched non-membership proof for all $\{p_j : j = (i-1)\ell + l, i \in I, l \in [\ell], v_j = 0\}$.

For the membership proofs, we can use ideas similar to the ones discussed earlier. In the commitment phase one can precompute $\{W_i = g^{b/b_i} : i \in [N]\}$ where $b_i = \prod_{l \in [\ell], v_{il} = 1} p_{(i-1)\ell + l}$, which can be done in time $O(N \log N \cdot \ell \log(\ell N))$ using the **RootFactor** algorithm from [188, 40]. This adds at most N elements of $\mathbb{G}$ to the advice information. Next, in the opening phase, in order to compute a membership witness for a set of positions I one can use the aggregation property to compute a witness W_I from all W_i with $i \in I$, which is doable in time $O(m \log m)$.

For the non-membership proof, there are instead two options:

1. Compute the batch-nonmembership witness from scratch
2. Precompute and store (unbatched) non-membership witnesses for all 0's of the vector and then aggregate the necessary ones to provide the opening asked.

We argue that an intermediate solution of precomputing a fraction of non-membership witnesses and computing the rest from scratch does not provide any benefit since even if a single non-membership witness needs to be computed, it requires the whole vector and computing the corresponding product of primes. So, in the end the intermediate solution will be more costly than both the above ones.

1. Compute non-membership witness from scratch. To compute a non-membership witness one needs the product b of all the primes in the accumulator (i.e., all primes that correspond to 1's in $\boldsymbol{v}$). There are in turn two possible ways to deal with this:

- Precompute and store b, which requires $O(\log(N\ell) \cdot N \cdot \ell)$ computation and $|b| = O(N \cdot \ell \cdot \log(N\ell))$ bits of storage.
- Compute b online from all p_i's, which requires $O(\log(N\ell) \cdot N \cdot \ell)$ computing power.

The computations needed to obtain a single non-membership witness is proportional to the size of b, which is $O(\ell \cdot N \cdot \log(\ell N))$ $\mathbb{G}$. Hence, virtually there is no big improvement in the opening time by precomputing b, since the group exponentiations are more costly (although concretely it saves the online computation of it). Furthermore, keeping $|b| = O(N \cdot \ell \cdot \log(N\ell))$ bits of storage may get impractical for big N.

2. Precompute non-membership witnesses and then aggregate. The idea is similar to the aggregation technique mentioned above for membership witnesses. However, a crucial difference is that, as stated in [40], for non-membership witnesses one has only *one-hop* aggregation. This means one must precompute and store non-membership witnesses for each block of the vector. However these non-membership witnesses have size proportional to the number of bits of each block (plus one group element).

This technique requires storage of $O(N)$ group elements plus $O(N \cdot \ell \log(N\ell))$ field elements on average. Precisely, the size of a non-membeship witness for each block is $|\mathbb{G}| + \log(N\ell) \times \#\{0\text{-bits in the block}\}$, hence the total size of non-membership witnesses is $N|\mathbb{G}| + N\ell \log(N\ell)$ in the worst case and $N|\mathbb{G}| + N\ell \log(N\ell)/2$ in an average case where half of the bits of the vector are 0. To conclude, with the VC of [40], one would need, on average, to precompute and store $2N|\mathbb{G}| + N\ell \log(N\ell)/2$ bits.

Comparison. To conclude, even if we consider the case $B = 1$, both our solutions require much less storage than in [40]: $2N|\mathbb{G}|$ vs. $2N|\mathbb{G}| + N\ell \log(N\ell)/2$ bits. In terms of computing time, the preprocessing has roughly the same complexity in all three solutions, although our second scheme is slightly less favorable due to the $\log^2 m$ factor in the opening. Comparing [40] and our first scheme, in [40] the computing time for an opening of m blocks requires at least 50% more time than in our first scheme due to the handling of non-membership witnesses (which leads to 25% more time in the average case).

7

INNER PRODUCT FUNCTIONAL COMMITMENTS FROM SET ACCUMULATORS

The results of this chapter appear in a paper under the title "Inner Product Functional Commitments with Constant-Size Public Parameters and Openings" published at the SCN 2022 conference [76].

7.1 Technical Contributions

We summarize the results of this chapter below.

- **FC for binary inner products with constant-size openings.** Our first result is a functional commitment that supports the evaluation of binary inner products over the integers. Namely one can commit to a vector $\boldsymbol{v} \in \{0,1\}^n$ and, for any $\boldsymbol{f} \in \{0,1\}^n$, open the commitment to $\langle \boldsymbol{v}, \boldsymbol{f} \rangle$ computed over $\mathbb{Z}$. The scheme works over groups of unknown order and, due to the use of succinct proofs of exponentiation from [40], relies on the random oracle and generic group models. The scheme's public parameters are four group elements, while openings consist of 21 elements of the hidden-order group, and 14λ bits.

  While all prior FCs for inner products use techniques that somehow rely on the homomorphic property of an underlying vector commitment, our construction departs from this blueprint and shows a new set of techniques for proving an inner product. In a nutshell, we start from the first vector commitment of Campanelli et al. [59], which uses an encoding of a vector based on two RSA accumulators, and then we show how to reduce the problem of proving an inner product with a public function to that of proving that a certain exponent lies in a range. To the best of our knowledge, this technique is novel. Also, a core part of this technique is a way to succinctly prove the cardinality of a set in an RSA accumulator, which we believe can be of independent interest.

- **FC for integer inner products.** Our second result is a collection of transformations that lift an FC for binary inner products, like the one above, to one that supports the computation of inner products over the integers and over finite rings. More in detail, we show two main transformations for the following functionality: one can commit to a vector $\boldsymbol{v} \in (\mathbb{Z}_{2^\ell})^n$ and, for any $\boldsymbol{f} \in (\mathbb{Z}_{2^m})^n$, open the commitment to $\langle \boldsymbol{v}, \boldsymbol{f} \rangle$ computed over $\mathbb{Z}$.

Through the first transformation, we obtain an FC whose openings have size of $O(\ell + m)$ group elements and additive $(\ell + m)\log(\ell n)$ bits, and whose algorithms running time is approximately $(\ell + m)$ times that of the FC for binary inner products.

Through the second transformation, we achieve a different tradeoff: the algorithms' running time grow by a factor $2^{\ell+m}$ but openings have a fixed size $O(1)$ group elements.

We also show analogues of both transformations for the case of inner products modulo any integer p, i.e., for $\langle\cdot,\cdot\rangle : \mathbb{Z}_p^n \times \mathbb{Z}_p^n \to \mathbb{Z}_p$, that yield FCs with the same complexity as the ones above, considering ℓ and m as the bitsize of p.

Among the two, the second transformation is of particular interest because, in the case of $\ell, m = O(\log \lambda)$ (resp. $p = \mathsf{poly}$) it yields functional commitment schemes with constant-size openings.

Finally, due to the known construction of polynomial commitments from functional commitments for inner products (see above), our FCs also imply polynomial commitments with transparent setup for polynomials in $\mathbb{Z}_p[X]$.

- **Comparison and concrete interpretation of our results.** As mentioned above, the objective of our work is to eliminate any dependence on the size of the parameters and proofs on the vector length n. Our constructions have sizes dependent only on the security parameter λ. When concretely instantiating the group of unknown order these sizes get $O(\lambda^2)$ for class groups [129, 35, 94] or $O(\lambda^3)$ for RSA groups.

  On the other hand, elliptic curve group elements typically have size $O(\lambda)$. Therefore, if we consider polynomial lengths $n = \mathsf{poly}$ then elliptic curve-based functional vector commitments as Bulletproofs [52] have proof size $O(\lambda \log n) = O(\lambda \log \lambda)$, which are concretely more efficient. For this, our results firstly serve as feasibility results for the complexity of the sizes of functional vector commitments. We note, however, that our solutions would still be asymptotically better if different unknown order group instantiations with optimal $O(\lambda)$ size were introduced, or in complexity leveraging scenarios where one considers super-polynomial vector sizes, $n > \mathsf{poly}$.

  This asymptotic drawback of constant-sized constructions is typical for many primitives based on groups of unknown order such as RSA accumulators [29, 18, 152, 40], vector commitments [66, 145, 40, 59] or SNARKs [145].

  Therefore, if we compare to the functional commitment built using the Bulletproofs inner product argument [52] (which to the best of our knowledge is currently the most efficient one that admits constant-sized and transparent parameters) the proof sizes of our schemes are concretely larger (for $n = \mathsf{poly}$). On the other hand, our FC has two main advantages. The first one is *flexibility*. Our FC "natively" supports inner products over $\mathbb{Z}_p$ for *any* integer p, whereas Bulletproofs only supports inner products over $\mathbb{Z}_q$ where q is the prime modulus of a group $\mathbb{G}$ where discrete logarithm holds.[49] The second advantage is that in our FC the verification algorithm admits *preprocessing*, that is, after spending $O(n)$ group exponentiations for a deterministic preprocessing of the function $\boldsymbol{f}$, the rest of the verification has a fixed cost $O(\lambda)$. Notably, inner product arguments based on the folding techniques of Bootle et al. [44] do not

[49] One could use Bulletproofs arithmetic circuit protocol in order to simulate $\mod p$ algebra over $\mathbb{Z}_q$, at the price of a prover's overhead.

admit this preprocessing, as their verification time is $O(n)$ independently of the time to read the statement.

7.2 Building Blocks

7.2.1 Succinct Proofs of Exponentiation

We make use of the following succinct arguments of knowledge of exponents over groups of unknown order. Below we describe the protocols' functionalities and defer their description to Section 7.6.

PoKE. First we recall the proof of knowledge of exponent (PoKE) of [40] for the language:

$$\mathcal{L}_{\mathsf{PoKE}} = \left\{(Y, u; x) \in \mathbb{G}^2 \times \mathbb{Z} : Y = u^x\right\}$$

parametrized by a group $\mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$ and a group element $g \leftarrow\!\!\$\ \mathbb{G}$. The protocol is succinct: it consists of 3 $\mathbb{G}$-element and 1 $\mathbb{Z}_{2^\lambda}$-element and the verification time is $O(\lambda)$, both regardless of the size, $\|x\|$, of the witness.

PoDDH. We also recall the proof of knowledge of a Diffie-Hellman tuple (PoDDH) of [59], for the language:

$$\mathcal{L}_{\mathsf{PoDDH}} = \left\{(Y_0, Y_1, Y; x_0, x_1) \in \mathbb{G}^3 \times \mathbb{Z}^2 : g_0^{x_0} = Y_0 \wedge g_1^{x_1} = Y_1 \wedge g^{x_0 x_1} = Y\right\}$$

parametrized by a group $\mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$ and three group elements $g, g_0, g_1 \leftarrow\!\!\$\ \mathbb{G}$. Notice that, unlike the usual DH-tuple, in the above protocol the bases are different and honestly generated in the setup. However, the same protocol can work for the same base, $g = g_0 = g_1$. Similarly to the PoKE, the protocol is succinct: 3 $\mathbb{G}$-elements and 2 $\mathbb{Z}_{2^\lambda}$-elements and $O(\lambda)$.

PoRE. We wil make use of a succinct protocol (PoRE) proving that the exponent of an element $Y = g^x$ lies in a certain range, $x \in [L, R]$.

$$\mathcal{L}_{\mathsf{PoRE}} = \left\{(Y, L, R; x) \in \mathbb{G} \times \mathbb{Z}^3 : L < x < R \wedge g^x = Y\right\}.$$

parametrized by a group $\mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$ and a group element $g \leftarrow\!\!\$\ \mathbb{G}$.

For this we rely on the square-decomposition technique [151, 125]. That is, an integer x is in the range $[L, R]$ if and only if there exist $(x_1, x_2, x_3) \in \mathbb{Z}^3$ such that $4(x-L)(R-x)+1 = \sum_{i=1}^{3} x_i^2$. The proof consists of the following subprotocols (run in parallel):

- For each $i = 1, 2, 3$, the prover computes x_i, sends $Z_i = g^{x_i^2}$ for $i \in [3]$ and involves with the verifier in a succinct argument of knowledge of square exponent (PoSE) proving the validity of the last:
$$\mathcal{L}_{\mathsf{PoSE}} = \left\{(Z_i; x_i) \in \mathbb{G} \times \mathbb{Z} : g^{x_i^2} = Z_i\right\}.$$
PoSE is presented as a stand-alone protocol in Section 7.6 Figure 7.8.
- The prover sends $Y' = g^{(x-L)(R-x)}$ and involves in a PoDDH protocol with the verifier for the tuple (g^{x-L}, g^{R-x}, Y'). Observe that g^{x-L}, g^{R-x} can be computed homomorphically by the verifier from $Y = g^x$, thus don't have to be sent.
- Finally, the verifier merely checks if $Y'^4 \cdot g = \prod_{i=1}^{3} Z_i$.

All the above protocols are knowledge-extractable in the generic group model for groups of unknown order [87, 40].

Non-interactive versions. All protocols can be made non-interactive by the standard Fiat-Shamir transformation [101].[50]

7.3 Our Functional Commitment for binary inner products

In this section we present our core construction of Functional Commitments for binary inner products with constant-size parameters and openings. Precisely, in the scheme we commit to binary vectors $\boldsymbol{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$ and the class of functions is $\mathcal{F} = \{\mathcal{F}_n\}$ where, for every positive integer n, $\mathcal{F}_n = \{f : \{0,1\}^n \to \mathbb{Z}\}$ such that f is a linear function represented as a vector of binary coefficients, i.e., $\boldsymbol{f} = (f_1, \ldots, f_n) \in \{0,1\}^n$, and computes the result as the inner product

$$y = \langle \boldsymbol{f}, \boldsymbol{v} \rangle = \sum_{i=1}^{n} f_i \cdot v_i \in \mathbb{Z}.$$

Note that, for a fixed n, every possible result y is an integer in $\{0, \ldots, n\}$. Our starting point is the vector commitment (VC) of Campanelli et. al. [59], which is based on RSA accumulators [29, 18, 152, 40]. In [59], each position of $\boldsymbol{v}$ is encoded as a prime, via a collision-resistant hash-to-prime function $\mathsf{Hprime}(i) \to p_i$ for each $i \in [n]$. Then, in order to commit to $\boldsymbol{v}$, one creates two RSA accumulators, C_0, C_1: the former that accumulates all primes corresponding to zero-values of $\boldsymbol{v}$ ($\{p_i = \mathsf{Hprime}(i) : v_i = 0\}$), and the latter for one-values ($\{p_i = \mathsf{Hprime}(i) : v_i = 1\}$) respectively. That is merely, $C_0 = g_0^{\prod_{v_i=0} p_i}$ and $C_1 = g_1^{\prod_{v_i=1} p_i}$. Observe that these two sets of primes form a partition of all the primes corresponding to positions $\{1, \ldots, n\}$. For binding of the commitment, they also add a succinct proof PoDDH to show that the sets in C_0 and C_1 are indeed a partition.

Starting from this vector commitment, our contribution is a new technique that allows us to create inner product opening proofs. To this end, our first key observation is that:

$$y = \langle \boldsymbol{v}, \boldsymbol{f} \rangle = \sum_i v_i f_i = \sum_{f_i=0} v_i \cdot 0 + \sum_{f_i=1} v_i \cdot 1 = \sum_{f_i=1} v_i = |\{i \in [n] : f_i = 1, v_i = 1\}|$$

since both v_i and f_i are binary. Then the prover commits to the subvector of $\boldsymbol{v}$ corresponding to positions where $f_i = 1$. This is done by using the same vector commitment described previously. That is, we compute $F_0 = g_0^{\prod_{f_i=1, v_i=0} p_i}$ and $F_1 = g_1^{\prod_{f_i=1, v_i=1} p_i}$, accompanied with a PoDDH proof π'_{PoDDH} of Diffie-Hellman tuple for the tuple (F_0, F_1, F), for $F = g^{\prod_{f_i=1} p_i}$. Notice that F can be computed by only knowing $\boldsymbol{f}$, without knowledge of $\boldsymbol{v}$.

The next step to prove the inner product is to show that (F_0, F_1) is actually a commitment to a subvector of $\boldsymbol{v}$. This is done by showing that F_0 accumulates a subset of the primes of C_0, and similarly F_1 accumulates a subset of the primes of C_1. Putting it in other words, the 'exponent' of F_b is contained in the accumulator C_b: there is a W_b such that $W_b^{\prod_{f_i=1, v_i=b} p_i} = C_b$ and $F_b = g_b^{\prod_{f_i=1, v_i=b} p_i}$, for $b = 0, 1$. The last can be proven in a succinct way via a simple concatenation of two PoKE proofs, π_0, π_1.

[50] In these types of proofs though one should set ℓ to be of size 2λ for the non-interactive case [39].

Observe now that the F_1 accumulates exactly the (primes corresponding to) positions that contribute to the inner product $\{i \in [n] : f_i = 1, v_i = 1\}$. The number of primes that F_1 contains in its 'exponent' is exactly y. All that is missing now is a way to convince the verifier about the number of primes in the exponent of F_1. For this, we set the size of each prime p_i to be such that the range of any product $\prod_{i\in\mathcal{I}} p_i$ determines uniquely the cardinality of $\mathcal{I}$ (i.e., the number of primes in the product). This way, a range proof for the 'exponent' of F_1 can convince the verifier about the cardinality of the accumulated set, which is the inner product result y. For this, we generate a succinct range proof π_{PoRE} using our protocol of section 7.2.1.

The verifier, holding the commitment $(C_0, C_1, \pi_{\mathsf{PoDDH}})$, receives the opening proof $(F_0, F_1, \pi'_{\mathsf{PoDDH}}, W_0, \pi_0, W_1, \pi_1, \pi_{\mathsf{PoRE}})$. It is important to make sure that F_1 contains exactly the primes of positions where $f_i = 1$ and $v_i = 1$. The (F_1, C_1)-'subvector' proof π_1 ensures that $v_i = 1$ for all its primes (since C_1 contains only primes for $v_i = 1$). For the $f_i = 1$ part, the verifier herself computes $F = g^{\prod_{f_i=1} p_i}$ and verifies that (F_0, F_1, F) is a DH tuple through π'_{PoDDH}. This ensures that (1) all the primes in the exponents of F_0, F_1 are for $f_i = 1$ and (2) no position i for $f_i = 1$ was excluded maliciously; all of them were either put in F_0 or F_1. This convinces the verifier that exactly the positions i where $f_i = 1, v_i = 1$ are in the 'exponent' of F_1.

7.3.1 Functional VCs for binary linear functions from range proofs

Here we formally describe our construction. We simplify the notation omitting the indicator $i \in [n]$ from the sums and the products below. For example $\sum_i x_i$ would implicitly mean $\sum_{i\in[n]} x_i$ and $\prod_{v_i=1} x_i$ would implicitly mean $\prod_{i\in[n],v_i=1} x_i$. Furthermore, we use abbreviations for some products we will use that can be found in Figure 7.1.

$\mathsf{prod} = \prod_i \mathsf{Hprime}(i)$	$\mathsf{fprod} = \prod_{f_i=1} \mathsf{Hprime}(i)$
$\mathsf{prod}_0 = \prod_{v_i=0} \mathsf{Hprime}(i)$	$\mathsf{fprod}_0 = \prod_{f_i=1,v_i=0} \mathsf{Hprime}(i)$
$\mathsf{prod}_1 = \prod_{v_i=1} \mathsf{Hprime}(i)$	$\mathsf{fprod}_1 = \prod_{f_i=1,v_i=1} \mathsf{Hprime}(i)$

Figure 7.1: Summary of symbols for the products used in the construction.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: The setup algorithm generates a hidden order group $\mathbb{G} \leftarrow \mathsf{Ggen}(1^\lambda)$ and samples three generators $g, g_0, g_1 \leftarrow_\$ \mathbb{G}$. It determines a collision-resistant function Hprime that maps integers to primes and it returns $\mathsf{pp} = (\mathbb{G}, g, g_0, g_1)$.

$\mathsf{Specialize}(\mathsf{pp}, n) \to \mathsf{pp}_n$: The algorithm samples a collision-resistant function Hprime that maps integers to primes.[51] Computes $\mathsf{prod} = \prod_i \mathsf{Hprime}(i)$ and sets $U_n = g^{\mathsf{prod}}$. Returns $\mathsf{pp}_n = (\mathsf{pp}, \mathsf{Hprime}, U_n)$.

