Theory and Applications of Verifiable Delay Functions

by

Charlotte Hoffmann

October, 2025

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Maximilian Jösch, Chair
Krzysztof Pietrzak
Tim Browning
Alon Rosen

The thesis of Charlotte Hoffmann, titled *Theory and Applications of Verifiable Delay Functions*, is approved by:

Supervisor: Krzysztof Pietrzak, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Tim Browning, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Alon Rosen, Bocconi University, Milan, Italy

Signature: _____

Defense Chair: Maximilian Jösch, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

ISTA Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I accept full responsibility for the content and factual accuracy of this work, including the data and their analysis and presentation, and the text and citation of other work.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Charlotte Hoffmann
October, 2025

Signed page is on file

Abstract

Verifiable Delay Functions (VDFs) introduced by Boneh et al. (CRYPTO'18) are functions that require a prescribed number of sequential steps T to evaluate, yet their output can be verified in time much faster than T. Since their introduction, VDFs have gained a lot of attention due to their applications in blockchain protocols, randomness beacons, timestamping and deniability. This thesis explores the theory and applications of VDFs, focusing on enhancing their soundness, efficiency and practicality.

The only practical VDFs known to date are based on repeated squaring in hidden order groups. Consider the function $\mathsf{VDF}(x, T) = x^{2^T}$. The iterated squaring assumption states that, for a random group element x, the result of VDF cannot be computed significantly faster than performing T sequential squarings if the group order is unknown. To make the result verifiable a prover can compute a proof of exponentiation (PoE) π. Given π, the output of VDF can be verified in time much less than T.

We first present new constructions of statistically sound proofs of exponentiation, which are an important building block in the construction of SNARKs (Succinct Non-Interactive Argument of Knowledge). Statistical soundness means that the proofs remain secure against computationally unbounded adversaries, in particular, it remains secure even when the group order is known. We thereby address limitations in previous PoE protocols which either required (non-standard) hardness assumptions or a lot of parallel repetitions. Our construction significantly reduces the proof size of statistically sound PoEs that allow for a structured exponent, which leads to better efficiency of SNARKs and other applications.

Secondly, we introduce improved batching techniques for PoEs, which allow multiple proofs to be aggregated and verified with minimal overhead. These protocols optimize communication and computation complexity in large-scale blockchain environments and enable scalable remote benchmarking of parallel computation resources.

We then construct VDFs with enhanced properties such as zero-knowledge and watermarkability. It was shown by Arun, Bonneau and Clark (ASIACRYPT'22) that these features enable new cryptographic primitives called short-lived proofs and signatures. The validity of such proofs and signatures expires after a predefined amount of time T, i.e., they are deniable after time T. Our constructions improve upon the constructions by Arun, Bonneau and Clark in several dimensions (faster forging times, arguably weaker assumptions).

Finally, we apply PoEs in the realm of primality testing, providing cryptographically sound proofs of non-primality for large Proth numbers. This work gives a surprising application of VDFs in the area of computational number theory.

Together, our contributions advance both the theoretical foundations and the real-world usability of VDFs in general and in particular of PoEs, making them more adaptable and secure for current and emerging cryptographic applications.

Acknowledgements

First and foremost I would like to thank my advisor Krzysztof Pietrzak for introducing me to the world of cryptographic research, for giving me the opportunity to be a part of his amazing group, for sharing his ideas, for his support and for many inspiring discussions.

I also want to thank Pavel Hubáček and Alon Rosen for hosting me on my two research visits and for their guidance throughout the rest of my PhD. I consider myself very lucky to have found two additional mentors in you.

To all the current and former members of the Crypto group: Ahad, Akin, Anshu, Benedikt, Chethan, Christoph, Guillermo, Karen, Michael, Michelle, Miguel and Ray. Thank you for many productive discussions and for many silly discussions. Thank you for ranting together (looking at you anonymous reviewer) and for celebrating together.

Thanks to my coauthors Alon, Andrej, Benedikt, Chethan, Guillermo, Juraj, Karen, Kristýna, Krzysztof, Mark, Martin, Miguel, Pavel, Pavel D., Svetlana and Tomáš for the nice collaboration and thanks to Rita and Christine for their support with everything administrative. Thanks to Tim Browning for being part of my thesis committee.

I also want to thank all of my friends from Cologne and Vienna, my sister Franzi, my grandparents Arnold and Hildegard and my partner Ben. Thank you all for believing in me and for making my life so bright and colorful.

Finally, I thank my mother Birgit for her unconditional love and support. You will always be my most important mentor.

About the Author

Charlotte Hoffmann completed a BSc in Mathematics at the University of Cologne and an MSc in Mathematics at the University of Vienna. In 2020, she joined ISTA for her PhD studies in the research group of Krzysztof Pietrzak.

Her main research interests are time-released cryptography and provable security. She has also worked on projects related to post-quantum encryption, generic lower bounds and secret sharing. During her PhD she visited Alon Rosen at Bocconi University in Milan and Pavel Hubáček at the Czech Academy of Sciences in Prague.

List of Collaborators and Publications

1. Chapter 3 is based on "Charlotte Hoffmann, Pavel Hubáček, Chethan Kamath, Karen Klein, Krzysztof Pietrzak. *Practical Statistically-Sound Proofs of Exponentiation in any Group*. In Advances in Cryptology - **CRYPTO 2022** [HHK+22]".

   [significant contribution on entire paper, most technical details carried out by herself]

2. Chapter 4 is based on "Charlotte Hoffmann, Pavel Hubáček, Svetlana Ivanova. *Practical Batch Proofs of Exponentiation*. IACR Communications in Cryptology, vol. 2, no. 3, Oct 06, 2025 [HHI25]".

   [significant contributions to the protocol design and security analysis]

3. Chapter 5 is based on "Charlotte Hoffmann, Krzyzstof Pietrzak. *Watermarkable and Zero-Knowledge Verifiable Delay Functions from any Proof of Exponentiation*. In International Conference on Theory and Practice of Public Key Cryptography - **PKC 2025** [HP25]".

   [significant contribution on entire paper, most technical details carried out by herself]

4. Chapter 6 is based on " Charlotte Hoffmann, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak. *Certifying Giant Nonprimes*. In International Conference on Theory and Practice of Public Key Cryptography - **PKC 2023** [HHKP23]".

   [significant contribution on entire paper, most technical details carried out by herself]

The following list contains other works written during the PhD period but are not included in the thesis.

5. Andrej Bogdanov, Miguel Cueto Noval, Charlotte Hoffmann, Alon Rosen. *Public-Key Encryption from Homogeneous CLWE*. In Theory of Cryptography - **TCC 2022** [BNHR22].

6. Charlotte Hoffmann, Mark Simkin.*Stronger Lower Bounds for Leakage-Resilient Secret Sharing*. In International Conference on Cryptology and Information Security in Latin America - **LATINCRYPT 2023** [HS23].

7. Benedikt Auerbach, Charlotte Hoffmann, Guillermo Pascual-Perez.*Generic-Group Lower Bounds via Reductions Between Geometric-Search Problems: With and Without Preprocessing*. In Theory of Cryptography - **TCC 2023** [AHPP23].

8. Charlotte Hoffmann, Pavel Hubáček, Chethan Kamath, Tomáš Krňák. *(Verifiable) Delay Functions from Lucas Sequences*. In Theory of Cryptography - **TCC 2023** [HHKK23].

9. Charlotte Hoffmann. *Traceable Secret Sharing based on the Chinese Remainder Theorem.* Preprint [Hof24].

10. Juraj Belohorec, Pavel Dvořák, Charlotte Hoffmann, Pavel Hubáček, Kristýna Mašková, Martin Pastyřík. *On Extractability of the KZG Family of Polynomial Commitment Schemes.* To appear in Advances in Cryptology - **CRYPTO 2025** [BDH+25].

Table of Contents

Abstract vii

Acknowledgements viii

About the Author ix

List of Collaborators and Publications x

Table of Contents xiii

List of Figures xiv

List of Tables xvi

1 Introduction 1
- 1.1 Statistically Sound Proofs of Exponentiation . . . 4
- 1.2 Batching PoEs . . . 6
- 1.3 Short-Lived Proofs from VDFs . . . 7
- 1.4 Primality Testing from PoEs . . . 8
- 1.5 Other Related Work . . . 9

2 Preliminaries 11
- 2.1 Relations and Interactive Proofs . . . 11
- 2.2 Verifiable Delay Functions . . . 12
- 2.3 The Group of Signed Quadratic Residues . . . 13
- 2.4 Assumptions . . . 13
- 2.5 Known Proofs of Exponentiation and Batching Protocols . . . 15

3 Statistically Sound Proofs of Exponentiation 19
- 3.1 Introduction . . . 19
- 3.2 Basic Protocol . . . 23
- 3.3 Reducing Verifier's Complexity by Modifying q . . . 32
- 3.4 Reducing (Verifier's) Complexity by Batching . . . 33
- 3.5 Application in Polynomial Commitments . . . 37
- 3.6 Conclusion . . . 41