$\mathsf{Com}(\mathsf{pp}_n, \boldsymbol{v}) \to C$: The commitment algorithm takes as input a vector of bits $\boldsymbol{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$. It computes the product of all primes that correspond to a zero-value of the vector (i.e., $v_i = 0$) as $\mathsf{prod}_0 = \prod_{v_i=0} \mathsf{Hprime}(i)$, and similarly $\mathsf{prod}_1 = \prod_{v_i=1} \mathsf{Hprime}(i)$ for the one-values. Next, it computes the accumulators

$$C_0 = g_0^{\mathsf{prod}_0} \qquad \text{and} \qquad C_1 = g_1^{\mathsf{prod}_1}$$

[51] As we discuss next and in more detail in Section 7.3.3, the choice of Hprime depends on n.

and a PoDDH proof $\pi = \mathsf{PoDDH.P}\left((\mathbb{G}, g, g_0, g_1), (C_0, C_1, U_n), (\mathsf{prod}_0, \mathsf{prod}_1)\right)$, which ensures that, given the above (C_0, C_1, U_n), it holds $\mathsf{prod} = \mathsf{prod}_0 \cdot \mathsf{prod}_1$:

$$\mathcal{L} = \left\{(C_0, C_1, U_n; \mathsf{prod}_0, \mathsf{prod}_1) : g_0^{\mathsf{prod}_0} = C_0 \wedge g_1^{\mathsf{prod}_1} = C_1 \wedge g^{\mathsf{prod}_0 \cdot \mathsf{prod}_1} = U_n\right\}$$

Returns $C = (C_0, C_1, \pi)$.

$\mathsf{Open}(\mathsf{pp}_n, C, \boldsymbol{v}, \boldsymbol{f}) \to (y, \Lambda)$: $\boldsymbol{f} = (f_1, \ldots, f_n) \in \{0,1\}^n$ is a vector of bits. The output of the function is

$$y = \langle \boldsymbol{v}, \boldsymbol{f} \rangle = \sum v_i f_i = \sum_{f_i = 0} v_i \cdot 0 + \sum_{f_i = 1} v_i \cdot 1 = \sum_{f_i = 1} v_i$$

Let $\mathsf{fprod} = \prod_{f_i=1} \mathsf{Hprime}(i)$, $\mathsf{fprod}_0 = \prod_{f_i=1, v_i=0} \mathsf{Hprime}(i)$ and $\mathsf{fprod}_1 = \prod_{f_i=1, v_i=1} \mathsf{Hprime}(i)$. Computes $F = g^{\mathsf{fprod}}$ and

$$F_0 = g_0^{\mathsf{fprod}_0} \qquad \text{and} \qquad F_1 = g_1^{\mathsf{fprod}_1}$$

Then computes the following arguments of knowledge:

- π_0: a proof that F_0 contains a 'subvector' of C_0, i.e. a proof for the language:
$$\mathcal{L}_0 = \left\{(F_0, W_0; \mathsf{fprod}_0) : W_0^{\mathsf{fprod}_0} = C_0 \wedge g_0^{\mathsf{fprod}_0} = F_0\right\}$$
- π_1: a proof that F_1 contains a 'subvector' of C_1, i.e. a proof for the language:
$$\mathcal{L}_1 = \left\{(F_1, W_1; \mathsf{fprod}_1) : W_1^{\mathsf{fprod}_1} = C_1 \wedge g_1^{\mathsf{fprod}_1} = F_1\right\}$$
- π_2: a PoDDH for F_0, F_1, F:
$$\mathcal{L}_2 = \left\{(F_0, F_1, F; \mathsf{fprod}_0, \mathsf{fprod}_1) : g_0^{\mathsf{fprod}_0} = F_0 \wedge g_1^{\mathsf{fprod}_1} = F_1 \wedge g^{\mathsf{fprod}_0 \cdot \mathsf{fprod}_1} = F\right\}$$
- π_3: a range proof that fprod_1 is in a certain range $L(y) < \mathsf{fprod}_1 < R(y)$, that is uniquely determined by y. L and R are public functions that depend on Hprime (see Figure 7.2 for their concrete description).
$$\mathcal{L}_3 = \left\{(F_1, y; \mathsf{fprod}_1) : L(y) < \mathsf{fprod}_1 < R(y) \wedge g_1^{\mathsf{fprod}_1} = F_1\right\}$$

Returns $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$

$\mathsf{Ver}(\mathsf{pp}_n, C, \Lambda, \boldsymbol{f}, y) \to b$: It computes $F = g^{\mathsf{fprod}} = g^{\prod_{f_i=1} \mathsf{Hprime}(i)}$ that depends only on $\boldsymbol{f}$ and outputs 1 iff all $\pi, \pi_0, \pi_1, \pi_2, \pi_3$ verify. Notice that computing F is necessary as is an input to the proof π_2.

$\mathsf{Hprime} : [n] \to \left(2^{\kappa(\lambda)}, 2^{\kappa(\lambda) + \frac{\kappa(\lambda)}{n}}\right)$ collision-resistant hash-to-prime function.

$$L(y) = \begin{cases} 2^{\kappa(\lambda) y}, & y \in [n] \\ 1, & y = 0 \end{cases} \qquad \text{and} \quad R(y) = \begin{cases} 2^{(\kappa(\lambda) + \frac{\kappa(\lambda)}{n}) y}, & y \in [n] \\ 1, & y = 0 \end{cases}$$

Figure 7.2: Definitions of the range functions L, R. The functions depend on the range Hprime, which in turn depends on n and λ (specified in the setup and specialize phases respectively).

Remark 22. *For ease of presentation, in the* Open *algorithm, we describe four distinct proofs, $\pi_0, \pi_1, \pi_2, \pi_3$. In order to optimize the proof size, they can be merged into a single proof avoiding redundancies. We present in details the (merged) protocol in Section 7.3.3.*

Determining the hash function and the range We need to find a proper hash-to-prime function and a corresponding range $[L(y), R(y)]$ for fprod_1 such that for any $y = 1, \ldots, n$:

$$\mathsf{fprod}_1 := \prod_{v_i=1, f_i=1} \mathsf{Hprime}(i) \in [L(y), R(y)] \Leftrightarrow \left|\{i \in [n] : v_i = 1, f_i = 1\}\right| = y$$

meaning that a range for the product of the primes should translate to its number of prime factors. And the correspondence should be unique. E.g. $p_2 p_7 p_{11} \in [L, R] \Leftrightarrow 3$ factors $\Leftrightarrow y = 3$. For the degenerate case of $y = 0$, $\mathsf{fprod} = 1$.

The following lemma shows that such Hprime, L, R exist and specifies their parameters:

Lemma 19. *Assume a collsion-resistant function that maps integers to prime numbers,* $\mathsf{Hprime} : [n] \to \left(2^{\kappa(\lambda)}, 2^{\kappa(\lambda)+\frac{\kappa(\lambda)}{n}}\right)$*, parametrized by λ and n, and functions $L : \{0, \ldots, n\} \to \mathbb{Z}$, $R : \{0, \ldots, n\} \to \mathbb{Z}$ such that $L(y) = 2^{\kappa y}$ and $R(y) = 2^{(\kappa+\frac{\kappa}{n})y}$ respectively. Then for any $\mathcal{I} \subseteq [n]$:*

$$\prod_{i \in \mathcal{I}} \mathsf{Hprime}(i) \in [L(y), R(y)] \Leftrightarrow |\mathcal{I}| = y$$

Proof. For any number of factors $y = 1, \ldots, n$ we have $2^{\kappa y} < \prod_{i \in [y]} p_i < 2^{(\kappa+\frac{\kappa}{n})y}$. Since $\kappa y + \frac{\kappa y}{n} < \kappa(y+1)$ for any $y \in [n]$ all ranges are distinct. So the mapping is '1-1'. □

In Section 7.3.3 we discuss concrete instantiations for the function Hprime and consequently L and R.

7.3.2 Security

Correctness. Follows from correctness of the [59] Vector Commitment, correctness of PoKE, PoDDH, PoRE arguments of knowledge and from Lemma 19.

Function Binding. Our proof strategy is the following. Given two openings Λ and Λ' of the same commitment C to distinct outputs y, y', we first use the 'subvector' proofs' extractors $\pi_0, \pi_1, \pi'_0, \pi'_1$ to argue that (the exponents of) (F_0, F_1) and (F'_0, F'_1) are subvectors of C. Then we use the PoDDH's extractors π_2, π'_2 to argue that in fact these subvectors are for the same subset of positions $\mathcal{I}_1 = \{i \in [n] : f_i = 1\}$. For the latter we also use the collision-resistance of Hprime. Then we use the extractors of PoRE π_3, π'_3 for F_1 and F'_1 resp., the fact that $y \neq y'$ (by definition of the game) and lemma 19 to argue that these subvectors (for the same positions) are different. Finally, we argue that this fact, having two different subvectors for the same subset of positions and commitment C, contradicts the position-binding property of the [59] Vector Commitment.

Theorem 30. *Let* Ggen *be a hidden order group generator where the [59] VC is position binding,* PoKE, PoDDH, PoRE *be succinct knowledge-extractable arguments of knowledge and* Hprime *be collision-resistant. Then our functional commitment for binary inner products is function binding.*

Proof. We organize the proof in hybrid arguments. To start with, we define the game G_0 as the original functional binding game of Definition 13, and our goal is to prove that for any PPT adversary $\mathcal{A}$ and any $n \in \mathbb{N}$, $\Pr[G_0 = 1] \in \mathsf{negl}(\lambda)$.

Game G_0:

$G_0 = \mathsf{FuncBind}_{\mathcal{A},\mathsf{FC}}(\lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda') \leftarrow \mathcal{A}(\mathsf{pp})$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda', \boldsymbol{f}, y')$

return b

For any adversary $\mathcal{A}$ against G_0, there exist the extractors of the proof of 'subvector' $\pi_0, \pi_1, \pi'_0, \pi'_1$.

Game G_1: is the same as G_0 except that we execute the extractors $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}'_0, \mathcal{E}'_1$ of $\pi_0, \pi_1, \pi'_0, \pi'_1$, which output (W_0, fprod_0), (W_1, fprod_1), $\left(W'_0, \mathsf{fprod}'_0\right)$ and $\left(W'_1, \mathsf{fprod}'_1\right)$, respectively.

G_1

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda') \leftarrow \mathcal{A}(\mathsf{pp})$

$\boxed{\mathsf{fprod}_0 \leftarrow \mathcal{E}_0(\mathbb{G}, g_0); \mathsf{fprod}_1 \leftarrow \mathcal{E}_1(\mathbb{G}, g_1)}$

$\boxed{\mathsf{fprod}'_0 \leftarrow \mathcal{E}'_0(\mathbb{G}, g_0); \mathsf{fprod}'_1 \leftarrow \mathcal{E}'_1(\mathbb{G}, g_1)}$

$$\boxed{b_{\mathbb{W}} = \bigwedge_{i=0,1} \left((W_i^{\mathsf{fprod}_i} = C_i) \wedge (W'^{\mathsf{fprod}'_i}_i = C_i) \wedge (g_i^{\mathsf{fprod}_i} = F_i) \wedge (g_i{}^{\mathsf{fprod}'_i} = F'_i) \right)}$$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda', \boldsymbol{f}, y')$

$\boxed{\textbf{if } b_{\mathbb{W}} = 0 \textbf{ then } b \leftarrow 0}$

return b

where above $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$, $\Lambda' = (F'_0, F'_1, W'_0, W'_1, \pi'_0, \pi'_1, \pi'_2, \pi'_3)$ and $\mathsf{pp}_n = (\mathbb{G}, g, g_0, g_1, \mathsf{Hprime}, U_n)$.

It is easy to see that the games G_0 and G_1 are identical except if $b_{\mathbb{W}} = 0$. However, $b_{\mathbb{W}} = 0$ only when one of the witnesses returned by the extractors is incorrect. By the knowledge extractability of the proof of 'subvector' we obtain that

$$\Pr[G_0 = 1] - \Pr[G_1 = 1] \leq \Pr[b_{\mathbb{W}} = 0] \in \mathsf{negl}(\lambda).$$

Game G_2: is the same as G_1 except the case $\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 \neq \mathsf{fprod}$ or $\mathsf{fprod}'_0 \cdot \mathsf{fprod}'_1 \neq \mathsf{fprod}$.

G_2

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda',) \leftarrow \mathcal{A}(\mathsf{pp})$

$\mathsf{fprod}_0 \leftarrow \mathcal{E}_0(\mathbb{G}, g_0); \mathsf{fprod}_1 \leftarrow \mathcal{E}_1(\mathbb{G}, g_1)$

$\mathsf{fprod}'_0 \leftarrow \mathcal{E}'_0(\mathbb{G}, g_0); \mathsf{fprod}'_1 \leftarrow \mathcal{E}'_1(\mathbb{G}, g_1)$

$b_{\mathrm{w}} = \bigwedge_{i=0,1} \Big((W_i^{\mathsf{fprod}_i} = C_i) \wedge ({W'}_i^{\mathsf{fprod}'_i} = C_i) \wedge (g_i^{\mathsf{fprod}_i} = F_i) \wedge (g_i{}^{\mathsf{fprod}'_i} = F'_i) \Big)$

$\boxed{b_{\mathsf{col}} = (\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 = \mathsf{fprod}) \wedge \big(\mathsf{fprod}'_0 \cdot \mathsf{fprod}'_1 = \mathsf{fprod}\big)}$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda', \boldsymbol{f}, y')$

$\boxed{\textbf{if } b_{\mathrm{w}} = 0 \vee b_{\mathsf{col}} = 0 \textbf{ then } b \leftarrow 0}$

$\textbf{return } b$

where above $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$, $\Lambda' = (F'_0, F'_1, W'_0, W'_1, \pi'_0, \pi'_1, \pi'_2, \pi'_3)$ and $\mathsf{pp}_n = (\mathbb{G}, g, g_0, g_1, \mathsf{Hprime}, U_n)$.

Lemma 20. *Assume that the Low order assumption holds for* Ggen*, then* $\Pr[G_1 = 1] - \Pr[G_2 = 1] \in \mathsf{negl}(\lambda)$.

Proof. We consider a game G'_2 by running the extractors of the PoDDH proofs π_2 and π'_2, which outputs $(\overline{\mathsf{fprod}_0}, \overline{\mathsf{fprod}_1})$ s.t. $g_0^{\overline{\mathsf{fprod}_0}} = F_0 \wedge g_1^{\overline{\mathsf{fprod}_1}} = F_1 \wedge g^{\overline{\mathsf{fprod}_0} \cdot \overline{\mathsf{fprod}_1}} = F$ and $(\overline{\mathsf{fprod}_0}', \overline{\mathsf{fprod}_1}')$ s.t. $g_0^{\overline{\mathsf{fprod}_0}'} = F'_0 \wedge g_1^{\overline{\mathsf{fprod}_1}'} = F'_1 \wedge g^{\overline{\mathsf{fprod}_0}' \cdot \overline{\mathsf{fprod}_1}'} = F$, respectively. It is easy to see that $\Pr[G_2 = 1] - \Pr[G'_2 = 1] \in \mathsf{negl}(\lambda)$. The following equalities hold:

$$g^{\overline{\mathsf{fprod}_0} \cdot \overline{\mathsf{fprod}_1}} = g^{\overline{\mathsf{fprod}_0}' \cdot \overline{\mathsf{fprod}_1}'} = F;$$

$$g_0{}^{\overline{\mathsf{fprod}_0}} = F_0; \quad g_0{}^{\overline{\mathsf{fprod}_0}'} = F'_0;$$

$$g_1{}^{\overline{\mathsf{fprod}_1}} = F_1 \text{ and } g_1{}^{\overline{\mathsf{fprod}_1}'} = F'_1.$$

The game G'_2 only differs from G_1 in the case $(\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 \neq \mathsf{fprod}) \vee \big(\mathsf{fprod}'_0 \cdot \mathsf{fprod}'_1 \neq \mathsf{fprod}\big)$. We divide it into two cases.

Case 1: $\overline{\mathsf{fprod}_0} \cdot \overline{\mathsf{fprod}_1} \neq \mathsf{fprod}$ or $\overline{\mathsf{fprod}'_0} \cdot \overline{\mathsf{fprod}'_1} \neq \mathsf{fprod}$. The first inequality implies that there exists $1 \neq v = \mathsf{fprod} - \overline{\mathsf{fprod}_0} \cdot \overline{\mathsf{fprod}_1} < 2^{\mathsf{poly}}$ such that $g^v = 1$, which is a violation to the Low order assumption and occurs with a negligible probability. The second one implies a similar result.

Case 2: the two above equalities hold. Then G'_2 differs from G_1 only if one of the four following inequalities holds.

$$\overline{\mathsf{fprod}_0} \neq \mathsf{fprod}_0; \overline{\mathsf{fprod}_0}' \neq \mathsf{fprod}'_0; \overline{\mathsf{fprod}_1} \neq \mathsf{fprod}_1 \text{ or } \overline{\mathsf{fprod}_1}' \neq \mathsf{fprod}'_1.$$

Again, any of these inequalities imply a low order root of 1.

Therefore, $\Pr[G_1 = 1] - \Pr[G_2 = 1] \in \mathsf{negl}(\lambda)$. □

Game G_3: is obtained by adding the range check for fprod_1 and fprod'_1 to G_2.

G_3

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda',) \leftarrow \mathcal{A}(\mathsf{pp})$

$\mathsf{fprod}_0 \leftarrow \mathcal{E}_0(\mathbb{G}, g_0); \mathsf{fprod}_1 \leftarrow \mathcal{E}_1(\mathbb{G}, g_1)$

$\mathsf{fprod}'_0 \leftarrow \mathcal{E}'_0(\mathbb{G}, g_0); \mathsf{fprod}'_1 \leftarrow \mathcal{E}'_1(\mathbb{G}, g_1)$

$b_{\mathbb{W}} = \bigwedge_{i=0,1} \Big((W_i^{\mathsf{fprod}_i} = C_i) \wedge (W'^{\mathsf{fprod}'_i}_i = C_i) \wedge (g_i^{\mathsf{fprod}_i} = F_i) \wedge ({g_i}^{\mathsf{fprod}'_i} = F'_i) \Big)$

$b_{\mathsf{col}} = (\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 = \mathsf{fprod}) \wedge \big(\mathsf{fprod}'_0 \cdot \mathsf{fprod}'_1 = \mathsf{fprod}\big)$

$\boxed{b_{\mathsf{range}} = (\mathsf{fprod}_1 \in [L(y), R(y)]) \wedge \big(\mathsf{fprod}'_1 \in [L(y'), R(y')]\big)}$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}, C, \Lambda', \boldsymbol{f}, y')$

$\boxed{\textbf{if } b_{\mathbb{W}} = 0 \vee b_{\mathsf{col}} = 0 \vee b_{\mathsf{range}} = 0 \textbf{ then } b \leftarrow 0}$

$\textbf{return } b$

where above $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$, $\Lambda' = (F'_0, F'_1, W'_0, W'_1, \pi'_0, \pi'_1, \pi'_2, \pi'_3)$ and $\mathsf{pp}_n = (\mathbb{G}, g, g_0, g_1, \mathsf{Hprime}, U_n)$.

The two games G_2 and G_3 are different only in the case the range check fails, which means either π_3 or π'_3 is incorrect. By the argument of knowledge of PoRE, we conclude that $\Pr[G_2 = 1] - \Pr[G_3 = 1] \in \mathsf{negl}(\lambda)$.

Game G_4: is the same as G_3 except we decode the subvectors $\boldsymbol{s}, \boldsymbol{s'}$ from the exponents fprod_1 and fprod'_1 corresponding to the subset $\{i : f_i = 1\}$. Then we recompute the exponents from the prime products at 0-positions and 1-positions and check if they coincide with fprod_0, fprod_1, fprod'_0 and fprod'_1.