4 Batching Proofs of Exponentiation 43
- 4.1 Introduction . . . 43
- 4.2 The Hybrid Protocol . . . 49
- 4.3 The Bucket Protocol . . . 52

4.4 Comparison . . . 54
4.5 Communication-Optimal Remote Parallel Benchmarking via Batch PoEs . . 56
4.6 Conclusion . . . 63

5 Practical Short-Lived Proofs from Verifiable Delay Functions 65
5.1 Introduction . . . 65
5.2 Three Zero-Knowledge Proofs of Knowledge . . . 68
5.3 Modified Discrete-Log Assumptions . . . 72
5.4 Watermarkable VDFs . . . 73
5.5 Zero-Knowledge VDFs . . . 77
5.6 Zero-Knowledge Proofs of Sequential Work . . . 80
5.7 Short-Lived Proofs from our Zero-Knowledge PoSW . . . 82
5.8 Conclusion . . . 84

6 Primality Testing from Proofs of Exponentiations 85
6.1 Introduction . . . 85
6.2 Pietrzak's PoE in Groups of Known Order . . . 91
6.3 Attacking Pietrzak's Protocol in Proth Number Groups . . . 93
6.4 Certifying Non-Primality of Proth Primes . . . 94
6.5 Conclusion . . . 100

7 Conclusion 103

Bibliography 105

List of Figures

2.1 Wesolowski's PoE [Wes20]. Primes(λ) denotes the set of the first λ prime numbers. . . . 15
2.2 Pietrzak's PoE [Pie19]. . . . 16
2.3 Block et al.'s PoE [BHR+21]. . . . 16
2.4 Known batch PoEs. . . . 17

3.1 For 80-bit security, (a) the number of (group) elements sent by the prover *per round* and (b) the number of (group) multiplications carried out by verifier, also per round, plotted for different values of the bound B. The dotted blue line, the solid orange curve, and the dashed green line represent, respectively, [BHR+21], our protocol, and [Pie19]. In Figure 3.3 we dissect the solid orange curve in (b). 20
3.2 Our basic Proof of Exponentiation. . . . 26

3.3 Number of multiplications of the verifier in one round for 80-bit security depending on the bound B. The orange solid curve is the total verifier's complexity for one round, the blue dotted graph is the cost of the interactive part of the protocol and the green dashed graph is the cost of the final exponentiation divided by the number of rounds (i.e., we amortize the cost of the final exponentiation over the number of rounds). 31
3.4 Number of multiplications of the verifier in one round for 80-bit security depending on the bound B. The solid blue line represents the number of multiplications in [BHR+21], the dotted orange curve represents the complexity of our protocol with $C = t\log(B)$, the solid red curve is the complexity in our protocol with $C = t\log(B)/2$ and the solid green line represents the verifier's complexity in [Pie19]. 33
3.5 Number of multiplications of the verifier in one round for 80-bit security depending on the bound B. The dotted blue line represents the number of multiplications in [BHR+21], the solid orange curve is the complexity of our protocol with $C = t$ and q as above and the dashed green line is verifier's complexity in [Pie19] (which is 240 multiplications). 33
3.6 Batching protocol for PoE. 35

4.1 Depiction of the known batch PoEs. 46
4.2 Depiction of the new batch PoEs. 46
4.3 Our Hybrid Protocol. 50
4.4 Our Bucket Protocol based on the bucket test from Bellare et al. [BGR98]. . . 53
4.5 The relative (4.5a) and absolute (4.5b) numbers of multiplications on 100 to 10^{14} instances compared to the random exponent batching approach from Rotem [Rot21] with the security parameter $\lambda = 128$ and optimal choice of k in the Bucket Protocol. 55
4.6 The relative (4.6a) and absolute (4.6b) time performance on 100 to 10M instances compared to the random exponent batching approach from Rotem [Rot21], with the security parameter $\lambda = 128$ and optimal choice of k in the Bucket Protocol. 56
4.7 A proof of sufficient resources to solve m repeated squaring instances in time less than $T + t$. 57

5.1 Proof of Knowledge of Discrete Log [Sch91, KTY04]. 69
5.2 Proof of Knowledge of same discrete log [KTY04] 70
5.3 PoKsDL: The watermarked non-interactive Proof of Knowledge of same discrete log 71
5.4 Proof of Knowledge of same discrete log with one hidden base 71
5.5 A Watermarkable VDF from any proof of exponentiation using the proof of knowledge PoKsDL presented in Figure 5.3. By $\mathtt{PoE}(\mathbf{pp}, x, y, T)$ we denote the chosen proof of exponentiation with group parameters $\mathbf{pp}$ and statement $x^{2^T} = y$. . . 75
5.6 A zero-knowledge VDF from any proof of exponentiation. PoKsDLh is the non-interactive version of the proof of knowledge presented in Figure 5.4. By $\mathtt{PoE}(\mathbf{pp}, x, y, T)$ we denote the chosen proof of exponentiation with group parameters $\mathbf{pp}$ and statement $x^{2^T} = y$. 79
5.7 A Zero-Knowledge Proof of Sequential Work from any proof of exponentiation PoE. PoKDL is the non-interactive version of the proof of knowledge presented in Figure 5.1. By $\mathtt{PoE}(\mathbf{pp}, x, y, T)$ we denote the chosen proof of exponentiation with group parameters $\mathbf{pp}$ and statement $x^{2^T} = y$. 81

5.8 A short-lived proof from our zero knowledge PoSW. PoKDL is the non-interactive version of the proof of knowledge presented in Figure 5.1. By $\texttt{PoE}(\mathbf{pp}, x, y, T)$ we denote the chosen proof of exponentiation with group parameters $\mathbf{pp}$ and statement $x^{2^T} = y$. 83

6.1 Overview of the protocol in Figure 6.3. All computations are done in the group $\mathbb{Z}_N^*$, where $N = k2^n + 1$. 89
6.2 An attack with success probability at least $1 - (1 - 1/\operatorname{ord}(\alpha))^{\log T}$. 93
6.3 The non-primality certificate. 96

List of Tables

3.1 Comparison of different PoEs. Verifier's complexity is measured in the number of multiplications and proof-size $|\pi|$ in the number of group elements. By λ, we denote the statistical security parameter. [Pie19] is statistically-sound only in groups without elements of small order. 32

4.1 The complexity of various batch PoEs for m instances with security parameter λ, and 2^k buckets in the Bucket Protocol. 54
4.2 The measured times (in seconds) for PoE batching approaches in a 2048-bit RSA group. The last row is extrapolated for the first two approaches due to excessive times. 55

6.1 Complexity of the protocol in Figure 6.3 depending on the step in which it outputs the result. Prover's and Verifier's complexity are measured in the number of multiplications and proof-size in the number of group elements. We denote by λ the statistical security parameter. 99

CHAPTER 1

Introduction

Resource-restricted cryptography considers problems that are moderately hard to break but not infeasible. It was initiated by Dwork and Naor [DN93] in 1992, who introduced the notion that today we call *proof of work* (PoW). A proof of work is a protocol that takes a prescribed amount of computational power to solve *on average*. Once the result is computed it can be quickly verified that the result is correct. One example of a proof of work is to find a preimage of a hash function output starting with d zeros, where d is a parameter that determines the hardness of the PoW. If the hash function behaves randomly, we don't know of any faster algorithm to solve this challenge than just brute-forcing different inputs to the hash function until we get a result of the required form. Then, given the preimage, anyone can quickly verify that it is a valid solution. Dwork and Naor introduced the notion of PoWs to prevent spam email: Assume that sending an email requires to solve a PoW that takes a few seconds on average. For an honest party that sends only a few emails per hour, this is not a problem. However, sending out spam emails in bulk now requires a lot of computation power.

In 1996, Rivest, Shamir and Wagner [RSW96] considered *time* as another computational resource that can be restricted. They introduced the notion of time lock puzzles (TLPs), which allow a sender to encode a message into a puzzle that can only be opened after T sequential steps of computation, i.e., after a prescribed amount of time T. The authors had several applications of encrypting a message "into the future" in mind, for example hiding a bid in an auction until after the bidding period is closed. Their construction is based on repeated squaring in RSA groups. To encode a message m, Alice samples a random key k and computes $c_1 = \mathsf{Enc}_k(m)$ with a suitable symmetric encryption scheme Enc. Then she samples two large random primes p and q and computes the modulus $N = pq$. Then Alice chooses a random $1 < x < N$ and computes $c_2 := k \cdot x^{2^T} \mod N$, publishes (N, T, c_1, c_2, x) as the puzzle and keeps p and q secret. Note that to generate the puzzle she can efficiently compute $\phi(N) = (p-1)(q-1)$, then $e := 2^T \mod \phi(N)$ and finally $c_2 := m \cdot x^e$. However, if one does not know the factorization of N, one cannot compute the reduced exponent e in this way since $\phi(N)$ is unknown. The *iterated squaring assumption* states that in groups of unknown order there is no significantly faster way to compute x^{2^T} than to perform T sequential squarings. This assumption was first stated in the work by Rivest, Shamir and Wagner and it still stands today.[1] Solving the TLP therefore requires to compute $y := x^{2^T}$