G_4

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda') \leftarrow \mathcal{A}(\mathsf{pp})$

$\mathsf{fprod}_0 \leftarrow \mathcal{E}_0(\mathbb{G}, g_0); \mathsf{fprod}_1 \leftarrow \mathcal{E}_1(\mathbb{G}, g_1)$

$\boxed{\textbf{for } i : f_i = 1 \textbf{ do}}$

$\boxed{p_i \leftarrow \mathsf{Hprime}(i); s_i = (p_i \mid \mathsf{fprod}_1); s'_i = (p_i \mid \mathsf{fprod}'_1)}$

$\boxed{\boldsymbol{s} = \{s_i\}_{f_i=1}; \boldsymbol{s'} = \{s'_i\}_{f_i=1}}$

$\mathsf{fprod}'_0 \leftarrow \mathcal{E}'_0(\mathbb{G}, g_0); \mathsf{fprod}'_1 \leftarrow \mathcal{E}'_1(\mathbb{G}, g_1)$

$b_{\mathrm{w}} = \bigwedge_{i=0,1} \left((W_i^{\mathsf{fprod}_i} = C_i) \wedge (W'^{\mathsf{fprod}'_i}_i = C_i) \wedge (g_i^{\mathsf{fprod}_i} = F_i) \wedge (g_i^{\mathsf{fprod}'_i} = F'_i) \right)$

$b_{\mathsf{col}} = (\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 = \mathsf{fprod}) \wedge \left(\mathsf{fprod}'_0 \cdot \mathsf{fprod}'_1 = \mathsf{fprod}\right)$

$b_{\mathsf{range}} = (\mathsf{fprod}_1 \in [L(y), R(y)]) \wedge \left(\mathsf{fprod}'_1 \in [L(y'), R(y')]\right)$

$$\boxed{b_{\mathsf{subv}} = \left(\mathsf{fprod}_0 = \prod_{f_i=1, s_i=0} \mathsf{Hprime}(i) \wedge \mathsf{fprod}'_0 = \prod_{f_i=1 : s'_i=0} \mathsf{Hprime}(i) \right)}$$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}, C, \Lambda', \boldsymbol{f}, y')$

$\boxed{\textbf{if } b_{\mathsf{subv}} = 0 \vee b_{\mathrm{w}} = 0 \vee b_{\mathsf{col}} = 0 \vee b_{\mathsf{range}} = 0 \textbf{ then } b \leftarrow 0}$

$\textbf{return } b$

where above $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$, $\Lambda' = (F'_0, F'_1, W'_0, W'_1, \pi'_0, \pi'_1, \pi'_2, \pi'_3)$, $\mathsf{pp}_n = (\mathbb{G}, g, g_0, g_1, \mathsf{Hprime}, U_n)$.

Lemma 21. *Assume that* Hprime *is collision-resistant,* $\Pr[G_3 = 1] - \Pr[G_4 = 1] \in \mathsf{negl}(\lambda)$.

Proof. The output of G_4 differs from G_3's only when $b_{\mathsf{subv}} = 0 \wedge b_{\mathrm{w}} = 1 \wedge b_{\mathsf{col}} = 1$. By assumption, for all i such that $f_i = 1$, $\mathsf{Hprime}(i)$'s are all distinct. Then by definition of $\boldsymbol{s}$,

$$\prod_{f_i=1, s_i=1} \mathsf{Hprime}(i) \mid \mathsf{fprod}_1 \text{ and } \prod_{f_i=1, s_i=0} \mathsf{Hprime}(i) \mid \mathsf{fprod}_0. \tag{7.1}$$

Multiplying both hand sides of the two equations, we obtain $\prod_{f_i=1} \mathsf{Hprime}(i) \mid \mathsf{fprod}$. Notice that the equality holds, hence it also holds for the equations in (7.1). □

Lemma 22. *Let* Ggen *be a hidden order group generator where the [59] SVC is position binding, then* $\Pr[G_4 = 1] \in \mathsf{negl}(\lambda)$.

Proof. We show that given any adversary $(\mathcal{A}, \mathcal{E})$, where $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}'_0, \mathcal{E}'_1)$, winning game G_4 with non-negligible advantage, we can construct $\mathcal{B}$ winning the position-binding game of the [59] VC.

Indeed, $\mathcal{B}$ on input pp outputs $(C, \mathcal{I}_1, \boldsymbol{s}, (W_0, W_1), \boldsymbol{s'}, (W'_0, W'_1))$, where $\mathcal{I}_1 = \{i \in [n] : f_i = 1\}$ and $(W_0, W_1), (W'_0, W'_1)$ play the role of the (accepting) opening proofs for the [59] SVC. The verifier for position binding computes $a_j = \prod_{f_i=1, s_i=j} \mathsf{Hprime}(i); a'_j =$

$\prod_{f_i=1,s'_i=j} \mathsf{Hprime}(i)$ for $j=0,1$. Since $b_{\mathsf{subv}}=1$, $a_j=\mathsf{fprod}_j$ and $a'_j=\mathsf{fprod}'_j$ for $j=0,1$. Next, since $b_{\mathsf{W}}=1$, $W_j^{a_j}=C_j$ and $W_j'^{a'_j}=C_j$, for $j=0,1$.

Now since $y\neq y'$, $b_{\mathsf{range}}=1$ and from lemma 19 we infer that $\mathsf{fprod}_1\neq\mathsf{fprod}'_1$ (because they have different number of factors) so $\boldsymbol{s}\neq\boldsymbol{s'}$. Combining these arguments, we obtain that $\mathsf{Ver}(C,\mathcal{I}_1,\boldsymbol{s},,(W_0,W_1))=1$, $\mathsf{Ver}(C,\mathcal{I}_1,\boldsymbol{s'},,(W'_0,W'_1))=1$ (for the VC) and $\boldsymbol{s}\neq\boldsymbol{s'}$, i.e., $\mathcal{B}$ succeeds in the position-binding game with the same probability of $(\mathcal{A},\mathcal{E})$. □

By combining all the lemmas we conclude that any PPT adversary has at most negligible probability of breaking the function binding of our SVC scheme. □

7.3.3 Instantiation

Instatiation of Hprime**.** As stated in Lemma 19, Hprime should be a collision-resistant function with domain $[n]$ that outputs prime numbers in the range $\left(2^{\kappa(\lambda)},2^{\kappa(\lambda)+\frac{\kappa(\lambda)}{n}}\right)$. Here we specify the function $\kappa(\cdot)$ and show instantiations for Hprime under these restrictions.

Hashing to primes is a well studied problem [115, 84, 55]. A standard technique is rejection sampling: on input x it computes $y=F_K(x,0)$, where F_K is a pseudorandom function with seed K and range $[A,B]$, and checks y for primality. If y is not prime it continues to $y=F_K(x,1)$ and so on, until it finds a prime $F(x,j)$ for some j. From the density of primes the expected number of tries is $\log(B-A)$. As an alternative, F_K can also be a random oracle.

Assume a collision-resistant hash function H that we model as a random oracle and its outputs are in the range $\left(2^{\kappa(\lambda)},2^{\kappa(\lambda)+\frac{\kappa(\lambda)}{n}}\right)$.[52] For H to be collision resistant we require, due to the birthday bound, its range to contain at least $2^{2\lambda}$ prime numbers. From the density of primes we know that in the above range there are about:

$$\frac{2^{\kappa+\frac{\kappa}{n}}}{\kappa+\frac{\kappa}{n}}-\frac{2^{\kappa}}{\kappa}\approx\frac{2^{\kappa+\frac{\kappa}{n}}}{\kappa}-\frac{2^{\kappa}}{\kappa}=\frac{2^{\kappa}\left(2^{\frac{\kappa}{n}}-1\right)}{\kappa}$$

prime numbers, where for the first approximate equality we assumed that $\kappa\ll n$.

So if we set κ (depending on λ and n) to be the smallest positive integer such that:

$$\frac{2^{\kappa}\left(2^{\frac{\kappa}{n}}-1\right)}{\kappa}\geq 2^{2\lambda}$$

then H gives sufficiently many primes to instantiate Hprime (via the rejection sampling method we described above).

For example for $n=2^{60}$ and $\lambda=128$: $\kappa=317$. So instantiating Hprime with the rejection sampling method based on the SHA512 (for F_K) fixing its output range to $(2^{317},2^{317+317/2^{60}})$ we get a sufficient Hprime for our functional vector commitment construction that satisfies lemma 19.

Instantiation of the Arguments of Knowledge. Here we present the merged argument of knowledge for our Open algorithm of section 7.3.1. As noted, it was presented modularly in order to ease the presentation of the protocol and its security proof, however we can merge

[52] We can securely fix the range of a hash function (as SHA512) by fixing some of its bits and truncating others.

the proofs for the four languages $\mathcal{L}_0 - -\mathcal{L}_3$ into a single protocol, using standard composition techniques. The unified language of the Open algorithm is:

$$\mathcal{L} = \left\{ \begin{array}{r} (F_0, F_1, F, W_0, W_1, R, L; \mathsf{fprod}_0, \mathsf{fprod}_1) : \\ W_0^{\mathsf{fprod}_0} = C_0 \wedge g_0^{\mathsf{fprod}_0} = F_0 \wedge \\ \wedge W_1^{\mathsf{fprod}_1} = C_1 \wedge g_1^{\mathsf{fprod}_1} = F_1 \wedge \\ \wedge g^{\mathsf{fprod}_0 \cdot \mathsf{fprod}_1} = F \wedge L < \mathsf{fprod}_1 < R \end{array} \right\}$$

The description of the protocol is in Figure 7.3.

The protocol gets non-interactive after the Fiat-Shamir transform. We note that for ℓ an instantiation of the random oracle with a hash-to-prime function of outputs in $\mathbb{P}(1, 2^{2\lambda})$ suffices.

Concrete Security Assumptions After the above instantiations we get that our overall binary inner product commitment is secure in the GGM and RO model assuming that H (used for Hprime as described above) is collision-resistant: the argument of knowledge of Figure 7.3 is knowledge-extractable in the generic group model and gets non-interactive in the random oracle model and the [59] SVC is position binding under the 2-strong RSA and Low order assumptions (that are secure in the GGM).

7.3.4 Efficiency

Our FC for binary inner products has $O(1)$ public parameters, $O(1)$ commitment size and $O(1)$ openings proof size. More in detail, $|\mathsf{pp}_n|$ consists of 4 $|\mathbb{G}|$-elements and the descriptions of $\mathbb{G}$ and Hprime (which are concise). $|C|$ is 5 $|\mathbb{G}|$-elements and 2 $|\mathbb{Z}_{2^{2\lambda}}|$-elements. Finally the opening proof $|\Lambda|$ is 21 $|\mathbb{G}|$-elements and 7 $|\mathbb{Z}_{2^{2\lambda}}|$-elements.[53]

Generating the public parameters, via Setup and Specialize, takes a $\mathbb{G}$-exponentiation of size $\kappa n = O_\lambda(n)$. The generation doesn't require any private coins and thus is transparent.

The prover's time (i.e., the running time of Open) is dominated by the computation of the square decomposition x_1, x_2, x_3, that is Pollack and Trevino algorithm [180] on input of size $2\kappa n$ running in time $O\left((2\kappa n)^2 / \log(2\kappa n)\right)$. Therefore, our prover requires $O_\lambda(\lambda n^2 / \log(\lambda n))$ integer operations and $O(n)$ group exponentiations.

The verifier's running time (i.e. Ver) is dominated by the computation of $F = g^{\prod_{f_i=1} \mathsf{Hprime}(i)}$, which takes (in the worst case where $\boldsymbol{f} = (1, 1, \ldots, 1)$) a $\mathbb{G}$-exponentiation of size $\kappa n = O_\lambda(n)$. The rest of the computations, i.e. the verification of the argument of knowledge, take constant time $O(2\lambda) = O_\lambda(1)$ $\mathbb{G}$-exponentiations. However, below we make two observations that can speed up the verification in two useful ways.

Preprocessing-based verification. Our Functional Commitment construction allows preprocessing the verification (see Remark 2). The verifier can compute a-priori the function-dependent value F so that the online verification gets $O_\lambda(n)$. Notably, this preprocessing is deterministic and proof-independent, and thus can be reused to verify an unbounded number of openings for the same function f.

From group-based to integers-based linear work. Even without preprocessing, the prover can compute and send to the verifier a PoE proof [215, 40] (see Section 7.6) for $F = g^{\prod_{f_i=1} \mathsf{Hprime}(i)}$.

[53] We do not consider optimizations for the arguments of knowledge with which we could reduce the size of $|C|$ by 1 and $|\Lambda|$ by 6 group elements respectively.

Then the verifier verifies PoE instead of computing F herself. This takes $O_\lambda(n)$ integer operations and $O_\lambda(1)$ group exponentiations, in place of $O_\lambda(n)$ group operations, which concretely gives a significant saving.

7.4 Our FC for Inner Products Over the Integers

In this section, we present two transformations that turn any functional commitment for binary inner products (like the one we presented in Section 7.3) into a functional commitment for inner products of vectors of (bounded) integers. Precisely, we build an FC where one commits to vectors $\boldsymbol{v} \in (\mathbb{Z}_{2^\ell})^n$ and the class of admissible functions is

$$\mathcal{F}_n = \{f : (\mathbb{Z}_{2^\ell})^n \to \mathbb{Z}\}$$

where each f is represented as a vector $\boldsymbol{f} \in (\mathbb{Z}_{2^m})^n$.

Consider an FC scheme bitFC for binary inner products, and let $t_{\mathsf{Com}}(n)$, $t_{\mathsf{Open}}(n)$, $t_{\mathsf{Ver}}(n)$ be the running times of its algorithms Com, Open and Ver respectively, and let $s(n)$ be the size of its openings.

Our two transformations yield FCs for the integer inner products functionality $\mathcal{F}$ that achieve different tradeoffs:

1. With our first transformation we obtain an FC where

$$t'_{\mathsf{Com}}(n) = t_{\mathsf{Com}}(n\ell),\ t'_{\mathsf{Open}}(n) = (\ell+m)\cdot t_{\mathsf{Open}}(n\ell),\ t'_{\mathsf{Ver}}(n) = (\ell+m)\cdot t_{\mathsf{Ver}}(n\ell),$$

$$s'(n) = (\ell+m)\cdot(s(n\ell)+\log(n\ell))$$

2. With our second transformation we obtain an FC where

$$t'_{\mathsf{Com}}(n) = t_{\mathsf{Com}}(n\ell 2^{\ell+m}),\ t'_{\mathsf{Open}}(n) = t_{\mathsf{Open}}(n\ell 2^{\ell+m}),\ t'_{\mathsf{Ver}}(n) = t_{\mathsf{Ver}}(n\ell 2^{\ell+m}),$$

$$s'(n) = s(n\ell 2^{\ell+m})$$

Given the tradeoffs, and considering instantiations of bitFC like ours, in which $s(n)$ is a fixed value in the security parameters, then the second transformation is particularly interesting in the case $\ell, m = O(1)$ are constant or $O(\log\lambda)$ as it yields an FC with constant, or polynomial, time overhead and constant-size openings.

7.4.1 Our lifting to FC for integer inner products with logarithmic-size openings

We start by providing here an intuitive description of our transformation. We give a formal description of the FC scheme slightly below.

For our transformation, we use a binary representation of the vectors of integers $\boldsymbol{v} \in (\mathbb{Z}_{2^\ell})^n$ and $\boldsymbol{f} \in (\mathbb{Z}_{2^m})^n$, that is:

$$\boldsymbol{v} = (v_1,\ldots,v_n) \in (\{0,1\}^\ell)^n \quad \text{and} \quad \boldsymbol{f} = (f_1,\ldots,f_n) \in (\{0,1\}^m)^n$$

Denote $v_i = \sum_{j=0}^{\ell-1} v_i^{(j)} 2^j$ and $f_i = \sum_{k=0}^{m-1} f_i^{(k)} 2^k$ the bit decomposition of v_i, f_i respectively. Then we can rewrite the inner product of $\boldsymbol{v}$ and $\boldsymbol{f}$ as

$$y = \langle \boldsymbol{f}, \boldsymbol{v} \rangle = \sum_{i=1}^{n} v_i f_i = \sum_{i=1}^{n} \left(\sum_{j=0}^{\ell-1} \sum_{k=0}^{m-1} v_i^{(j)} f_i^{(k)} 2^{j+k} \right). \tag{7.2}$$

If we swap the counters we conclude to:

$$y = \sum_{j=0}^{\ell-1} \sum_{k=0}^{m-1} \left(\sum_{i=1}^{n} v_i^{(j)} f_i^{(k)} \right) 2^{j+k} = \sum_{j=0}^{\ell-1} \sum_{k=0}^{m-1} \langle \boldsymbol{v}^{(j)}, \boldsymbol{f}^{(k)} \rangle 2^{j+k}$$

where above $\boldsymbol{v}^{(j)} = (v_1^{(j)}, \ldots, v_n^{(j)}) \in \{0,1\}^n$ is a bit-vector of the j-th bits of all entries v_i (and similarly for $\boldsymbol{f}^{(k)}$). The inner product y of $\boldsymbol{v}$ and $\boldsymbol{f}$ is hereby broken into the above sum of ℓm binary inner products. So, a first idea to open the inner product over the integers would be to let one create ℓ commitments, one for each $\boldsymbol{v}^{(j)}$ of length n, and then open to the inner product y by revealing all the ℓm binary inner products, each with its corresponding opening proof. The issue with this idea is that it yields an $O(\ell m \log n)$-size opening.

Next, we show a more efficient way to use an FC for binary inner product that avoids this quadratic blowup.

To this end, we show that y can also be represented as the sum of $\ell + m$ binary inner products between vectors of length $n\ell$. We start observing that we can rewrite (7.2) as

$$y = \sum_{i=1}^{n} \sum_{j=0}^{\ell-1} v_i^{(j)} \cdot \left(\sum_{k=0}^{m-1} f_i^{(k)} 2^{j+k} \right) = \sum_{i=1}^{n} \sum_{j=0}^{\ell-1} v_i^{(j)} \cdot \hat{f}_i^{(j)} \tag{7.3}$$

where, for every i, j, each $\hat{f}_i^{(j)}$ is the integer $\sum_{k=0}^{m-1} f_i^{(k)} \cdot 2^{j+k} \in [0, 2^{\ell+m-1}]$. Now the inner product y over integers is reshaped as an inner product between an $n\ell$-long binary vector

$$\boldsymbol{v}' = \boldsymbol{v}^{(0)} \| \ldots \| \boldsymbol{v}^{(\ell-1)}$$

and an $n\ell$-long function with coefficients in $[0, 2^{\ell+m-1}]$. It is left to appropriately grind this inner product into binary inner products. We are about to show that those binary inner products are between $\boldsymbol{v}'$ and the following binary vectors

$$\boldsymbol{f}'_h = \boldsymbol{f}^{(h)} \| \boldsymbol{f}^{(h-1)} \| \ldots \| \boldsymbol{f}^{(h-\ell+1)}$$

where $h \in [0, \ell + m - 2]$, $\boldsymbol{f}^{(k)} = (0, \ldots, 0)$ for all $k \notin [0, m-1]$.

Indeed, let $y'_h = \langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle$. Then by changing the variable $k = h - j$ and rearranging the summation of j, we can rewrite (7.2) as

$$y = \sum_{h=0}^{\ell+m-2} \sum_{j=0}^{\ell-1} \langle \boldsymbol{v}^{(j)}, \boldsymbol{f}^{(h-j)} \rangle \cdot 2^h = \sum_{h=0}^{\ell+m-2} \langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle \cdot 2^h = \sum_{h=0}^{\ell+m-2} y'_h \cdot 2^h.$$

Using as a building block the binary functional vector commitment we get a functional vector commitment for bounded-integers as follows: only one commitment C is needed for the concatenating vector $\boldsymbol{v}'$. Then the opening proof consists of the partial outputs $\{y_h\}_{h \in [0, \ell+m-2]}$

together with their corresponding functional opening proofs $\{\Lambda_h\}_{h\in[0,\ell+m-2]}$, one for each binary inner product $\langle \boldsymbol{v}', \boldsymbol{f}'_h\rangle$. For verification, one is checking that each Λ_h verifies with respect to C and f'_h, to ensure that y_h are the correct partial outputs. Then it reconstructs $y = \sum_{h=0}^{\ell+m-2} y_h 2^h$ according to the above equality.