[1] Bernstein and Sorenson [BS07] showed that one can reduce the *sequential* time of computing an iterated squaring instance x^{2^T} from T to $T/\log\log(T)$ using at least T^2 processors that need to be able to

by performing T sequential squarings. Then the message can be obtained by computing $k = c_2 \cdot y^{-1}$ and then decrypting c_1 with k.

15 years after their introduction, PoWs found a completely different application. They played a crucial role in the construction of the bitcoin blockchain. In the white paper, Nakamoto [Nak08] used PoWs to construct a consensus protocol to decide on the order of blocks in a chain that works as follows: Every party participating in the consensus protocol keeps a copy of the chain. To add a new block to the chain, each party tries to solve a PoW on an input that is parameterized by the new block they want to append and the last block in their blockchain. Whoever solves the PoW the fastest, broadcasts the result to all parties. All parties that receive the message append the corresponding block to their chain and start trying to solve a new PoW parameterized by this new block and the block they want to append.

In 2013, Mahmoody, Moran and Vadhan [MMV13] considered another application of timed cryptography: non-interactive timestamping. Assume Alice wants to generate a proof that in the future certifies that she knew a document D already at time t_0. A natural idea would be for Alice to start solving a PoW with input D at time t_0. If we knew that it takes roughly time T to compute the PoW, then the PoW would show that Alice already knew D at least T timesteps ago. The authors observed that PoWs, as described above, are not suitable for this application since they are easily parallelizable: Given n processors, the PoW can be solved n times faster than with just one processor. Note that the TLP by Rivest, Shamir and Wagner cannot be parallelized. However, the TLP needs to be set up by someone who knows the group order (or another trapdoor) and hence it needs to involve a trusted party that knows the result of the TLP ahead of time. To solve these problems, Mahmoody, Moran and Vadhan introduce the notion of proofs of sequential work (PoSWs). Those are proofs of work that are inherently sequential and, hence, the time it takes to solve them is (almost) independent of the number of processors available. Their construction uses depth-robust graphs and, as a consequence, requires space linear in T from the prover. Cohen and Pietrzak [CP18] improve upon their construction, giving a PoSW with better parameters, where the space used by the prover is only logarithmic in T. A problem that remained open in both of these works, however, is to construct a PoSW with unique output.

To address this, Boneh et al. [BBBF18] introduced verifiable delay functions (VDFs) in 2018. VDFs are functions that take a prescribed amount of T sequential steps to compute and can be verified in time much less than T. The function takes an input x together with a *time parameter* T. They have found a lot of applications including the design of blockchains [CP19], computational time-stamping [CE12, LSS20] and randomness beacons [Rab83, LW15, SJH+21]. A randomness beacon is a (pseudo-)random number constructed by n parties that don't trust each other and don't want any party to be able to bias the number. Combining the ideas of Lenstra and Wesolowski [LW15] with verifiable delay functions, one can obtain randomness beacons as follows: Every party i submits an arbitrary number r_i during a submission period. After this period, a VDF challenge is computed from $r_1, \ldots, r_n$ and the randomness beacon is set to be the hash of the output of the VDF. If the time parameter T is larger than the time of the submission period, it is impossible to predict the randomness value before the submission period has ended and hence no party can bias the outcome via their choice of r_i.

The first construction of a VDF was given in [BBBF18] based on incrementally-verifiable computation [Val08]. Loosely speaking, the authors used repeated (structured) hashing as their

communicate. Note that in the context of time lock puzzles, where T is in the order of 2^{30}, this would mean a massive amount of parallelism for a relatively small speed up.

delay function and then relied on incremental succinct non-interactive arguments (SNARGs) to enable efficient verifiability of the result of the repeated hashing. Unfortunately, such generic SNARGs are comparatively slow as they involve heavy cryptographic tools, in particular probabilisticly checkable proofs (PCPs).