FC scheme Consider bitFC as an arbitrary FC for binary inner products, we present below a formal description of the transformation.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: runs $\mathsf{pp} = \mathsf{bitFC.Setup}(1^\lambda)$. Returns pp

$\mathsf{Specialize}(\mathsf{pp}, \mathcal{F}_n) \to \mathsf{pp}_{\mathcal{F}_n}$: given the description of the functions class $\mathcal{F}_n$, which includes the bounds ℓ, m and the vector length n, the specialization algorithms sets $N = n\ell$ and returns $\mathsf{pp}_{\mathcal{F}_n} = \mathsf{bitFC.Specialize}(\mathsf{pp}, N)$.

$\mathsf{Com}(\mathsf{pp}_{n,\ell}, \boldsymbol{v}) \to C$: Let $\boldsymbol{v} = (v_1, \ldots, v_n) \in (\{0,1\}^\ell)^n$ be a vector of ℓ-bit entries, and let $\boldsymbol{v}^{(j)} = (v_1^{(j)}, \ldots, v_n^{(j)})$ be the binary vector expressing the j-th bit of all entries in $\boldsymbol{v}$, i.e., it holds $\boldsymbol{v} = \left(\sum_{j=0}^{\ell-1} v_1^{(j)} 2^j, \ldots, \sum_{j=0}^{\ell-1} v_n^{(j)} 2^j\right)$.

The commitment algorithm computes the commitment

$$C = \mathsf{bitFC.Com}(\mathsf{pp}_{\mathcal{F}_n}, \boldsymbol{v}'), \text{ s.t. } \boldsymbol{v}' = \boldsymbol{v}^{(0)} \| \ldots \| \boldsymbol{v}^{(\ell-1)}$$

and returns C.

$\mathsf{Open}(\mathsf{pp}_{\mathcal{F}_n}, C, \boldsymbol{v}, \boldsymbol{f}) \to (y, \Lambda)$: $\boldsymbol{f} = (f_1, \ldots, f_n) \in (\{0,1\}^m)^n$ is a vector of m-bit entries.

If $\boldsymbol{f} = \left(\sum_{k=0}^{m-1} f_1^{(k)} 2^k, \ldots, \sum_{k=0}^{m-1} f_n^{(k)} 2^k\right)$ then $\boldsymbol{f}^{(k)} = (f_1^{(k)}, \ldots, f_n^{(k)})$ is the binary vector of the k-th bit of all entries of $\boldsymbol{f}$.

The opening algorithm proceeds as follows. For each $h = 0, \ldots, \ell + m - 2$:

$$\text{set } \boldsymbol{f}'_h = \boldsymbol{f}^{(h)} \| \boldsymbol{f}^{(h-1)} \| \ldots \| \boldsymbol{f}^{(h-\ell+1)}, \text{ where } \boldsymbol{f}^{(i)} = (0, \ldots, 0) \forall i \notin [0, m-1],$$

$$\text{and compute } y_h = \langle \boldsymbol{v}', \boldsymbol{f}'^h \rangle \text{ and } \Lambda_h = \mathsf{bitFC.Open}(\mathsf{pp}_{\mathcal{F}_n}, C, \boldsymbol{v}', \boldsymbol{f}'_h),$$

Return $\Lambda = \{y_h, \Lambda_h\}_{h\in[0,\ell+m-2]}$.

$\mathsf{Ver}(\mathsf{pp}_{\mathcal{F}_n}, C, \Lambda, \boldsymbol{f}, y) \to b$: returns 1 iff:

1. $\mathsf{bitFC.Ver}\left(\mathsf{pp}_{\mathcal{F}_n}, C, \Lambda_h, \boldsymbol{f}'_h, y_h\right) = 1$, for each $h \in [0, \ell + m - 2]$.
2. $y = \sum_{h=0}^{\ell+m-2} y_h 2^h$.

Theorem 31. *If the binary functional vector commitment is functional binding, then our bounded-integer functional vector commitment is functional binding.*

Proof. The proof is straightforward from the fact that two valid openings y, z over integer imply immediately that there exists at least an index h for which there are two valid openings for distinct binary inner products $y_h \neq z_h$. □

7.4.2 Our lifting to FC for integer inner products with constant-size openings

Here we provide a different method to lift an FC for binary inner products to an FC for integer inner products that achieves a different tradeoff. The prover time and verification time are $2^{\ell+m}$ times those of the bitFC scheme (with function inputs of bit-size $n\ell 2^{\ell+m}$), while openings are exactly the same as those of bitFC (and thus constant-size using our scheme of Section 7.3).

Intuition In the transformation of the previous section we showed how how to express the inner product $y = \langle \boldsymbol{v}, \boldsymbol{f} \rangle$ of n-long vectors of integers into the weighted sum of $\ell + m - 1$ binary inner products of vectors of length $n\ell$:

$$y = \sum_{h=0}^{\ell+m-2} \langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle \cdot 2^h = \sum_{h=0}^{\ell+m-2} y_h \cdot 2^h$$

The drawback of this transformation is that we need to include all the y_h in the opening, and each of this integer is up to $\log n$-bits long.

It turns out that we can iterate the same idea and encode the above weighted sum into a single inner product $\langle \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{f}} \rangle$ of binary vectors of length $n\ell H$ with $H = \sum_{h=0}^{\ell+m-2} 2^h = 2^{\ell+m-1} - 1$.

For every $h \in [0, \ell + m - 2]$, define the vector $\tilde{\boldsymbol{f}}_h = \boldsymbol{f}'_h \| \cdot \| \boldsymbol{f}'_h \in \{0,1\}^{n\ell 2^h}$, that is the concatenation of 2^h copies of $\boldsymbol{f}'_h$. Similarly, set $\tilde{\boldsymbol{v}}_h = \boldsymbol{v}' \| \cdots \| \boldsymbol{v}' \in \{0,1\}^{n\ell 2^h}$. Next, if we define

$$\tilde{\boldsymbol{v}} = \tilde{\boldsymbol{v}}_h \| \cdots \| \tilde{\boldsymbol{v}}_h \in \{0,1\}^{n\ell H} \text{ and } \tilde{\boldsymbol{f}} = \tilde{\boldsymbol{f}}_0 \| \cdots \| \tilde{\boldsymbol{f}}_{\ell+m-2} \tag{7.4}$$

it can be seen that

$$\langle \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{f}} \rangle = \sum_{h=0}^{\ell+m-2} \langle \tilde{\boldsymbol{v}}_h, \tilde{\boldsymbol{f}}_h \rangle = \sum_{h=0}^{\ell+m-2} \langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle \cdot 2^h = y$$

FC Scheme More in detail the FC scheme works as follows.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: runs $\mathsf{pp} = \mathsf{bitFC.Setup}(1^\lambda)$. Returns pp

$\mathsf{Specialize}(\mathsf{pp}, \mathcal{F}_n) \to \mathsf{pp}_{\mathcal{F}_n}$: given the description of the functions class $\mathcal{F}_n$, which includes the bounds ℓ, m and the vector length n, the specialization algorithms sets $N = n\ell H$, with $H = 2^{\ell+m-1} - 1$, and returns $\mathsf{pp}_{\mathcal{F}_n} = \mathsf{bitFC.Specialize}(\mathsf{pp}, N)$.

$\mathsf{Com}(\mathsf{pp}_{\mathcal{F}_n}, \boldsymbol{v}) \to C$: Given $\boldsymbol{v} = (v_1, \ldots, v_n) \in (\{0,1\}^\ell)^n$, compute a vector $\tilde{\boldsymbol{v}} \in \{0,1\}^{n\ell H}$ as in equation (7.4), and return the commitment

$$C = \mathsf{bitFC.Com}(\mathsf{pp}_{\mathcal{F}_n}, \tilde{\boldsymbol{v}}).$$

$\mathsf{Open}(\mathsf{pp}_{\mathcal{F}_n}, C, \boldsymbol{v}, \boldsymbol{f}) \to \Lambda$: Given $\boldsymbol{f} = (f_1, \ldots, f_n) \in (\{0,1\}^m)^n$, compute vectors $\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{f}} \in \{0,1\}^{n\ell H}$ as in equation (7.4), and return the opening

$$\Lambda = \mathsf{bitFC.Open}(\mathsf{pp}_{\mathcal{F}_n}, \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{f}}).$$

$\mathsf{Ver}(\mathsf{pp}_{\mathcal{F}_n}, C, \Lambda, \boldsymbol{f}, y) \to b$: returns 1 iff $\mathsf{bitFC.Ver}(\mathsf{pp}_{\mathcal{F}_n}, C, \Lambda, \tilde{\boldsymbol{f}}, y) = 1$.

Theorem 32. *If* bitFC *is functional binding, then the FC described above is functional binding.*

The proof is straightforward based on the observation that two valid openings for distinct $y \neq y'$ of our FC are also two valid proofs, for the same commitment and outputs, for the bitFC scheme.

7.5 Our FC for Inner Products mod p

In this section, we show how to extend the transformations of the previous section in order to build FCs for inner products modulo an integer p, starting from an FC for binary inner products. Namely we build FCs for

$$\mathcal{F}_{p,n} = \{f : (\mathbb{Z}_p)^n \to \mathbb{Z}_p\}.$$

Solutions with logarithmic-size openings. For the FC of our first transformation of Section 7.4.1, the adaptation to support the inner product mod p is easy. The only change is to run that construction by setting $\ell = m = \|p\|$ and by letting the second verification check be: $y = \sum_{h=0}^{\ell+m-2} y_h \cdot 2^h \bmod p$. Notice that the FC scheme has exactly the same complexity analysis, considering $\ell = m = \|p\|$.

More in general, given any FC for integer inner products it is possible to construct one for inner products modulo an integer p, at the cost of additionally including $\log(np^2)$ bits in the opening: one simply adds to the opening the result y over the integers, and the verifier additionally checks that $y_p = y \bmod p$.

Solutions with constant-size openings. To build an FC for $\mathcal{F}_{p,n}$ in which openings remain of constant size, we discuss two solutions based on our second transformation of Section 7.4.2.

The first solution is described in Section 7.5.1. It shows how to use an FC for integer inner products to obtain an FC for inner products modulo p, for $p = \mathsf{poly}$, with no overhead in the size of openings. This construction can be instantiated using the FCs obtained with our second transformation of Section 7.4.2. To avoid a quadratic blowup in verification time, this construction can start from FC for integer inner products that enjoy preprocessing-based verification. This way, the verification of the resulting FC remains $O(n)$; as drawback, however the resulting FC does not have preprocessing anymore.

FC *for $\mathbb{Z}$ inner products, with $O(1)$ proofs and preprocessing*	$\Longrightarrow$	FC *for $\mathbb{Z}_p$ inner products, with $O(1)$ proofs (no preprocessing)*

The second solution consists into using the same transformation of Section 7.4.2 with the following differences: set $\ell = m = \|p\|$ and, as a building block, use an FC for binary inner products modulo p, i.e., for computing $\langle \boldsymbol{v}, \boldsymbol{f} \rangle \pmod p$ for $\boldsymbol{v}, \boldsymbol{f} \in \{0,1\}^n$. If such a building block is available and it has constant size proofs, it is easy to see that this variant of the transformation is correct and secure. Clearly, due to the complexity of the transformation we can only use it for small integers $p = O(1), O(\log \lambda)$.

The only missing piece for this construction is showing this building block. In Section 7.5.2 we describe a construction of such a scheme, obtained by tweaking our scheme of section 7.3. This solution preserves preprocessing verification.

FC *for $\mathbb{Z}_2$ inner products mod p, with $O(1)$ proofs*	$\Longrightarrow$	FC *for $\mathbb{Z}_p$ inner products, with $O(1)$ proofs*

7.5.1 Using FC for integer inner products with preprocessing

As explained above, given an FC for integer inner products intFC (like the ones of Section 7.4) one can easily build one for inner products $\pmod p$ by including in the opening proof the

result y of the inner product over the integers, while the actual result is $y_p = y \mod p$. But y (worst-case) can be $n \cdot p^2$. So the size of the proof gets at least $\|y\| < 2\log(p) + \log(n)$.

To remove the logarithmic dependence on n, we observe that having $y_p = y \mod p$ (which the verifier always has as it is an input of the verification algorithm) suffices for the verifier to check the opening. That is, since the euclidean division of y by p is $y = k \cdot p + y_p$ then $0 \leq k \leq np$. Therefore the verifier, after receiving y_p, can brute-force try all the quotients $k \in [0, np]$, set $y(k) = k \cdot p + y_p$ and check if the proof verifies with respect to the integer $y(k)$. If there is no $k \in [0, np]$ such that the proof verifies, it rejects, otherwise she finds the actual y which is accepting.

Naively, this approach would take time $np \cdot t_{\mathsf{Ver}}(n)$, where $t_{\mathsf{Ver}}(n)$ is the verification time of intFC, when checking inner products of length n. The problem with this is that, for $t_{\mathsf{Ver}}(n) = O_\lambda(n)$, the verification becomes $O_\lambda(n^2)$. However, we notice that the verification time of this brute-force search can be kept linear if the scheme intFC one starts from has the preprocessing property. The observation is that all the np verifications are done with respect to the same function. Thus one can first compute vk_f using the preprocessing algorithm, in time $O_\lambda(n)$ ($O_\lambda(n \log \lambda)$ for $p = O(\log \lambda)$), and then run the np verifications, each in fixed n-independent time $O_\lambda(1)$. Hence, using preprocessing we can achieve a verification that is $O_\lambda(pn)$, which is $O_\lambda(n)$ (or $O_\lambda(n \log \lambda)$) for small domains $p = O(1)$ (or $p = O(\log \lambda)$ resp.). As a drawback, one can notice that this brute-force search inherently loses the possibility of achieving preprocessing verification.

We conclude observing that, by considering an instantiation of intFC obtain by applying the transformation of Section 7.4.2 to the FC of Section 7.3 we obtain, for $p = O(1)$ ($p = O(\log \lambda)$), an FC for inner products $\pmod p$ in which openings have fixed size $O_\lambda(1)$ and verification is $O_\lambda(n)$ ($O_\lambda(n \log \lambda)$ resp.).

7.5.2 A variant of our FC for binary inner products mod p

In this section, we present another approach of our FC in 7.3 to treat binary inner products mod p. That is, we let the prover computes g to the power of $L(y), R(y)$ for the verifier. The add-ons to the opening proof is the g to the power of $L(y), R(y)$ accompanied with an argument of knowledge showing that the integer y encoded in these exponentiation is equivalent to y_p modulo p.

Intuition Here we explain the initial idea of our argument of knowledge. For simplicity, we first consider p to be a power of 2, in particular, $p = 2^\ell$. Then we can express $y = q2^\ell + y_p$ with some quotient q and residue y_p. Recall from fig. 7.2 that $L(y) = 2^{\kappa y}$ and $R(y) = 2^{(\kappa + \frac{\kappa}{n})y}$. Here we are ready to describe the hints.

The prover should send $Q_L = g^{2^{\kappa q}}$, $Q_R = g^{2^{(\kappa+\frac{\kappa}{n})q}}$ and also $Q_{L,j} = g^{2^{\kappa q 2^j}}$ and $Q_{R,j} = g^{2^{(\kappa+\frac{\kappa}{n})q2^j}}$ for $j = 1, \ldots, \ell$ to the verifier.

She also uses a Proof of Square Exponent PoSE for each consecutive pair $Q_{L,j}, Q_{L,j+1}$, to show that each pair is of the form g^x, g^{x^2} for the known base g and some x. Similarly, she also needs to prove this relation for each consecutive pair $Q_{R,j}, Q_{R,j+1}$

At the end, the verifier checks the validity of the PoSE's and also checks that $Q_{L,\ell}^{2^{\kappa y_p}} = g^{L(y)}$ and $Q_{R,\ell}^{2^{(\kappa+\frac{\kappa}{n})y_p}} = g^{R(y)}$.

However, the modulo p could be known before $g^{L(y)}$ and $g^{R(y)}$ are fixed. In order to turn this idea into an argument of knowledge, we let the verifer send another prime modulo e as

a challenge. Moreover, to treat generalized modulo p that are not powers of 2, we merge the construction with a square-and-multiply algorithm.

PoKEEM. The proof of knowledge of exponent of exponent modulo p (PoKEEM) is for the following relation

$$\mathcal{L}_{\mathsf{PoKEEM}} = \left\{(Y, a, p, x_p; x) \in \mathbb{G} \times \mathbb{Z} : Y = g^{a^x}, x = x_p \bmod p\right\}$$

parametrized by a group $\mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$ and a group element $g \leftarrow\!\!\$\ \mathbb{G}$. The protocol is in Figure 7.4.

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$, $g \leftarrow\!\!\$\ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: Sends $e \leftarrow\!\!\$\ \mathbb{P}(1, 2^\lambda)$
Denote the binary representation $ep = \overline{b_s \cdots b_1}$ and $e_i = \lfloor e/2^{s-i} \rfloor$;

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Computes $q = \lfloor x/e \rfloor$, $r = x \mod ep$ and sets $Q_i = g^{a^{e_i q}}$ for each $i \in [1, s]$ and $Q'_i = g^{a^{2e_i q}}$ for each $i \in [1, s-1]$. For each $i \in [1, s-1]$, computes $\pi^{(i)}_{\mathsf{PoSE}} = \mathsf{PoSE.P}(\mathsf{crs}, (Q_i, Q'_i), a^{e_i q})$ and $\pi^{(i)}_{\mathsf{PoDDH}} = \mathsf{PoDDH.P}(\mathsf{crs}, (Q_i, Q'_i, Q_{i+1}), (a^{e_i q}, a^{2e_i q}))$ if $b_{i+1} = 1$ or $\pi^{(i)}_{\mathsf{PoDDH}} = (Q_{i+1} = Q'_i)$ if $b_{i+1} = 0$.

Sends $\Lambda := (\boldsymbol{Q}, \boldsymbol{Q'}, \boldsymbol{\pi}, r)$ for $\boldsymbol{Q} = (Q_1, \ldots, Q_s), \boldsymbol{Q'} = (Q'_1, \ldots, Q_{s-1}), \boldsymbol{\pi} = (\pi^{(i)}_{\mathsf{PoSE}})_{i=1}^{s-1}$ and $\boldsymbol{\pi'} = (\pi^{(i)}_{\mathsf{PoDDH}})_{i=1}^{s-1}$.

$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \Lambda)}$: Outputs 1 iff $r \in [ep]$, $x_p = r \bmod p$, ${Q_s}^{a^r} = Y$ and all the proofs $\boldsymbol{\pi}, \boldsymbol{\pi'}$ verify.

Figure 7.4: The succinct argument of knowledge of exponent (PoKEEM) protocol.

Theorem 33. *The protocol* PoKEEM *is an argument of knowledge for the language* $\mathcal{L}_{\mathsf{PoKEEM}}$.

Proof. (Sketch) The proof starts from running the extractor for every proof $\pi^{(i)}_{\mathsf{PoSE}}$ and $\pi^{(i)}_{\mathsf{PoSE}}$ in $\boldsymbol{\pi}, pi'$ to obtain a list of witnesses. These witnesses are the z_i's and z'_i's such that $g^{z_i} = Q_i$ and $g^{z'_i} = Q'_i$. The subsequent argument is to show that these exponent values are consistent (i.e. we should not extract two distinct values for some z_i coming from different proofs). This holds indeed, otherwise we find some low-order relation $g^c = 1$ for c is the difference of the two values. In particular, $z_s = epz_1$.

Now, simplifying the notation $z = z_1$, we want to show that z is of the form a^x for some integer x. Rewinding the protocol for another challenge e', we obtain another response r' and extracted value z' such that $Y = g^{z^{ep} a^r} = g^{z^{e'p} a^{r'}}$. We argue that $z^{ep} a^r = z'^{e'p} a^{r'}$, otherwise we find a low-order relation. Now we represent $z = da^c$ and $z' = d'a^{c'}$ such that $d, d' \nmid a$. Therefore, we have $d^{ep} a^{cep - c'e'p + r - r'} = d'^{e'p}$. W.l.o.g we may assume that $t = cep - c'e'p + r - r'$ is positive, and the relation $d^{ep} a^t = d'^{e'p}$ is over $\mathbb{N}$. By some number-theoretical arguments together with an observation that d, d' are chosen before e, e' are sampled and also $(e, e') = 1$ with overwhelming probability, we derive that $t = 0$ and $d = d' = 1$. In other words, $z = a^c$ and $Y = g^{a^{cep + y_p}}$.