Two much more practical VDFs were constructed concurrently by Pietrzak [Pie19] and by Wesolowski [Wes20]. In both cases the delay function is the repeated squaring function proposed by Rivest, Shamir and Wagner.[2] However, in the setting of VDFs no one knows the order of the group so there exists no party that can compute the result quickly. Instead, to achieve verifiability, Pietrzak and Wesolowski both construct what we call a *proof of exponentiation* (PoE). In a PoE in a group $\mathbb{G}$, a prover $\mathcal{P}$ tries to convince a verifier $\mathcal{V}$ that a tuple $(y, x, q, T) \in \mathbb{G}^2 \times \mathbb{N}^2$ satisfies $y = x^{q^T}$. In other words, a PoE is a proof or argument system for the language

$$L_{\mathbb{G}} := \left\{(y, x, q, T) \in \mathbb{G}^2 \times \mathbb{N}^2 : y = x^{q^T} \text{ over } \mathbb{G}\right\}. \tag{1.1}$$

We refer to q as the exponent and T as the time parameter. We have already seen that if the order of $\mathbb{G}$, denoted $\mathrm{ord}(\mathbb{G})$, is easily computable, then $\mathcal{V}$ can efficiently compute x^{q^T} on its own. Therefore, the language is non-trivial only if it is hard to compute $\mathrm{ord}(\mathbb{G})$. Such groups are called *groups of unknown order* or *hidden order groups*. The two types of groups of unknown order that are presently used are RSA groups [RSA78] and class groups [BW88]. PoEs have found applications as building blocks for not only constructing VDFs but also succinct non-interactive arguments of knowledge (SNARKs) [BHR+21]. In both applications, we want the cost incurred by $\mathcal{P}$ to compute the PoE to be marginal compared to the T exponentiations that are necessary to compute y in the first place.

A PoE needs to satisfy two properties: completeness and soundness. Completeness says that the verifier should accept the PoE whenever the statement $y \stackrel{?}{=} x^{q^T}$ is correct. Soundness says that whenever the statement $y \stackrel{?}{=} x^{q^T}$ is false, then no matter which strategy a malicious prover chooses, the verifier should reject the PoE with high probability.

Wesolowski's PoE [Wes20] is designed as follows: To prove the statement $y \stackrel{?}{=} x^{q^T}$, the verifier $\mathcal{V}$ samples a random prime ℓ from the set of the first 2^λ primes and sends it to $\mathcal{P}$. The prover $\mathcal{P}$ then computes integers s and $0 \leq r < \ell$ such that $q^T = s\ell + r$. Then it computes $\pi := x^s$ and sends it to $\mathcal{V}$. The verifier $\mathcal{V}$ computes $r = q^T \mod \ell$ and checks if $y = \pi^\ell x^r$ in $\mathbb{G}$. Completeness of the PoE follows immediately. Soundness is based on the *adaptive root assumption*, which states that the following game is hard: The adversary $\mathcal{A}$ outputs a group element $g \neq 1$. Then the challenger $\mathcal{C}$ gives a random prime ℓ from the first 2^λ primes to $\mathcal{A}$ and $\mathcal{A}$ needs to compute $g^{1/\ell}$.

Pietrzak's PoE [Pie19] to prove the statement $y \stackrel{?}{=} x^{q^T}$ is constructed in a different way. The prover $\mathcal{P}$ first sends the element $\mu = x^{2^{T/2}}$ to $\mathcal{V}$. Note that this implicitly splits up the initial statement into two smaller ones, namely $\mu \stackrel{?}{=} x^{q^{T/2}}$ and $y \stackrel{?}{=} \mu^{q^{T/2}}$. To fold the two smaller statements back into one, $\mathcal{V}$ samples a random challenge $c \in \{1, \ldots, 2^\lambda\}$ and sends it to $\mathcal{P}$. Then both $\mathcal{P}$ and $\mathcal{V}$ compute a new statement $y^c\mu \stackrel{?}{=} (\mu^c x)^{q^{T/2}}$ and proceed to prove this statement recursively in the same manner until they arrive at a statement of the form $\tilde{y} \stackrel{?}{=} \tilde{x}^q$ for some $\tilde{x}, \tilde{y} \in \mathbb{G}$. The verifier $\mathcal{V}$ can check this statement on its own by performing a single exponentiation in $\mathbb{G}$. Again completeness of the PoE follows by inspecting the protocol.

[2] VDFs have also been proposed in other algebraic settings, which we discuss in Section 1.5. However, the VDFs based on repeated squaring remain the only practical candidates.

Soundness is based on the *low order assumption*, which states that it is hard to find elements of low order in the given group and output it together with the order.