□

Our variant We are ready to describe our FC for binary inner products mod p more precisely. We only describe the differences with the scheme of Section 7.3.

In the Open algorithm, the prover additionally includes the values $g^{L(y)}, g^{R(y)}$ in the opening proof Λ. She also plugs the PoKEEM proof, computed for $a = 2^{\kappa}$ and $a = 2^{(\kappa+\frac{\kappa}{n})}$, in order to prove that the exponents of exponents in $g^{L(y)}$ and $g^{R(y)}$ are congruent to y_p, respectively.

In the Ver algorithm, in order to verify an output y_p, the verifier first runs the verification of Section 7.3 with the difference that it checks the range proof by using the values $g^{L(y)}, g^{R(y)}$ included in Λ. Next, the verifier checks the validity of the PoKEEM proof for elements $g^{L(y)}, g^{R(y)}$ and exponent y_p.

Security We state the theorem below to fulfill the security of our FC variant over $\mathbb{Z}_p$.

Theorem 34. *If protocol* PoKEEM *is an argument of knowledge for* $\mathcal{L}_{\mathsf{PoKEEM}}$ *and our binary functional vector commitment is functional binding, then our variant of binary functional vector commitment over* $\mathbb{Z}_p$ *is functional binding.*

The proof directly comes from the extraction of an integer y congruent to y_p modulo p from $g^{L(y)}, g^{R(y)}$ and an observation that the existence two valid opening modulo p immediately implies the existence of two opening over integer.

Efficiency This modification adds up the opening size for binary inner products an $O_\lambda(\log p + \lambda)$ complexity. It also takes time $O_\lambda(\log p + \lambda) + t_{\mathsf{Ver}}(n)$ to verify, where $t_{\mathsf{Ver}}(n)$ is the verification time of intFC.

Considering an instantiation of intFC deriving from applying the transformation of Section 7.4.2 to the FC of Section 7.3 we then, for $p = O(1)$, obtain an FC for inner products modulo p in which openings are of size $O_\lambda(1)$ and verification is in time $O_\lambda(1)$ with preprocessing.

7.6 Argument of Knowledge Protocols

Protocol PoE: To optimize our constructions' verification time we can utilize the PoE protocol, introduced by Wesolowski [215] for exponents of the form 2^T and generalized by [40] for arbitrary exponents. That is a sound argument for the relation:

$$\mathcal{L}_{\mathsf{PoE}} = \left\{ \left(Y, u, x \in \mathbb{G}^2 \times \mathbb{Z}; \varnothing\right) : Y = u^x \right\}$$

$\mathsf{Setup}(\lambda)$: $\mathbb{G} \leftarrow_{\$} \mathsf{Ggen}(\lambda)$, $g \leftarrow_{\$} \mathbb{G}$, set crs $:= (\mathbb{G}, g)$.

$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: Sends $\ell \leftarrow_{\$} \mathbb{P}(1, 2^\lambda)$

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Computes $q = \lfloor x/\ell \rfloor$ and sets $Q = u^q$. Sends $\pi := Q$.

$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Computes $r = x \mod \ell$ and outputs 1 iff $Q^\ell u^r = Y$.

Figure 7.5: The succinct sound argument (PoE).

PoE is a sound argument system under the adaptive root assumption for Ggen and it is not knowledge sound, since there is no witness. The verifier knows the exponent x. It is used for to improve verification: the verifier performs an $O(\lambda)$ group exponentiation and $O(\|x\|)$ integer

operations, instead of an $O(\|x\|)$ group exponentiation. Although asymptotically this is the same, concretely it gives a significant improvement, since integer operations are much more efficient. Furthermore, its size is 1 group element independently of the size of x.

The proof of knowledge of exponent.

$$\mathcal{L}_{\mathsf{PoKE}} = \left\{(Y, u; x) \in \mathbb{G}^2 \times \mathbb{Z} : Y = u^x\right\}$$

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow_\$ \mathsf{Ggen}(\lambda)$, $g \leftarrow_\$ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{P} \to \mathsf{V}}$: Sends $z = g^x$

$\underline{\mathsf{V} \to \mathsf{P}}$: Sends $\ell \leftarrow_\$ \mathbb{P}(1, 2^\lambda)$

$\underline{\mathsf{P} \to \mathsf{V}}$: Computes $q = \lfloor x/\ell \rfloor$, $r = x \mod \ell$ and sets $Q = u^q, Q' = g^q$. Sends $\pi := (Q, Q', r)$.

$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Outputs 1 iff $r \in [\ell]$, $Q^\ell u^r = Y$ and $Q'^\ell g^r = z$.

Figure 7.6: The succinct argument of knowledge of exponent (PoKE) protocol.

Remark 23. *The above protocol is for arbitrary bases u, adversarially chosen. As noted in [40] the protocol gets simpler in case the base is trusted, generated in the setup phase (i.e. $u = g$ and $Y = g^x$). In particular, z shall not be sent (since it's the statement, $z = Y$) and the proof is 1 group element less.*

The proof of Diffie-Hellman tuple.

$$\mathcal{L}_{\mathsf{PoDDH}} = \left\{(Y_0, Y_1, Y; x_0, x_1) \in \mathbb{G}^3 \times \mathbb{Z}^2 : g_0^{x_0} = Y_0 \wedge g_1^{x_1} = Y_1 \wedge g^{x_0 x_1} = Y\right\}$$

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow_\$ \mathsf{Ggen}(\lambda)$, $g, g_0, g_1 \leftarrow_\$ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{V} \to \mathsf{P}}$: Sends $\ell \leftarrow_\$ \mathbb{P}(1, 2^\lambda)$

$\underline{\mathsf{P} \to \mathsf{V}}$: Computes $q = \lfloor x_0 x_1/\ell \rfloor$, $q_0 = \lfloor x_0/\ell \rfloor$, $q_1 = \lfloor x_1/\ell \rfloor$, $r_0 = x_0 \mod \ell$, $r_1 = x_1 \mod \ell$ and sets $Q = g^q$, $Q_0 = g_0^{q_0}$, $Q_1 = g_1^{q_1}$. Sends $\pi := (Q, Q_0, Q_1, r_0, r_1)$.

$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Computes $r = r_0 r_1 \mod \ell$. Outputs 1 iff $Q^\ell g^r = Y$ and $Q_0^\ell g^{r_0} = Y_0$ and $Q_1^\ell g_1^{r_1} = Y_1$.

Figure 7.7: The succinct argument of knowledge of Diffie-Hellman tuple (PoDDH), under different bases g, g_0, g_1.

The succinct proof of square exponent.

$$\mathcal{L}_{\mathsf{PoSE}} = \left\{(Z_i; x_i) \in \mathbb{G} \times \mathbb{Z} : g^{x_i^2} = Z_i\right\}.$$

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow_{\$} \mathsf{Ggen}(\lambda)$, $g \leftarrow_{\$} \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Sends $z_i = g^{x_i}$

$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: Sends $\ell \leftarrow_{\$} \mathbb{P}(1, 2^{\lambda})$

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Computes $q = \lfloor x_i/\ell \rfloor$, $r = x_i \mod \ell$ and sets $Q = z_i^q, Q' = g^q$. Sends $\pi := (Q, Q', r)$.

$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Outputs 1 iff $r \in [\ell]$, $Q^{\ell} z_i^r = Z_i$ and $Q'^{\ell} g^r = z_i$.

Figure 7.8: The succinct argument of knowledge of square exponent (PoSE).

The succinct range proof protocol

$$\mathcal{L}_{\mathsf{PoRE}} = \left\{ (Y, L, R; x) \in \mathbb{G} \times \mathbb{Z}^3 : L < x < R \wedge g^x = Y \right\}.$$

$\mathsf{Setup}(\lambda)$: $\mathbb{G} \leftarrow_{\$} \mathsf{Ggen}(\lambda)$, $g, g_0, g_1 \leftarrow_{\$} \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g, g_0, g_1)$.

$\mathsf{P} \rightarrow \mathsf{V}$: Computes:

- $F_1' = g_1^{(\mathsf{fprod}_1 - L)(R - \mathsf{fprod}_1)}$
- Run square decomposition algorithm in [180] to find x_1, x_2, x_3 such that $(\mathsf{fprod}_1 - L)(R - \mathsf{fprod}_1) = \sum_{i=1}^{3} x_i^2$.
- For $i = 1, 2, 3$: $Z_{x_i} = g_1^{x_i^2}$, $z_{x_i} = g_1^{x_i}$

Sends $(F_1', \{Z_{x_i}, z_{x_i}\}_{i=1}^{3})$

$\mathsf{V} \rightarrow \mathsf{P}$: Sends $\ell \leftarrow_{\$} \mathbb{P}(1, 2^{\lambda})$

$\mathsf{P} \rightarrow \mathsf{V}$: Computes:

- For $b = 0, 1$: $q_b = \lfloor \mathsf{fprod}_b / \ell \rfloor$, $Q_b = W_b^{q_b}$, $Q_b' = g_b^{q_b}$, $r_b = \mathsf{fprod}_b \mod \ell$.
- $q = \lfloor \mathsf{fprod}_0 \cdot \mathsf{fprod}_1 / \ell \rfloor$, $Q = g^q$, $r = \mathsf{fprod}_0 \cdot \mathsf{fprod}_1 \mod \ell$
- For $i = 1, 2, 3$: $q_{x_i} = \lfloor x_i / \ell \rfloor$, $Q_{x_i} = z_i^{q_{x_i}}$, $Q_{x_i}' = g_i^{q_{x_i}}$, $r_{x_i} \mod \ell$
- $q_L = \lfloor (\mathsf{fprod}_1 - L) / \ell \rfloor$, $Q_L = g_1^{q_L}$, $r_L = (\mathsf{fprod}_1 - L) \mod \ell$
- $q_R = \lfloor (R - \mathsf{fprod}_1) / \ell \rfloor$, $Q_R = g_1^{q_R}$, $r_R = (R - \mathsf{fprod}_1) \mod \ell$
- $q_{LR} = \lfloor (\mathsf{fprod}_1 - L) \cdot (R - \mathsf{fprod}_1) / \ell \rfloor$, $Q_{LR} = g_1^{q_{LR}}$, $r_{LR} = (\mathsf{fprod}_1 - L) \cdot (R - \mathsf{fprod}_1) \mod \ell$

Sends $\pi := \big(\{Q_b, Q_b', r_b\}_{b=0}^{1}, Q, \{Q_{x_i}, Q_{x_i}', r_{x_i}\}_{i=1}^{3}, Q_L, r_L, Q_R, r_R, Q_{LR}\big)$.

$\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)$: Outputs 1 iff:

- $r_0, r_1, r_{x_1}, r_{x_2}, r_{x_3}, r_L, r_R \in [\ell]$
- $Q_b'^{\ell} g_b^{r_b} = F_b$, $Q_b^{\ell} W_b^{r_b} = C_b$, for $b = 0, 1$
- $Q^{\ell} g^{r} = F$
- $Q_{x_i}'^{\ell} g_1^{r_{x_i}} = z_i$, $Q_{x_i}^{\ell} z_{x_i}^{r_{x_i}} = Z_{x_i}$, for $i = 1, 2, 3$
- $Q_L^{\ell} g_1^{r_L} = F_1 \cdot g_1^{-L}$, $Q_R^{\ell} g_1^{r_R} = g_1^{R} \cdot F_1^{-1}$, $Q_{LR}^{\ell} g_1^{r_{LR}} = F_1'$
- $F_1'^4 \cdot g_1 = \prod_{i=1}^{3} Z_i$

where $r = r_0 r_1 \mod \ell$ and $r_{LR} = r_L r_R \mod \ell$.

Figure 7.3: Our merged protocol for the Open algorithm of our binary inner product Functional Commitment (section 7.3.1). The proof size and verification time are independent of the size of fprod and thus n.

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow_{\$} \mathsf{Ggen}(\lambda)$, $g, g_0, g_1 \leftarrow_{\$} \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Computes:

- $Y' = g^{(x-L)(R-x)}$
- For $i = 1, 2, 3$: $Z_{x_i} = g_1^{x_i^2}$, $z_{x_i} = g_1^{x_i}$

Sends $(Y', \{Z_{x_i}, z_{x_i}\}_{i=1}^3)$

$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: Sends $\ell \leftarrow_{\$} \mathbb{P}(1, 2^\lambda)$

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Computes:

- $q = \lfloor x/\ell \rfloor$, $Q_b = g^q$, $r = x \mod \ell$.
- For $i = 1, 2, 3$: $q_{x_i} = \lfloor x_i/\ell \rfloor$, $Q_{x_i} = z_i^{q_{x_i}}$, $Q'_{x_i} = g_i^{q_{x_i}}$, $r_{x_i} \mod \ell$
- $q_L = \lfloor (x-L)/\ell \rfloor$, $Q_L = g_1^{q_L}$, $r_L = (x-L) \mod \ell$
- $q_R = \lfloor (R-x)/\ell \rfloor$, $Q_R = g_1^{q_R}$, $r_R = (R-x) \mod \ell$
- $q_{LR} = \lfloor (x-L)\cdot(R-x)/\ell \rfloor$, $Q_{LR} = g_1^{q_{LR}}$, $r_{LR} = (x-L)\cdot(R-x) \mod \ell$

Sends $\pi := \big(Q, r, \{Q_{x_i}, Q'_{x_i}, r_{x_i}\}_{i=1}^3, Q_L, r_L, Q_R, r_R, Q_{LR}\big)$.

$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Outputs 1 iff:

- $Q^\ell g^r = Y$
- $Q'^\ell_{x_i} g^{r_{x_i}} = z_i$, $Q^\ell_{x_i} z_{x_i}^{r_{x_i}} = Z_{x_i}$, for $i = 1, 2, 3$
- $Q_L^\ell g^{r_L} = Y \cdot g^{-L}$, $Q_R^\ell g^{r_R} = g^R \cdot Y^{-1}$, $Q_{LR}^\ell g^{r_{LR}} = Y'$
- $Y'^4 \cdot g = \prod_{i=1}^3 Z_i$

where $r = r_0 r_1 \mod \ell$ and $r_{LR} = r_L r_R \mod \ell$.

Figure 7.9: The succinct Argument of Knowledge of range of an exponent.

8

KEY-VALUE MAPS FROM ANY VECTOR COMMITMENTS

The results of this chapter appear in a paper under the title "Cuckoo Commitments: Registration-Based Encryption and Key-Value Map Commitments for Large Spaces" that will appear at the ASIACRYPT 2023 conference [104].

8.1 Technical Contributions

In this chapter we present a construction that compiles *any* vector commitment into a key-value map commitment (KVC) [40, 3] for arbitrary-size keys.

The core technique of our approach is a *novel use of cuckoo hashing* [169] in cryptography that can be of independent interest. Cuckoo hashing is a powerful (probabilistic) technique to store elements from a large universe $\mathcal{X}$ into a small table $\boldsymbol{T}$ so that one can later access them in constant-time. Concretely, the latter means that for an element x the cuckoo hash returns $k = O(1)$ possible locations of $\boldsymbol{T}$ where to find x; the cuckoo hashing algorithms take care of resolving collisions by reallocating elements in $\boldsymbol{T}$ whenever a collision occurs.

To put some context, we describe a simple version of Cuckoo Hashing with a stash [140]. For this, we have 2 hash functions h_1, h_2, a table $\boldsymbol{T}$ of size $4n$ and a (unordered) set S, called the 'stash'. To insert a new element x, one first computes $x^{(1)} = h_1(x)$ and if $\boldsymbol{T}[x^{(1)}] = \mathsf{empty}$ then stores x in $\boldsymbol{T}[x^{(1)}]$. Otherwise, if $\boldsymbol{T}[x^{(1)}] = y$ then x 'evicts' y; namely, x is stored in $\boldsymbol{T}[x^{(1)}]$ and y is inserted in $\boldsymbol{T}[y^{(2)}]$ (assume for this example that y was previously 'sent' to $\boldsymbol{T}[x^{(1)}]$ using h_1). Subsequently, if $\boldsymbol{T}[y^{(2)}]$ is occupied by z then y 'evicts' z, and z gets sent to the location specified by the alternative hash function. Observe that one always begin with h_1 for a new element and when the element is evicted always uses the next hash function (and if the last is reached, then the first one again). This procedure continues until either an empty position is found or M attempts have been made. If the latter event occurs, then the last element that was evicted gets stored in the stash S. It can be shown that for random hash functions h_1, h_2, if $M = O(\lambda \log n)$ then the size of the stash is in $O(\log n)$ with overwhelming probability [13].

There are many variants of the above mechanism: Cuckoo Hashing with $k > 2$ hash functions [106] or having tables where every position/bucket has capacity $\ell > 1$ [92]. We refer to [216] for an insightful systematization of knowledge and [219] for an overview from a cryptographic perspective.

In our work, we formally define a *Cuckoo Hashing scheme* in a cryptographic manner in Section 8.2, closely following the definitions of [219]. For the rest, we mostly treat Cuckoo Hashing as a black-box, assuming that it uses k hash functions with buckets of size $\ell = 1$.

Finally, we informally describe how we can use our cuckoo hashing technique in the context of vector commitments, specifically to transform *any* VC scheme into a key-value map commitment for keys from a large space. Assuming one needs to commit to a key-value map consisting of n key-value pairs $(\mathsf{k}_i, \mathsf{v}_i)$ for $i = 1$ to n, one can "cuckoo hash" all the keys so as to obtain a table $\boldsymbol{T}$, a vector, that stores all the keys at certain positions. Then, one can compute a vector commitment $C_{\boldsymbol{T}}$ to $\boldsymbol{T}$ and another vector commitment $C_{\boldsymbol{V}}$ to a vector $\boldsymbol{V}$ built in such a way that $\boldsymbol{V}[j]$ stores a value v if the key k associated to v is stored in $\boldsymbol{T}[j]$. Namely, each pair $(\mathsf{k}_i, \mathsf{v}_i)$ is stored in $\boldsymbol{T}$ and $\boldsymbol{V}$ at the *same* position j. By correctness of cuckoo hashing, for every k such an index j exists. In order to open the commitment $(C_{\boldsymbol{T}}, C_{\boldsymbol{V}})$ to a key k one can use cuckoo hashing to find the set of h candidate indices $(j_1, \ldots, j_h)$ where k is (potentially) stored and open $C_{\boldsymbol{T}}$ at those positions, to find the index j^* such that $\boldsymbol{T}[j^*] = \mathsf{k}$. One then also opens $C_{\boldsymbol{V}}$ to position j^* and its value v. The verifier then would run similarly: for a key-value (k, v), she runs the cuckoo hashing to find out $(j_1, \ldots, j_h)$ associated to k, verify the openings of $C_{\boldsymbol{T}}$ to $(\boldsymbol{T}[j_1], \ldots, \boldsymbol{T}[j_h])$, and the opening of $C_{\boldsymbol{V}}$ to v in the position j^* such that $\boldsymbol{T}[j^*] = \mathsf{k}$. For security it is essential that all $(j_1, \ldots, j_h$ of $(C_{\boldsymbol{T}}, C_{\boldsymbol{V}})$ are opened so that the fact that k is stored in exactly one position can be verified.

In Section 8.3 we give more details on other technicalities of this construction, such as how to: deal with elements in the stash, prove that a key is not committed, reduce key-binding to the position binding of the VC. Notably, this transformation is black-box, *i.e.*, it works by only invoking the algorithms of the underlying VC. This stands in contrast to, *e.g.*, Verkle tree approaches [67, 142].

8.2 Cuckoo Hashing Schemes

Cuckoo Hashing (CH) [169] is a technique to store a set of m elements from a large universe $\mathcal{X}$ into a linear-size data structure that allows efficient memory accesses. In our work we abstract away the properties of a family of cuckoo hashing constructions that can be used in our RBE and KVC constructions. We do this by defining the notion of *Cuckoo Hashing schemes*. Our definition is a variant of the one recently offered by Yeo [219]; in our definition, we use *deterministic* Insert algorithms.