Boneh, Bünz and Fisch [BBF24] give a reduction from the adaptive root assumption to the low order assumption. That means that if the adaptive root assumption holds, then so does the low order assumption. A reduction in the other direction is not known. In fact, there exist groups where the low order assumption holds unconditionally: Pietrzak [Pie19] proposes to instantiate his VDF in groups that do not contain any low order subgroups. Such groups are constructed as follows: Choose two large *safe primes* p and q. That is, $(p-1)/2$ and $(q-1)/2$ are also prime numbers. Set $N = p \cdot q$. Then we have that the group of quadratic residues modulo N has no low order subgroups. Hence, the adaptive root assumption is a stronger assumption than the low order assumption.

Since the applications of PoEs are non-interactive in nature, a PoE itself must be non-interactive, i.e., we need to remove the interaction with $\mathcal{V}$ above. Since the protocols are *public-coin*, i.e., the messages of $\mathcal{V}$ are uniformly random numbers and $\mathcal{V}$ does not have any secrets, we can use the Fiat-Shamir transform [FS87]. This heuristic simply replaces a verifier message by a hash of the statement and the previous transcript, which $\mathcal{P}$ can compute itself.

In this thesis we consider both constructional aspects and applications of VDFs and PoEs. We will see how to construct practical statistically sound PoEs and how to batch PoEs efficiently. Then we build VDFs with special properties that are needed for the application to short-lived proofs, i.e., proofs that are deniable after a prescribed amount of time. Finally, we will see how to apply PoEs to the context of primality testing. Each of those topics is covered in one paper. We now give an overview about the motivation and results of each paper.

1.1 Statistically Sound Proofs of Exponentiation

The first paper of this thesis considers statistical soundness of PoEs [HHK+22].[3] A PoE is said to be statistically-sound if even a computationally-unbounded cheating prover $\tilde{\mathcal{P}}$ cannot convince $\mathcal{V}$ of a false statement $(\tilde{y}, x, q, T), \tilde{y} \neq x^{q^T}$. Such a strong soundness guarantee is required in some applications. In [BHR+21], where a PoE is used to construct a SNARK, the underlying PoE must be statistically-sound so that the statistical knowledge-soundness of the SNARK holds. In the context of VDFs, statistical soundness ensures some security *even when* the group order is somehow revealed, e.g., in the following use-case.

Chia is a secure, permissionless blockchain [CP19], built using VDFs and proofs of space [DFKP15]. In particular, it uses a PoE-based VDF instantiated in class groups, which is sampled freshly every 10 minutes. It relies on both security properties of this VDF, i.e., (i) *sequentiality*, which (loosely-speaking) requires it to be hard to compute the output $y := x^{q^T}$ in less than T steps; and (ii) *soundness*, which, as for PoE, requires it to be hard to generate proofs for a wrong output $\tilde{y} \neq x^{q^T}$. However, the reliance on these two security properties is to different degrees, as we explain next. An attacker that occasionally learns the group order and, therefore, is able to compute the output fast only has a short-term impact on the security. On the other hand, an attacker that breaks soundness is devastating since it potentially leads to double-spending.[4] Therefore, if the VDF used is statistically-sound then,

[3] This section is taken essentially from [HHK+22].

[4] A minor nuisance would be the need to roll back the blockchain once a flawed proof was added and recognized. But an attacker that can forge proofs controls the randomness and, thus, can do things like attaching a pre-computed chain to the current one in order to do a double-spending attack with only little resources.

even in the worst case where the attacker learns the group order,[5] it will only be able to compute the correct output fast but will still *not* be able to lie about its value.

In other scenarios, where the group order is *supposed* to be known by some parties, using statistically-sound PoEs allows for a much more efficient setup. The RandRunner protocol [SJH+21] uses VDF to construct randomness beacons [Rab83]. Every party participating in the protocol realizing the beacon will sample two safe primes, which defines a "safe" RSA group where Pietrzak's PoE is guaranteed to have statistical soundness. The fact that these parties know the factorization is actually a feature, as they are occasionally required to use it as a trapdoor to *quickly* compute and broadcast a VDF output and the PoE certifying its correctness. To prevent cheating, each party must provide a zero-knowledge proof that their modulus is indeed the product of two safe primes. If one, instead, uses PoEs with statistical soundness in *arbitrary* groups, the protocol can be instantiated in *any* RSA group. Thus, the expensive zero-knowledge proof can be avoided (at the cost of larger PoEs for the individual proofs).