In a nutshell, a cuckoo hashing scheme inserts n elements $x_1, \ldots, x_n \in \mathcal{X}$ in a vector $\boldsymbol{T}$ so that each element x_i can be found exactly once in $\boldsymbol{T}$, or in a stash set S. The efficient memory access comes from the fact that for a given x one can efficiently compute the k indices $i_1, \ldots, i_k$ such that $x \in \{\boldsymbol{T}[i_1], \ldots, \boldsymbol{T}[i_k]\} \cup S$. The idea of cuckoo hashing constructions is to sample k random hash functions $H_1, \ldots, H_k : \mathcal{X} \to [n]$ and use them to allocate x in one of the k indices $H_1(x), \ldots, H_k(x)$. Each construction uses a specific algorithm to search the index allocated to x, requiring to move existing elements whenever a position is going to be allocated to another element. The most efficient algorithms are local search allocation [139] and random walks [110, 107, 109, 212, 219].

We define a Cuckoo Hashing scheme with the following algorithms:

Definition 36 (Cuckoo Hashing Schemes Algorithms)**.** *A Cuckoo Hashing scheme* $\mathsf{CH} = (\mathsf{Setup}, \mathsf{Insert}, \mathsf{Lookup})$ *consists of the following algorithms:*

- $\mathsf{Setup}(1^\lambda, \mathcal{X}, n) \rightarrow (\mathsf{pp}, \boldsymbol{T}, S)$: *is a probabilistic algorithm that on input the security parameter, the space of input values $\mathcal{X}$ and a bound n on the number of insertions, outputs public parameters* pp*, $k \geq 2$, an empty vector $\boldsymbol{T}$ with N entries (with N a multiple of k), along with an empty stash set S, (denoting $s \geq 0$ its size, at this point, $s = 0$);*
- $\mathsf{Insert}(\mathsf{pp}, \boldsymbol{T}, S, x_1, \ldots, x_m) \rightarrow (\boldsymbol{T}', S')$: *is a deterministic algorithm that on input vector $\boldsymbol{T}$ where each non-empty component contains an element in $\mathcal{X} \in \mathsf{pp}$, inserts each $x_1, \ldots, x_m \in \mathcal{X}$ in the vector exactly once and returns the updated vector with moved elements, $\boldsymbol{T}', S'$.*
- $\mathsf{Lookup}(\mathsf{pp}, x) \rightarrow (i_1, \ldots, i_k)$: *is a deterministic algorithm that on input public parameters* pp *and $x \in \mathcal{X}$, returns $(i_1, \ldots, i_k)$, the candidate indices where x could be stored.*

Remark 24. *Our Cuckoo Hashing schemes are, overall, probabilistic with the probability taken over the choice of* pp*. Once* pp *is fixed, everything is deterministic;* Insert *and* Lookup*, that take* pp *as input, are deterministic algorithms.*

Our definition above differs from the one in [219] in the following aspects. First, we consider dynamic cuckoo hashing schemes in which one can keep inserting elements, while [219] considers the static case in which the set is hashed all at once. Second, in our notion each entry of $\boldsymbol{T}$ can store a single element, whereas [219] considers the more general case where it can store $\ell \geq 1$ elements, which occurs in some constructions.

We define correctness of cuckoo hashing by looking at the probability that either the insertion algorithm fails or, if it does not fail, an inserted element is not stored in the appropriate indices returned by Lookup. To model this notion we give two definitions. The first one is the "classical" correctness definition of cuckoo hashing that takes this probability over any choice of inputs but for a random and independent sampling of the hash functions. Intuitively this models the scenario where an adversary for correctness does not have explicit access to the hash functions, but can still choose any input set.

Definition 37 (Correctness)**.** *A cuckoo hashing scheme* CH *is ϵ-**correct** if for any n, any set of $m \leq n$ items $x_1, \ldots, x_m \in \mathcal{X}$ such that $x_i \neq x_j$ for all $i \neq j$ and any $\ell \in [m]$:*

$$\Pr\left[\begin{array}{c} \boldsymbol{T}' = \bot \\ \vee\ (\boldsymbol{T}' \neq \bot \wedge \\ x_\ell \notin \{\boldsymbol{T}'[i_1], \ldots, \boldsymbol{T}'[i_k]\} \cup S') \end{array} : \begin{array}{r} (\mathsf{pp}, \boldsymbol{T}, S) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{X}, n) \\ (\boldsymbol{T}', S') \leftarrow \mathsf{Insert}(\mathsf{pp}, \boldsymbol{T}, S, x_1, \ldots, x_m) \\ (i_1, \ldots, i_k) \leftarrow \mathsf{Lookup}(\mathsf{pp}, x_\ell) \end{array}\right] \leq \epsilon$$

and one simply says that CH *is **correct** if it is ϵ-correct with $\epsilon = \mathsf{negl}(\lambda)$.*

8.2.1 Robust Cuckoo Hashing

The second definition (introduced by Yeo [219]) instead considers the case of inputs that are chosen by a PPT adversary after having seen the hash functions. This models the scenario where an adversary has explicit access to the hash functions before choosing the set of elements.

Definition 38 (Robustness)**.** *A cuckoo hashing scheme* CH *is ϵ-**robust** if for any n, any PPT adversary $\mathcal{A}$:*

$$\Pr\left[\begin{array}{c} \boldsymbol{T}' = \bot \\ \vee\ (\boldsymbol{T}' \neq \bot \wedge \\ x_\ell \notin \{\boldsymbol{T}'[i_1], \ldots, \boldsymbol{T}'[i_k]\} \cup S') \end{array} : \begin{array}{r} (\mathsf{pp}, \boldsymbol{T}, S) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{X}, n) \\ \{x_1, \ldots, x_m, \ell\} \leftarrow \mathcal{A}(\mathsf{pp}) \\ x_i \neq x_j \forall i \neq j \in [m] \\ (\boldsymbol{T}', S') \leftarrow \mathsf{Insert}(\mathsf{pp}, \boldsymbol{T}, S, x_1, \ldots, x_m) \\ (i_1, \ldots, i_k) \leftarrow \mathsf{Lookup}(\mathsf{pp}, x_\ell) \end{array}\right] \leq \epsilon$$

8.2.2 Efficiency parameters of cuckoo hashing

For our applications, the following parameters will dictate the efficiency of a cuckoo hashing scheme: k, the number of possible indices (and of hash functions); N, the size of the table $\boldsymbol{T}$; s, the size of the stash S; d, the number of changes in the table (i.e., number of evictions) after a single insertion. While in most constructions, the parameters k and N are fixed at Setup time, in some cuckoo hashing schemes the values of s and d may depend on the randomness and the choice of inputs. As in the case of correctness vs. robustness, we define s and d in the average case (i.e., for any set of inputs and for random and independent execution of Setup) or in the worst case (i..e, for adversarial choice of inputs after seeing pp).

8.2.3 Existing cuckoo hashing schemes

The following theorem encompasses a few existing cuckoo hashing schemes.

Theorem 35. *For a security parameter λ and an upper bound n, there exist the following cuckoo hashing schemes:*

- CH_2 *where $k = 2$, $N = 2kn$, that achieves $\mathsf{negl}(\lambda)$-correctness, and average case $s = \log n$, $d = O(1)$ [140].*
- $\mathsf{CH}_2^{(\mathsf{rob})}$ *where $k = 2$, $N = 2kn$, that achieves $\mathsf{negl}(\lambda)$-robustness, and worst case $s = n$, $d = O(1)$ [140, 219] in the Random Oracle Model.*
- $\mathsf{CH}_\lambda^{(\mathsf{rob})}$ *where $k = \lambda$, $N = 2\lambda n$, that achieves $\mathsf{negl}(\lambda)$-robustness, and worst case $s = 0$, $d = \lambda$ [219] in the Random Oracle Model.*

8.3 Key-Value Map Commitments from Cuckoo Hashing and Vector Commitments

Given a key space $\mathcal{K}$ and a value space $\mathcal{V}$, a key-value map $\mathcal{M} \subseteq \mathcal{K} \times \mathcal{V}$ is a collection of pairs $(\mathsf{k}, \mathsf{v}) \in \mathcal{K} \times \mathcal{V}$. Key-value map commitments (KVC) [40, 3] are a cryptographic primitive that allows one to commit to a key-value map $\mathcal{M}$ in such a way that one can later open the commitment at a specific key, i.e., prove that (k, v) is in the committed map $\mathcal{M}$, and do so in a key-binding way. Namely, it is not possible to open the commitment at two distinct values $\mathsf{v} \neq \mathsf{v}'$ for the same key k. KVCs are a generalization of vector commitments [66]: while in VCs the key space is the set of integers $\{1, \ldots, n\}$, in a KVC the key space is usually a set of exponential size.

In this section, we present a construction of KVCs based on a combination of vector commitments and cuckoo hashing. The resulting KVC needs to fix at setup time a bound on the cardinality of the key-value map, but otherwise supports a key space and a value space of arbitrary sizes.

8.3.1 KVM Construction from Cuckoo Hashing and Vector Commitments

We present a construction of a KVC for keys of arbitrary size. Our scheme is obtained by combining a Cuckoo Hashing scheme CH and a Vector Vommitment one VC. We refer to Section 8.1 for an intuitive description.

- $\mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V}) \to \mathsf{crs}$: runs $(\mathsf{pp}_{\mathsf{CH}}, \hat{\boldsymbol{T}}, \hat{S}) \leftarrow \mathsf{CH.Setup}(1^\lambda, n)$, and generates $\mathsf{crs}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, N)$, then returns $\mathsf{crs} \leftarrow (\mathsf{crs}_{\mathsf{VC}}, \mathsf{pp}_{\mathsf{CH}})$.
- $\mathsf{Com}(\mathsf{crs}, \mathcal{M}) \to (C, \mathsf{aux})$: on input a key-value map $\mathcal{M} = \{(\mathsf{k}_i, \mathsf{v}_i)\}_{i=1}^m$:

  1. if there exists $i, j \in [m], i \neq j$ such that $\mathsf{k}_i = \mathsf{k}_j$, it aborts;
  2. $(\boldsymbol{T}, S) \leftarrow \mathsf{CH.Insert}(\mathsf{pp}_{\mathsf{CH}}, \hat{\boldsymbol{T}}, \hat{S}, \mathsf{k}_1, \ldots, \mathsf{k}_m)$; if $\boldsymbol{T} = \perp$ it aborts, else sets $\boldsymbol{T}' \leftarrow \mathbf{cat}(\boldsymbol{T})$;
  3. $(C_{\boldsymbol{T}}, \mathsf{aux}_{\boldsymbol{T}}) \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, \boldsymbol{T}')$;
  4. For $j = 1$ to m, let $\mathsf{ind}_j \in [N]$ be the index such that $\boldsymbol{T}'[\mathsf{ind}_j] = \mathsf{k}_j$. If ind_j exists, it sets $\boldsymbol{V}[\mathsf{ind}_j] \leftarrow \mathsf{v}_j$, otherwise, if $\mathsf{k}_j \in S$, adds $(\mathsf{k}_j, \mathsf{v}_j)$ to S^*.
  5. $(C_{\boldsymbol{V}}, \mathsf{aux}_{\boldsymbol{V}}) \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, \boldsymbol{V})$

  It return $C = (C_{\boldsymbol{T}}, C_{\boldsymbol{V}}, S^*)$ and $\mathsf{aux} = (\mathsf{aux}_{\boldsymbol{T}}, \mathsf{aux}_{\boldsymbol{V}}, S^*, \boldsymbol{T}', S, \mathcal{M})$.
- $\mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k}) \to \Lambda$:

  1. $(\mathsf{ind}_1, \ldots, \mathsf{ind}_k) \leftarrow \mathsf{CH.Lookup}(\mathsf{pp}_{\mathsf{CH}}, \mathsf{k})$;
  2. if $\mathsf{k} \notin \{\boldsymbol{T}[\mathsf{ind}_1], \ldots, \boldsymbol{T}[\mathsf{ind}_k]\} \cup S$ aborts;
  3. for $j = 1$ to k: $\Lambda_j \leftarrow \mathsf{VC.Open}(\mathsf{crs}_{\mathsf{CH}}, \mathsf{aux}_{\boldsymbol{T}}, \mathsf{ind}_j)$.
  4. Let $\mathsf{ind}^* \in [N]$ be the index such that $\boldsymbol{T}[\mathsf{ind}^*] = \mathsf{k}$. If ind^* exists, it computes $\Lambda^* \leftarrow \mathsf{VC.Open}(\mathsf{crs}_{\mathsf{CH}}, \mathsf{aux}_{\boldsymbol{V}}, \mathsf{ind}^*)$, else sets $\Lambda^* = \perp$.

  Return $\Lambda = (\Lambda_1, \boldsymbol{T}[\mathsf{ind}_1], \ldots, \Lambda_k, \boldsymbol{T}[\mathsf{ind}_k], \Lambda^*)$.
- $\mathsf{Ver}(\mathsf{crs}, C, \Lambda, (\mathsf{k}, \mathsf{v})) \to b$: parses $C = (C_{\boldsymbol{T}}, C_{\boldsymbol{V}}, S^*)$ and $\Lambda = (\Lambda_1, t_1, \ldots, \Lambda_k, t_k, \Lambda^*)$ and proceeds as follows:

  1. $(\mathsf{ind}_1, \ldots, \mathsf{ind}_k) \leftarrow \mathsf{CH.Lookup}(\mathsf{pp}_{\mathsf{CH}}, \mathsf{k})$;
  2. for $j = 1$ to k: $b_j \leftarrow \mathsf{VC.Ver}(\mathsf{crs}_{\mathsf{CH}}, C_{\boldsymbol{T}}, \Lambda_j, \mathsf{ind}_j, t_j)$; if $\bigwedge_{j=1}^k b_j = 0$ outputs 0, else continues;
  3. if $\mathsf{k} \notin \{t_1, \ldots, t_k\}$, outputs 1 if and S^* is valid (i.e., it does not contain any repeated key and no entry (k, ϵ) and $(\mathsf{k}, \mathsf{v}) \in S^*$, else outputs 0.
  4. Otherwise, let j^* be the first index such that $\mathsf{k} = t_{j^*}$: it computes $b^* \leftarrow \mathsf{VC.Ver}(\mathsf{crs}_{\mathsf{CH}}, C_{\boldsymbol{V}}, \Lambda^*, \mathsf{ind}_{j^*}, \mathsf{v})$, and returns b^*.

Correctness and succinctness One can see by inspection that the proposed KVC scheme is robust (with overwhelming probability) under the assumption that VC is perfectly correct and that the cuckoo hashing scheme CH is robust. Succinctness of our KVC scheme is also easy to see by inspection, under the assumption that VC is succinct and that we use an instantiation of CH that satisfies $k = O(\log n)$.

Theorem 36 (Key binding)**.** *If* VC *is position binding, then* KVM *is a key-binding KVC.*

Proof. Assume by contradiction that there exists a PPT adversary $\mathcal{A}$ that breaks the position binding of our KVC scheme. Then we show how to build a reduction $\mathcal{B}$ that breaks the position binding of VC. $\mathcal{B}$ takes as input $\mathsf{crs}_{\mathsf{VC}}$, generates the CH public parameters and runs $\mathcal{A}$ on input $\mathsf{crs} \leftarrow (\mathsf{crs}_{\mathsf{VC}}, \mathsf{pp}_{\mathsf{CH}})$.

Assume that $\mathcal{A}$ returns a tuple $(C, \mathsf{k}, \mathsf{v}, \Lambda, \tilde{\mathsf{v}}, \tilde{\Lambda})$ that breaks key binding with non-negligible probability. Let

$$\Lambda = (\Lambda_1, t_1, \ldots, \Lambda_k, t_k, \Lambda^*), \quad \tilde{\Lambda} = (\tilde{\Lambda}_1, \tilde{t}_1, \ldots, \tilde{\Lambda}_k, \tilde{t}_k, \tilde{\Lambda}^*)$$

By the winning condition of key binding we have that $\mathsf{v} \neq \tilde{\mathsf{v}}$ and that both opening proofs are accepted by the Ver algorithm. In particular, since Ver is invoked on the same key k, the first step of verification computes the same indices $\mathsf{ind}_1, \ldots, \mathsf{ind}_k$ in the verification of both Λ and $\tilde{\Lambda}$.

First, notice that it must be the case that $\forall j \in [k]$, $t_j = \tilde{t}_j$. Otherwise, one can immediately break the VC position binding with the tuple $(C_{\boldsymbol{T}}, \mathsf{ind}_j, \Lambda_j, t_j, \tilde{\Lambda}_j, \tilde{t}_j)$.

Second, if $\mathsf{k} \notin \{t_1, \ldots, t_k\}$ then by construction of Ver (step 3), $\mathcal{A}$ cannot break position binding.

Finally, let j^* be the index such that $\mathsf{k} = t_{j^*}$. Then one can break the VC position binding with the tuple $(C_{\boldsymbol{V}}, \mathsf{ind}_{j^*}, \Lambda^*, \mathsf{v}, \tilde{\Lambda}^*, \tilde{\mathsf{v}})$. □

In Section 8.4.2 we show that this KVC is updatable.

8.3.2 Key-Value Map Instantiations

We can instantiate the generic construction of the previous section with the CH scheme CH_λ from Theorem 35 (which is robust in the random oracle model) and any of the existing vector commitment schemes. If the VC has constant-size openings, say $O(\lambda)$, then the resulting KVC constructions have openings of size $O(\lambda^2)$. The most interesting implication of this KVC instantiation is that we obtain the first KVCs for unbounded key space based on pairings in a black-box manner. More in detail, we can obtain a variety of updatable KVCs according to which updatable VC we start from, e.g., we can use [66] to obtain a KVC based on CDH, [150] for one based on q-DHE. Prior to this work, an updatable KVC under these assumptions could only be obtained by instantiating the Merkle tree scheme with one of [150, 66] VCs. However, Merkle trees with algebraic VCs need to make a non-black-box use of the underlying groups in order to map commitments back to the message space. In contrast, all our KVCs are black-box, if so are the underlying VC (as it is the case for virtually all existing schemes).

8.3.3 Accumulators from Vector Commitments with Cuckoo Hashing

It is easy to see that a KVC for a key space $\mathcal{K}$ immediately implies a universal accumulator [18, 147] for universe $\mathcal{K}$. The idea is simple: to accumulate $\mathsf{k}_1, \ldots, \mathsf{k}_n$ one commits to the key-value map $\{(\mathsf{k}_1, 1), \ldots, (\mathsf{k}_n, 1)\}$; a membership proof for k is an opening to $(\mathsf{k}, 1)$, and a non-membership proof is a KVC opening to (k, ϵ). The security of this construction (i.e., undeniability [152] – the hardness of finding a membership and a non-membership proof for the same element) follows straightforwardly from key binding. Furthermore, if the KVC is updatable, the accumulator is updatable (aka dynamic, in accumulators lingo).

From this, we obtain new accumulators for large universe enjoying properties not known in prior work. For instance, we obtain the first dynamic accumulators for a large universe that are based on pairings in a black-box manner. To the best of our knowledge, prior black-box pairing-based accumulators either support a small universe [56, 66], or are not dynamic [167, 137]. Notably, using the CDH-based VC of [66] we obtain the first universal accumulator for a large universe that is dynamic, based on the CDH problem over bilinear groups, and black-box.

8.4 Updatable Key-Value Map Commitments

8.4.1 Definitions

We define *updatable* key-value map commitments as an extension of KVCs in which, akin to updatable VCs, one can efficiently update the commitment and the openings with respect to changes in the committed key-value map.

We model an update in a key-value map $\mathcal{M}$ as a pair (k, δ) where $\mathsf{k} \in \mathcal{K}$ is a key and δ is an update information which can be:

- $\delta = (\mathsf{insert}, \mathsf{v})$ to denote the insertion of the pair (k, v), i.e., $\mathcal{M}' \leftarrow \mathcal{M} \cup (\mathsf{k}, \mathsf{v})$;
- $\delta = (\mathsf{delete}, \mathsf{v})$ to denote the deletion of the pair (k, v), i.e., $\mathcal{M}' \leftarrow \mathcal{M} \setminus (\mathsf{k}, \mathsf{v})$;
- $\delta = (\mathsf{update}, \mathsf{v}, \mathsf{v}')$ to denote the change of value from v to v' associated to the key k, i.e., $\mathcal{M}' \leftarrow (\mathcal{M} \setminus (\mathsf{k}, \mathsf{v})) \cup (\mathsf{k}, \mathsf{v}')$.

Also, we say that (k, δ) is valid for $\mathcal{M}$ if the operation is well defined, namely: it inserts a key that is not present in the map, it deletes or changes the value of an already existing key.

Definition 39 (Updatable KVC)**.** *A key-value map has* updatable commitments *if there exist two additional algorithms* ComUpdate *and* ProofUpdate *that work as follows.*

- $\mathsf{ComUpdate}(\mathsf{crs}, C, (\mathsf{k}, \delta), \mathsf{aux}) \rightarrow (C', \mathsf{aux}', \pi_\delta)$*: Given a commitment C, key k, update information δ and an auxiliary information* aux*, the algorithm outputs a new commitment C', auxiliary information* aux$'$*, and update hint π_δ.*
- $\mathsf{ProofUpdate}(\mathsf{crs}, \Lambda_\mathsf{k}, (\hat{\mathsf{k}}, \delta), \pi_\delta) \rightarrow \Lambda'_\mathsf{k}$*): Given an opening Λ_k for some key $\mathsf{k} \in \mathcal{K}$, a key $\hat{\mathsf{k}}$ (possibly different from k), update information δ associated to $\hat{\mathsf{k}}$, and an update hint π_δ, the algorithm outputs a new opening Λ'_k.*

Robust Correctness of Updatable KVCs. *An updatable* KVM *is* robust *if for any public parameters* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V})$*, any adversarial choice of a key-value map $\mathcal{M} \subseteq \mathcal{K} \times \mathcal{V}$ of size $\leq n$, $\mathsf{v} \in (\mathcal{V} \cup \{\epsilon\})$, $(\mathsf{k}, \mathsf{v}') \in \mathcal{K} \times (\mathcal{V} \cup \{\epsilon\})$, and sequence of valid updates $\{(\hat{\mathsf{k}}_j, \delta_j)\}_{j \in [m]}$ that eventually yields a $\mathcal{M}'$ of size $\leq n$ such that $(\mathsf{k}, \mathsf{v}') \in \mathcal{M}'$, any initial commitment $(C_0, \mathsf{aux}_0) \leftarrow \mathsf{Com}(\mathsf{crs}, \mathcal{M})$ and opening $\Lambda^{(0)}_\mathsf{k} \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}_0, \mathsf{k})$, and, sequentially, for $j \in [m]$, updated commitments and openings $(C_j, \mathsf{aux}_j, \pi_{\delta j}) \leftarrow \mathsf{ComUpdate}(\mathsf{crs}, C_{j-1}, (\hat{\mathsf{k}}_j, \delta_j), \mathsf{aux}_{j-1})$ and $\Lambda^{(j)}_\mathsf{k} \leftarrow \mathsf{ProofUpdate}(\mathsf{crs}, \Lambda^{(j-1)}_\mathsf{k}, (\hat{\mathsf{k}}_j, \delta_j), \pi_{\delta j})$, we have that $\mathsf{Ver}(\mathsf{crs}, C_m, \Lambda^{(m)}_\mathsf{k}, (\mathsf{k}, \mathsf{v}')) = 1$ holds with overwhelming probability in λ.*

Efficiency of Updatable KVCs. *An updatable KVC is said efficient if:*

(i) The efficiency definition of Definition 17 holds also for commitments and openings produced by ComUpdate *and* ProofUpdate *respectively. Additionally, the update hints π_δ produced by* ComUpdate *should also have polylogarithmic size.*

(ii) The runtimes of ComUpdate *and* ProofUpdate *are polylogarithmic in n.*

Our updateability notion for KVCs corresponds to what is known as *stateful updates* in the VC literature. This is due to the fact that ComUpdate requires knowledge of the auxiliary information aux related to the commitment C (which possibly contains the committed vector), and produces an update hint π_δ that can be used by anyone to update local proofs. This model is useful in applications where a central party can perform an update and enables everyone else to update proofs by publishing succinct information.

8.4.2 An Updatable Key-Value Map Construction from Cuckoo Hashing and Vector Commitments

Here we show that the scheme described above is also updatable, defining the following algorithms:

- $\mathsf{ComUpdate}(\mathsf{crs}, C, (\mathsf{k}, \delta), \mathsf{aux}) \rightarrow (C', \mathsf{aux}', \pi_\delta)$: this algorithm initializes the following vectors: $\hat{\boldsymbol{T}}' \leftarrow \hat{\boldsymbol{T}}, \hat{\boldsymbol{V}}' \leftarrow \hat{\boldsymbol{V}}$, and computes:

  1. $(\mathsf{ind}_1, \ldots, \mathsf{ind}_k) \leftarrow \mathsf{CH.Lookup}(\mathsf{pp}_{\mathsf{CH}}, \mathsf{k})$.
  2. if $\delta[1] \in \{\mathsf{delete}, \mathsf{update}\}$, then:
     (a) if looks for $\mathsf{ind} \in \{\mathsf{ind}_1, \ldots, \mathsf{ind}_k\}$ such that $\hat{\boldsymbol{T}}[\mathsf{ind}] = \mathsf{k}$. If there are several, it aborts, if there is none, then it looks for an element in $\hat{S}$ with first component k: if there are several or none it aborts; if there is one such element but its second component is not $\mathsf{v} = \delta[2]$, then it also aborts;
     (b) if $\delta[1] = \mathsf{delete}$, it removes the value in $\hat{\boldsymbol{T}}'[\mathsf{ind}]$ if ind existed, and else it removes (k, v) from $\hat{S}$;
     (c) else if $\delta[1] = \mathsf{update}$, it sets $\hat{\boldsymbol{V}}'[\mathsf{ind}] \leftarrow \mathsf{v}'$ (where $\mathsf{v}' = \delta[3]$) if ind existed, and else it removes (k, v) from S^* and adds $(\mathsf{k}, \mathsf{v}')$ in S^*;
  3. if $\delta[1] = \mathsf{insert}$, then:
     (a) $(\boldsymbol{T}, S) \leftarrow \mathsf{CH.Insert}(\mathsf{pp}_{\mathsf{CH}}, \hat{\boldsymbol{T}}, \hat{S}, \mathsf{k})$. If $\boldsymbol{T} = \perp$, it aborts. Else it updates $\hat{\boldsymbol{T}}' \leftarrow \mathbf{cat}(\boldsymbol{T})$;
     (b) it finds $\mathsf{ind} \in \{\mathsf{ind}_1, \ldots, \mathsf{ind}_k\}$ such that $\hat{\boldsymbol{T}}'[\mathsf{ind}] = \mathsf{k}$. If there are several, it aborts, if there is none, if looks for k in S; if $\mathsf{k} \notin S$, then it aborts;
     (c) if ind existed, it sets $\hat{\boldsymbol{V}}'[\mathsf{ind}] \leftarrow \mathsf{v}$ (where $\mathsf{v} = \delta[2]$), else it adds (k, v) to S^*.
  4. $\pi_{\boldsymbol{T}}$ is initialized as $\emptyset$. For each index i such that $\hat{\boldsymbol{T}}'[i] \neq \hat{\boldsymbol{T}}[i]$, it sequentially updates $(C_{\boldsymbol{T}}, \mathsf{aux}_{\boldsymbol{T}}) \leftarrow \mathsf{VC.ComUpdate}(\mathsf{crs}_{\mathsf{VC}}, C_{\boldsymbol{T}}, i, \hat{\boldsymbol{T}}[i], \hat{\boldsymbol{T}}'[i])$, and adds $(i, \hat{\boldsymbol{T}}[i], \hat{\boldsymbol{T}}'[i])$ to $\pi_{\boldsymbol{T}}$.
  5. it initializes $\pi_{\boldsymbol{V}} \leftarrow \emptyset$. For each index i such that $\hat{\boldsymbol{V}}'[i] \neq \hat{\boldsymbol{V}}[i]$, it sequentially updates $(C_{\boldsymbol{V}}, \mathsf{aux}_{\boldsymbol{V}}) \leftarrow \mathsf{VC.ComUpdate}(\mathsf{crs}_{\mathsf{VC}}, C_{\boldsymbol{V}}, i, \hat{\boldsymbol{V}}[i], \hat{\boldsymbol{V}}'[i])$, and adds $(i, \hat{\boldsymbol{V}}[i], \hat{\boldsymbol{V}}'[i])$ to $\pi_{\boldsymbol{V}}$.
  6. finally it returns: $C' \leftarrow (C_{\boldsymbol{T}}, C_{\boldsymbol{V}}, S^*)$, $\mathsf{aux}' \leftarrow (\mathsf{aux}_{\boldsymbol{T}}, \mathsf{aux}_{\boldsymbol{V}}, S^*, \hat{\boldsymbol{T}}, S, \mathcal{M})$, and $\pi_\delta \leftarrow (\pi_{\boldsymbol{T}}, \pi_{\boldsymbol{V}}, \hat{\boldsymbol{T}}[\mathsf{ind}_1], \ldots, \hat{\boldsymbol{T}}[\mathsf{ind}_k], \hat{S})$.

- $\mathsf{ProofUpdate}(\mathsf{crs}, \Lambda_{\mathsf{k}} = (\Lambda_1, t_1, \ldots, \Lambda_k, t_k, \Lambda^*), (\hat{\mathsf{k}}, \delta), \pi_\delta = (\pi_{\boldsymbol{T}}, \pi_{\boldsymbol{V}}, t'_1, \ldots, t'_k, S)) \rightarrow \Lambda'_{\mathsf{k}}$: this algorithm computes:

  1. $(\mathsf{ind}_1, \ldots, \mathsf{ind}_k) \leftarrow \mathsf{CH.Lookup}(\mathsf{pp}_{\mathsf{CH}}, \hat{\mathsf{k}})$.
  2. if $\hat{\mathsf{k}} \notin \{t'_1, \ldots, t'_k\} \cup S$, then it aborts;
  3. for each $j \in [k]$, for each $(i, t, t') \in \pi_{\boldsymbol{T}}$, it updates: $\Lambda_j \leftarrow \mathsf{VC.ProofUpdate}(\mathsf{crs}_{\mathsf{VC}}, \Lambda_j, i, t, t')$;
  4. it looks for $i^* \in [k]$ such that $t'_{i^*} = \hat{\mathsf{k}}$. If there are several such indices, it aborts. Else if i^* does not exist, it updates $\Lambda^* \leftarrow \perp$. Else if i^* exists and is unique, for each $(i, v, v') \in \pi_{\boldsymbol{V}}$, it sequentially updates $\Lambda^* \leftarrow \mathsf{VC.ProofUpdate}(\mathsf{crs}_{\mathsf{VC}}, \Lambda^*, i, v, v')$.
  5. finally it returns $\Lambda'_{\mathsf{k}} = (\Lambda_1, t'_1, \ldots, \Lambda_k, t'_k, \Lambda^*)$.

One can remark that in the ProofUpdate algorithm, if the input $\hat{\mathsf{k}}$ is equal to the k of the input Λ_{k}, then $t'_1, \ldots, t'_k$ will not be required in δ as they will be equal to the $t_1, \ldots, t_k$ in Λ_{k}.

Robust Correctness. We show robust correctness of the above updatable KVC, $\mathsf{KVM} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver}, \mathsf{ComUpdate}, \mathsf{ProofUpdate})$, built from the Cuckoo Hashing scheme CH and the Vector Commitment scheme VC, if CH is robust and VC is correct.

Let n be an integer, λ a security parameter, $\mathcal{K}$ a key space and $\mathcal{V}$ a value space which are subspaces of the input space of VC's vectors, and: $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V})$. Let $\mathcal{A}$ be a PPT adversary, who chooses a key-value map $\mathcal{M} \subset \mathcal{K} \times \mathcal{V}$ of size n or less, $(\mathsf{k}, \mathsf{v}) \in \mathcal{K} \times \mathcal{V} \cup \{\epsilon\}$, $\mathsf{v}' \in \mathcal{V} \times \{\epsilon\}$, and a sequence of valid updates $\{(\hat{\mathsf{k}}_j, \delta_j)\}_{j \in [m]}$ that eventually yields the udpated key-value map $\mathcal{M}'$ of size n or less such that $(\mathsf{k}, \mathsf{v}') \in \mathcal{M}'$.

Let $(C_0, \mathsf{aux}_0) \leftarrow \mathsf{Com}(\mathsf{crs}, \mathcal{M})$, $\Lambda_k^{(0)} \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k})$, $(C_j, \mathsf{aux}_j, \pi_{\delta j}) \leftarrow \mathsf{ComUpdate}(\mathsf{crs}, C_{j-1}, (\hat{\mathsf{k}}_j, \delta_j), \mathsf{aux}_{j-1})$ and $\Lambda_{\mathsf{k}}^{(j)} \leftarrow \mathsf{ProofUpdate}(\mathsf{crs}, \Lambda_{\mathsf{k}}^{(j-1)}, (\hat{\mathsf{k}}_j, \delta_j), \pi_{\delta j})$.

We write this demonstration by induction. If there are no updates ($m = 0$), then $\mathcal{M}' = \mathcal{M}$, and since we required uniqueness of the keys in our key-value map construction, $\mathsf{v}' = \mathsf{v}$ is the value associated to k and $\mathsf{Ver}(\mathsf{crs}, C_0, \Lambda_{\mathsf{k}}^{(0)}, (\mathsf{k}, \mathsf{v})) = 1$ from the correctness of VC. Now, for the end of the induction, let us suppose that for some $i \in [0; m-1]$, and for $(\mathsf{k}, \mathsf{v}_i)$ the key-value pair of k in the $\mathcal{M}$ after update i, $\mathsf{Ver}(\mathsf{crs}, C_i, \Lambda_{\mathsf{k}}^{(i)}, (\mathsf{k}, \mathsf{v}_i)) = 1$.

Let $(\mathsf{k}, \mathsf{v}_{i+1})$ be the key-value pair for k in the map after update $i+1$. Then, the update hint $\pi_{\delta i+1}$, returned by ComUpdate under the robustness of CH, provides the changes from the key-value map after the i-th update, $\mathcal{M}_i$, to the one after the $(i+1)$-th update $(\hat{\mathsf{k}}_{i+1}, \delta_{i+1})$, $\mathcal{M}_{i+1}$. Running on this $\pi_{\delta i+1}$ with the same $(\hat{\mathsf{k}}_{i+1}, \delta_{i+1})$, ProofUpdate will also make Λ_{i+1} an opening for the map $\mathcal{M}_{i+1}$ in k. As $(\mathsf{k}, \mathsf{v}_{i+1})$ is the key-value pair for k after the $(i+1)$-th update, $\mathsf{Ver}(\mathsf{crs}, C_{i+1}, \Lambda_{\mathsf{k}}^{(i+1)}, (\mathsf{k}, \mathsf{v}_{i+1}))$ will thus be equal to 1 from the correctness of VC.

Efficiency. We show that if VC and CH are efficient, and CH is robust, then KVM also is. Indeed, if they are, then crs is of polylogarithmic size, as well as commitments and openings under the efficiency of the VC. The auxiliary information input to ComUpdate is of polylogarithmic size, and the key and update information of constant size. Under the efficiency of CH, the output π_δ is polylogarithmic except with probability negligible in λ, as its size is at most in the number of elements moved by a CH.Insert operation, and in the stash size, which is also polylogarithmic if CH is robust. Under the efficiency of VC ComUpdate and ProofUpdate then both run in polylogarithmic time.

Part IV

CONCLUSION

9

CONCLUSION AND FUTURE WORK

In the second part of the thesis (Part II) we studied zero-knowledge protocols for the set membership problem. We provided succinct zero-knowledge proofs of set membership and non-membership for singletons using the RSA accumulator as the underlying set commitment. We also provided succinct zero-knowledge proofs of set membership of singletons using a position-hiding variant of EDRAX Vector Commitment as the underlying set commitment. Then we proceeded to succinct batch set membership proofs for multiple elements, again using the RSA accumulator.

There are various interesting research avenues open in this direction. One of the most interesting open problems would be to extend our methodology of batch set membership proofs for RSA accumulators to succinct batch zero-knowledge set non-membership proofs, i.e. provding that multiple (committed) elements are not members of the set but preserving the small size of the proof. Another possible future work is to construct batch proofs where the zero-knowledge property is perfect (or statistical) instead of computational. In our batch set-membership protocol zero-knowledge holds under a computational assumption (DDH-II) over groups of unknown order. Achieving perfect zero-knowledge would lead to everlasting privacy for the elements we prove membership for. This would translate to privacy even against quantum computers.

On a more general view, another possible research avenue would be to investigate post-quantum zero-knowledge poofs of set membership. With recent advances in succinct lattice-based commitments and zero-knowledge proofs this direction seems more tangible in the lattice cryptographic setting.

The most important conclusion is that specialized SNARKs for specific relations can be far more beneficial than applying general purpose SNARKs. In our case we started from a relation (the one of RSA Accumulators) that was in practice infeasible to prove via general purpose constructions. However, it turned out that special algebraic techniques relevant to the specific idiosyncracy of the problem can make a zero-knowledge proof feasible.

This conclusion can be generalized. A great research effort has been put to construct and improve SNARKs for all (NP) languages. Our thesis indicates that it is reasonable to dedicate efforts for special-type systems for specific languages. This conclusion could be strengthened by the fact that in real-world applications the types of languages that SNARKs are used for are essentially not many or at least are not very diverse.

Therefore, identifying interesting relations, their underlying properties and study the development of specialized SNARKs for these relations can be another fruitful research avenue.

In the third part of the thesis (Part III) we switch to vector commitments. Firsty, we introduced the notion of Incremental Aggregation for vector commitments and provide two efficient constructions from groups of unknown order. We show concrete applications of incrementally aggregatable vector commitments to space-time efficient precomputation of proofs of opening and Verifiable Decentralized Storage. Then we turned to functional commitments, presenting constructions of functional commitment with constant-sized public parameters and opening proofs for linear functions. The constructions are in the groups of unknown order setting. At the end we showed a generic way to transform any vector commitment to a key-value map commitment using a well-known probabilistic data structure, Cuckoo-Hashing.

Again, one can think of many fascinating open problems for future work. An intriguing and challenging open problem is to construct incrementally aggregatable vector commitments from other cryptographic settings such as bilinear groups or lattices. This is probably the most challenging open problem left in the thesis. It would be surpising if a black-box algebraic construction from bilinear groups exists that supports fully-fledged incremental aggregation. However, it would be interesting to understand what is the closest weaker notion of incremental aggregation that is feasible in this setting. On the other hand, a construction over lattices may be more tangible.

In case a construction of incrementally aggregatable vector commitments from other–than groups of unknown order–settings is not found it would make sense to formally understand what is the structure of groups of unknown order that makes it feasible in comparison to the other settings.

On the functional commitments' side there are many relevant open questions standing such as "Can we achieve inner product functional commitments with constant parameters from other settings?". Or "Is there a non-trivial way to generalize the result to a richer (than linear) class of functions", where the "trivial" way for higher degree polynomials would be to just linearize them.

Functional commitments is a very active research area and many constructions with rich functionalities have appeared, mostly from other settings, lattices or bilinear groups. An outstanding question is why in other settings the constructions require linear–in the size of the vector–public parameters or super-constant proofs (unless not fully succinct or using non black-box general purpose proof systems) and, again, what is the structural property of groups of unknown order that allows for constant-sized public parameters.

A slightly different direction would be to investigate 'direct' construction of functional commitments over groups of unknown order. By 'direct' we mean constructions that make only black-box use of group operations and do not arithmetize the underying function. Surpisingly, such constructions exist in bilinear groups and lattices but remain hard to build in the setting of groups of unknown order. This is a very challenging problem in the area that remains open for a long time.

Finally, for key-value map commitments it remains an open problem to explore if one could use the same approach of vector commitments and cuckoo-hashing but avoiding the linear–in the security parameter–overhead in the opening proof size. More generally, since cuckoo-hashing is underexplored in public-key cryptography it's fascinating to investigate if it could have further applications on that domain.

Part V

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Semaphore rln, rate limiting nullifier for spam prevention in anonymous p2p setting, February 2019. https://ethresear.ch/t/semaphore-rln-rate-limiting-nullifier-for-spam-prevention-in-anonymous-p2p-setting/5009.

[2] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 643–673. Springer, Heidelberg, August 2018.

[3] Shashank Agrawal and Srinivasan Raghuraman. KVaC: Key-Value Commitments for blockchains and beyond. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 839–869. Springer, Heidelberg, December 2020.

[4] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

[5] Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri Aravinda Krishnan Thyagarajan. Lattice-based SNARKs: Publicly verifiable, preprocessing, and recursively composable - (extended abstract). In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 102–132. Springer, Heidelberg, August 2022.

[6] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

[7] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018.

[8] Arasu Arun, Chaya Ganesh, Satya V. Lokam, Tushar Mopuri, and Sriram Sridhar. Dew: A transparent constant-sized polynomial commitment scheme. In Alexandra Boldyreva

and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 542–571. Springer, Heidelberg, May 2023.

[9] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12:1–12:34, June 2011.

[10] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Provable data possession at untrusted stores. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 598–609. ACM Press, October 2007.

[11] Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 549–579, Virtual Event, August 2021. Springer, Heidelberg.

[12] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 113–142. Springer, Heidelberg, November 2022.

[13] Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014.

[14] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy*, pages 271–286. IEEE Computer Society Press, May 2015.

[15] David Balbás, Dario Catalano, Dario Fiore, and Russell W. F. Lai. Chainable functional commitments for unbounded-depth circuits. Cryptology ePrint Archive, Paper 2022/1365, 2022. https://eprint.iacr.org/2022/1365.

[16] Foteini Baldimtsi, Ioanna Karantaidou, and Srinivasan Raghuraman. Oblivious accumulators. *Cryptology ePrint Archive*, 2023.

[17] Endre Bangerter, Jan Camenisch, and Stephan Krenn. Efficiency limitations for S-protocols for group homomorphisms. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 553–571. Springer, Heidelberg, February 2010.

[18] Niko Bari and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494. Springer, Heidelberg, May 1997.

[19] James Bartusek, Fermi Ma, and Mark Zhandry. The distinction between fixed and random generators in group-based assumptions. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 801–830. Springer, Heidelberg, August 2019.

[20] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and

Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 368–385. Springer, Heidelberg, September 2018.

[21] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[22] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

[23] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Heidelberg, August 2019.

[24] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[25] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

[26] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.

[27] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 5:1–5:32. LIPIcs, January 2020.

[28] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 111–131. Springer, Heidelberg, August 2011.

[29] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, Heidelberg, May 1994.

[30] Daniel Benarroch, Matteo Campanelli, and Dario Fiore. Community standards proposal for commit-and-prove zero-knowledge proof systems, 2019. https://www.binarywhales.com/assets/misc/zkproof-cp-standards.pdf.

[31] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. In Nikita

Borisov and Claudia Díaz, editors, *FC 2021, Part I*, volume 12674 of *LNCS*, pages 393–414. Springer, Heidelberg, March 2021.

[32] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: efficient, succinct, modular. *Des. Codes Cryptogr.*, 91(11):3457–3525, 2023.

[33] Fabrice Benhamouda, Stephan Krenn, Vadim Lyubashevsky, and Krzysztof Pietrzak. Efficient zero-knowledge proofs for commitments from learning with errors over rings. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *ESORICS 2015, Part I*, volume 9326 of *LNCS*, pages 305–325. Springer, Heidelberg, September 2015.

[34] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of statistical physics*, 22:563–591, 1980.

[35] Jean-François Biasse, Michael J. Jacobson, and Alan K. Silvester. Security estimates for quadratic field based cryptosystems. In Ron Steinfeld and Philip Hawkes, editors, *ACISP 10*, volume 6168 of *LNCS*, pages 233–247. Springer, Heidelberg, July 2010.

[36] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017.

[37] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *CRYPTO'81*, volume ECE Report 82-04, pages 11–15. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981.

[38] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[39] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712.

[40] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

[41] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[42] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, Heidelberg, May 2011.

[43] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

[44] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

[45] Wieb Bosma and Peter Stevenhagen. On the computation of quadratic 2-class groups. *Journal de théorie des nombres de Bordeaux*, 8(2):283–313, 1996.

[46] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences*, 37(2):156–189, 1988.

[47] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proc. of the ACM SOSP*, 2013.

[48] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 644–662. Springer, Heidelberg, December 2012.

[49] Johannes Buchmann and Safuat Hamdy. A survey on iq cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–16. De Gruyter, 2011.

[50] Johannes Buchmann and Hugh C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, June 1988.

[51] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable certificate management using undeniable attestations. In Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati, editors, *ACM CCS 2000*, pages 9–17. ACM Press, November 2000.

[52] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[53] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

[54] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[55] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 402–414. Springer, Heidelberg, May 1999.

[56] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 481–500. Springer, Heidelberg, March 2009.

[57] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, August 2002.

[58] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2021.

[59] Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. Incrementally aggregatable vector commitments and applications to verifiable decentralized storage. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 3–35. Springer, Heidelberg, December 2020.

[60] Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. Succinct zero-knowledge batch proofs for set accumulators. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 455–469. ACM Press, November 2022.

[61] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.

[62] Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. Linear-map vector commitments and their practical applications. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 189–219. Springer, Heidelberg, December 2022.

[63] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 455–469. Springer, Heidelberg, August 1997.

[64] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[65] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Heidelberg, December 2018.

[66] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Heidelberg, February / March 2013.

[67] Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 433–450. Springer, Heidelberg, April 2008.

[68] Dario Catalano, Dario Fiore, and Ida Tucker. Additive-homomorphic functional commitments and applications to homomorphic signatures. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 159–188. Springer, Heidelberg, December 2022.

[69] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985.

[70] David Chaum. Showing credentials without identification: Signatures transferred between unconditionally unlinkable pseudonyms. In Franz Pichler, editor, *EUROCRYPT'85*, volume 219 of *LNCS*, pages 241–244. Springer, Heidelberg, April 1986.

[71] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, Heidelberg, April 2023.

[72] Brian Chen, Yevgeniy Dodis, Esha Ghosh, Eli Goldin, Balachandar Kesavan, Antonio Marcedone, and Merry Ember Mou. Rotatable zero knowledge sets - post compromise secure auditable dictionaries with application to key transparency. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 547–580. Springer, Heidelberg, December 2022.

[73] Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang. Edrax: A cryptocurrency with stateless transaction validation. 2018.

[74] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[75] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020.

[76] Hien Chu, Dario Fiore, Dimitris Kolonelos, and Dominique Schröder. Inner product functional commitments with constant-size public parameters and openings. In *International Conference on Security and Cryptography for Networks*, pages 639–662. Springer, 2022.

[77] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, pages 46–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[78] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy*, pages 253–270. IEEE Computer Society Press, May 2015.

[79] Geoffroy Couteau and Dominik Hartmann. Shorter non-interactive zero-knowledge arguments and zaps for algebraic languages. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*, pages 768–798. Springer, 2020.

[80] Geoffroy Couteau, Helger Lipmaa, Roberto Parisella, and Arne Tobias Ødegaard. Efficient nizks for algebraic sets. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III*, pages 128–158. Springer, 2021.

[81] Geoffroy Couteau, Thomas Peters, and David Pointcheval. Removing the strong RSA assumption from arguments over the integers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 321–350. Springer, Heidelberg, April / May 2017.

[82] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. https://eprint.iacr.org/2006/291.

[83] Ronald Cramer. Modular design of secure yet practical cryptographic protocols. *Ph. D. Thesis, CWI and University of Amsterdam*, 1996.

[84] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In Juzar Motiwalla and Gene Tsudik, editors, *ACM CCS 99*, pages 46–51. ACM Press, November 1999.

[85] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 125–142. Springer, Heidelberg, December 2002.

[86] Ivan Damgard, Carmit Hazay, and Angela Zottarel. Short paper on the generic hardness of ddh-ii. 2014.

[87] Ivan Damgård and Maciej Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 256–271. Springer, Heidelberg, April / May 2002.

[88] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. https://eprint.iacr.org/2008/538.

[89] Leo de Castro and Chris Peikert. Functional commitments for all functions, with transparent setup and from SIS. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 287–320. Springer, Heidelberg, April 2023.

[90] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 64–93. Springer, Heidelberg, December 2020.

[91] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 127–144. Springer, Heidelberg, April 2015.

[92] Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68, 2007.

[93] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[94] Samuel Dobson, Steven Galbraith, and Benjamin Smith. Trustless unknown-order groups. *arXiv preprint arXiv:2211.16128*, 2022.

[95] Justin Drake. Accumulators, scalability of utxo blockchains, and data availability. https://ethresear.ch/t/accumulators-scalability-of-utxo-blockchains-and-data-availability/176, 2017.

[96] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. *Cryptology ePrint Archive*, 2022.

[97] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[98] Alex Escala and Jens Groth. Fine-tuning Groth-Sahai proofs. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 630–649. Springer, Heidelberg, March 2014.

[99] Shimon Even. Protocol for signing contracts. In Allen Gersho, editor, *CRYPTO'81*, volume ECE Report 82-04, pages 148–153. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981.

[100] Nelly Fazio and Antonio Nicolosi. Cryptographic accumulators: Definitions, constructions and applications. *Paper written for course at New York University: www. cs. nyu. edu/nicolosi/papers/accumulators. pdf*, 2002.

[101] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[102] FINRA. https://www.finra.org/rules-guidance/rulebooks/finra-rules/2090#the-rule.

[103] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1304–1316. ACM Press, October 2016.

[104] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces. *Cryptology ePrint Archive*, 2023.

[105] Ben Fisch. Poreps: Proofs of space on useful data. Cryptology ePrint Archive, Report 2018/678, 2018. https://eprint.iacr.org/2018/678.

[106] Dimitris Fotakis, R. Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38:229–248, 2003.

[107] Nikolaos Fountoulakis, Konstantinos Panagiotou, and Angelika Steger. On the insertion time of cuckoo hashing, 2013.

[108] Pierre-Alain Fouque and Mehdi Tibouchi. Close to uniform prime number generation with fewer random bits. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 991–1002. Springer, Heidelberg, July 2014.

[109] Alan M. Frieze and Tony Johansson. On the insertion time of random walk cuckoo hashing. *CoRR*, abs/1602.04652, 2016.

[110] Alan M. Frieze, Páll Melsted, and Michael Mitzenmacher. An analysis of random-walk cuckoo hashing. In *International Workshop and International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2009.

[111] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 16–30. Springer, Heidelberg, August 1997.

[112] Ariel Gabizon and Dmitry Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. *Cryptology ePrint Archive*, 2022.

[113] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[114] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[115] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 123–139. Springer, Heidelberg, May 1999.

[116] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[117] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[118] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[119] Esha Ghosh, Olga Ohrimenko, Dimitrios Papadopoulos, Roberto Tamassia, and Nikos Triandopoulos. Zero-knowledge accumulators and set algebra. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 67–100. Springer, Heidelberg, December 2016.

[120] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984.

[121] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[122] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[123] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2007–2023. ACM Press, November 2020.

[124] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

[125] Jens Groth. Non-interactive zero-knowledge arguments for voting. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *ACNS 05*, volume 3531 of *LNCS*, pages 467–482. Springer, Heidelberg, June 2005.

[126] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[127] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018.

[128] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

[129] Safuat Hamdy and Bodo Möller. Security of cryptosystems based on class groups of imaginary quadratic orders. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 234–247. Springer, Heidelberg, December 2000.

[130] Samer Hassan and Primavera De Filippi. Decentralized autonomous organization. *Internet Policy Review*, 10(2):1–10, 2021.

[131] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*, pages 267–288. Springer, Heidelberg, June 1998.

[132] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.*, 2016. https://github.com/zcash/zips/blob/master/protocol/sapling.pdf.

[133] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021.

[134] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Heidelberg, February 2002.

[135] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 584–597. ACM Press, October 2007.

[136] Ioanna Karantaidou and Foteini Baldimtsi. Efficient constructions of pairing based accumulators. In Ralf Küsters and Dave Naumann, editors, *CSF 2021 Computer Security Foundations Symposium*, pages 1–16. IEEE Computer Society Press, 2021.

[137] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[138] Auguste Kerckhoffs. *La cryptographie militaire, ou, Des chiffres usités en temps de guerre: avec un nouveau procédé de déchiffrement applicable aux systèmes à double clef*. Librairie militaire de L. Baudoin, 1883.

[139] Megha Khosla. Balls into bins made faster. In *Embedded Systems and Applications*, 2013.

[140] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM Journal on Computing*, 39(4):1543–1561, 2010.

[141] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. TRUESET: Faster verifiable set computations. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 765–780. USENIX Association, August 2014.

[142] John Kuszmaul. Verkle trees: V(ery short m)erkle trees, 2018.

[143] John Kuszmaul. Verkle trees. *Verkle Trees*, 1:1, 2019.

[144] Protocol Labs. Filecoin: A decentralized storage network, 2017. https://filecoin.io/filecoin.pdf.

[145] Russell W. F. Lai and Giulio Malavolta. Subvector commitments with application to succinct arguments. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 530–560. Springer, Heidelberg, August 2019.

[146] Jonathan Lee. The security of groups of unknown order based on jacobians of hyperelliptic curves. Cryptology ePrint Archive, Report 2020/289, 2020. https://eprint.iacr.org/2020/289.

[147] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 253–269. Springer, Heidelberg, June 2007.

[148] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 1–31. Springer, Heidelberg, May 2016.

[149] Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl, July 2016.

[150] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 499–517. Springer, Heidelberg, February 2010.

[151] Helger Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 398–415. Springer, Heidelberg, November / December 2003.

[152] Helger Lipmaa. Secure accumulators from euclidean rings without trusted setup. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 12*, volume 7341 of *LNCS*, pages 224–240. Springer, Heidelberg, June 2012.

[153] Helger Lipmaa and Roberto Parisella. Set (non-) membership nizks from determinantal accumulators. *Cryptology ePrint Archive*, 2022.

[154] Helger Lipmaa and Kateryna Pavlyk. Succinct functional commitment for a large class of arithmetic circuits. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 686–716. Springer, Heidelberg, December 2020.

[155] Helger Lipmaa, Janno Siim, and Michal Zajac. Counting vampires: From univariate sumcheck to updatable ZK-SNARK. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 249–278. Springer, Heidelberg, December 2022.

[156] Jing Liu and Liang Feng Zhang. Matproofs: Maintainable matrix commitment with efficient aggregation. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2041–2054. ACM Press, November 2022.

[157] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 74–90. Springer, Heidelberg, May 2004.

[158] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. SMILE: Set membership from ideal lattices with applications to ring signatures and confidential transactions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 611–640, Virtual Event, August 2021. Springer, Heidelberg.

[159] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

[160] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005.

[161] Ralph C Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[162] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988.

[163] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

[164] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE Computer Society Press, May 2013.

[165] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.

[166] Moni Naor. Bit commitment using pseudo-randomness. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 128–136. Springer, Heidelberg, August 1990.

[167] Lan Nguyen. Accumulators from bilinear pairings and applications. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 275–292. Springer, Heidelberg, February 2005.

[168] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. Scaling verifiable computation using efficient set accumulators. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2075–2092. USENIX Association, August 2020.

[169] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[170] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 222–242. Springer, Heidelberg, March 2013.

[171] Charalampos Papamanthou, Elaine Shi, Roberto Tamassia, and Ke Yi. Streaming authenticated data structures. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 353–370. Springer, Heidelberg, May 2013.

[172] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[173] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 79–93. ACM Press, November 2019.

[174] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[175] Chris Peikert. A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939, 2015. https://eprint.iacr.org/2015/939.

[176] Chris Peikert, Zachary Pepin, and Chad Sharp. Vector and functional commitments from lattices. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 480–511. Springer, Heidelberg, November 2021.

[177] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 502–519. Springer, Heidelberg, August 2010.

[178] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 515–530. USENIX Association, August 2015.

[179] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over gf (p) and its cryptographic significance (corresp.). *IEEE Transactions on information Theory*, 24(1):106–110, 1978.

[180] Paul Pollack and Enrique Treviño. Finding the four squares in lagrange's theorem. *Integers*, 18A:A15, 2018.

[181] John M Pollard. Monte carlo methods for index computation mod p. *Mathematics of computation*, 32(143):918–924, 1978.

[182] Jim Posen and Assimakis A Kattis. Caulk+: Table-independent lookup arguments. *Cryptology ePrint Archive*, 2022.

[183] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Heidelberg.

[184] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[185] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. https://eprint.iacr.org/2006/145.

[186] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[187] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 555–572. Springer, Heidelberg, August 1999.

[188] Tomas Sander, Amnon Ta-Shma, and Moti Yung. Blind, auditable membership proofs. In Yair Frankel, editor, *FC 2000*, volume 1962 of *LNCS*, pages 53–71. Springer, Heidelberg, February 2001.

[189] U.S. Securities and Exchange Commission. Anti-money laundering (aml) source tool for broker-dealers, October 2018. https://www.sec.gov/about/offices/ocie/amlsourcetool.htm.

[190] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020.

[191] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020. https://eprint.iacr.org/2020/1275.

[192] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems (TOCS)*, 1(1):38–44, 1983.

[193] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.

[194] Adi Shamir, Ronald L Rivest, and Leonard M Adleman. *Mental poker*. Springer, 1981.

[195] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[196] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

[197] Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. Hyperproofs: Aggregating and maintaining proofs in vector commitments. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 3001–3018. USENIX Association, August 2022.

[198] Shravan Srinivasan, Ioanna Karantaidou, Foteini Baldimtsi, and Charalampos Papamanthou. Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2719–2733, 2022.

[199] Roberto Tamassia. Authenticated data structures. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003*, pages 2–5, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[200] Björn Terelius and Douglas Wikström. Efficiency limitations of S-protocols for group homomorphisms revisited. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 461–476. Springer, Heidelberg, September 2012.

[201] Peter Todd. Making utxo set growth irrelevant with low-latency delayed txo commitments. https://petertodd.org/2016/delayed-txo-commitments, 2016.

[202] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 45–64. Springer, Heidelberg, September 2020.

[203] Alin Tomescu, Yu Xia, and Zachary Newman. Authenticated dictionaries with cross-incremental proof (dis)aggregation. Cryptology ePrint Archive, Report 2020/1239, 2020. https://eprint.iacr.org/2020/1239.

[204] V.A. Hyperledger indy. https://www.hyperledger.org/use/hyperledger-indy, 2022.

[205] V.A. Iden3. https://iden3.io, 2022.

[206] V.A. Sovrin. https://sovrin.org, 2022.

[207] V.A. Zcash. https://z.cash, 2022.

[208] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.

[209] Alexander Vlasov and Konstantin Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. Cryptology ePrint Archive, Report 2019/1020, 2019. https://eprint.iacr.org/2019/1020.

[210] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

[211] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, jan 2015.

[212] Stefan Walzer. Insertion time of random walk cuckoo hashing below the peeling threshold, 2022.

[213] Weijie Wang, Annie Ulichney, and Charalampos Papamanthou. {BalanceProofs}: Maintainable vector commitments with fast aggregation. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4409–4426, 2023.

[214] Hoeteck Wee and David J. Wu. Succinct vector, polynomial, and functional commitments from lattices. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 385–416. Springer, Heidelberg, April 2023.

[215] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

[216] Udi Wieder et al. Hashing, load balancing and multiple choice. *Foundations and Trends® in Theoretical Computer Science*, 12(3–4):275–379, 2017.

[217] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[218] Reuben Yap. Cryptographic description of zerocoin attack, 2019. https://zcoin.io/cryptographic-description-of-zerocoin-attack/.

[219] Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 197–230, Cham, 2023. Springer Nature Switzerland.

[220] Zuoxia Yu, Man Ho Au, Rupeng Yang, Junzuo Lai, and Qiuliang Xu. Lattice-based universal accumulator with nonmembership arguments. In *Information Security and Privacy: 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings 23*, pages 502–519. Springer, 2018.

[221] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3121–3134, 2022.

[222] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. *Cryptology ePrint Archive*, 2022.

[223] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876. IEEE Computer Society Press, May 2020.

[224] Y. Zhang, J. Katz, and C. Papamanthou. An expressive (zero-knowledge) set accumulator. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 158–173, April 2017.

[225] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy*, pages 863–880. IEEE Computer Society Press, May 2017.

[226] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A zero-knowledge version of vSQL. Cryptology ePrint Archive, Report 2017/1146, 2017. https://eprint.iacr.org/2017/1146.

[227] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy*, pages 908–925. IEEE Computer Society Press, May 2018.

[228] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. An expressive (zero-knowledge) set accumulator. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 158–173. IEEE, 2017.