A similar scenario comes up in the fair multiparty coin-flipping protocol of Freitag et al. [FKPS21]. This methodology might also be useful for (non-interactive) timed commitments and encryption [BN00, KLX20].

We have already seen above that Pietrzak's PoE is statistically sound in certain RSA groups. For the setting of class groups, on the other hand, our understanding of the low-order and adaptive root assumptions is still only developing: for example, [BKSW20] showed how to break the low-order assumption in class groups for some classes of prime numbers and these are, therefore, not suitable to instantiate [Pie19] and [Wes20]. However, class groups do have one major advantage over RSA groups in that they have a "transparent" set-up, i.e., the group can be sampled *obliviously* in the sense that a random string specifies a group *without* revealing the order of the group. Compare this with the RSA group, where the only known way to generate the group is to first sample primes p_1 and p_2 and then output the modulus $N = p_1 p_2$. But this means the sampler knows the factorization and, thus, the group order $(p_1 - 1)(p_2 - 1)$. For such groups with *trusted set-up* to be used in VDFs or SNARKs, one must either employ some trusted party to sample N and then delete p_1 and p_2, or sample N using expensive multiparty computation (see, e.g., [FLOP18, CCD+20] and the references therein). We would therefore like to have PoEs that are statistically sound in *any* group.

Block et al. [BHR+21] construct the first such PoE. Their construction is a clever parallel repetition of λ runs of Pietrzak's PoE, where λ is a statistical security parameter. Unfortunately, this repetition also increases the proof size from $\log T$ to $\lambda \cdot \log T$. Therefore, to achieve, say $\lambda = 80$ bits of security, the number of repetitions required, and hence the (multiplicative) overhead incurred in proof-size, is larger than 80 group elements, which is too large for practical applications.

In Chapter 3 we present a statistically sound PoE in any group for the case where the exponent q is the product of all primes up to some bound B. For such a structured exponent, we show that it suffices to run only $\lambda / \log(B)$ parallel instances of Pietrzak's PoE. This reduces the concrete proof-size compared to Block et al. by an order of magnitude. Furthermore, we show that in the known applications, where PoEs are used as a building block, such structured exponents q are viable. Finally, we also discuss batching of our PoE, showing that many proofs (for the same $\mathbb{G}$ and q but different x and T) can be batched by adding only a single element to the proof per additional statement. The results in Chapter 3 are published in [HHK+22].

[5] This can happen, e.g., if the trusted setup failed or the class group sampled turned out weak.

1.2 Batching PoEs

While the paper presented in the last section includes a batching protocol tailored to the PoE variants of Pietrzak's PoE, the second paper of the thesis constructs protocols to batch multiple proofs of exponentiation for *any* PoE in hidden order groups [HHI25].[6] In higher-level protocols, many PoEs are commonly generated, and there is a clear incentive to batch them to minimize the communication complexity as well as the computational overhead for the users. Consider, for example, the Chia Network [Inc23] blockchain that currently generates 32 VDF proofs in expectation every 10 minutes and has a block length of around 4.8 million blocks.[7] One possible application of batch PoEs for such blockchains is for efficient onboarding of new users with "light" clients. Specifically, the miners (or some third party) could regularly batch all the intermediate PoE statements and provide a single batch PoE. Thus, reducing simultaneously the storage overhead of the blockchain and clients' computation when joining the blockchain and having to verify all the VDFs.

We will also see that batch PoEs can be used towards *remote attestation of parallel computational power*. Consider a prover claiming it has a GPU with ten thousand cores. To prove it indeed has such parallel computational power, the verifier could generate ten thousand independent PoE-based VDF challenges, each with time parameter T, and ask the prover to solve all instances in time T. After solving all the instances and computing the corresponding proofs, the prover sends all the results and proofs to the verifier. If the prover can answer in time not much longer than T, and all the PoEs are accepting, the verifier would be convinced. The clear downside of this solution is that the communication complexity and verifier's running time grow linearly with the claimed parallelism of the prover; the prover sends ten thousand PoEs, and the verifier must verify all of them. To improve the communication complexity, it would be natural for the prover to batch the instances to a single one and send a single PoE for the batched instance. This would decrease the number of proofs but the prover still needs to send all the results and let the verifier check that the batched instance corresponds to them. Otherwise, the prover could simply batch only the challenges and solve just the batched challenge without the need for any parallelism.

Block et al. [BHR+21] and Rotem [Rot21] construct the first batching protocols for PoEs. The first protocol, introduced concurrently by Block et al. and Rotem, is constructed as follows: To batch n statements of the form $y_1 \stackrel{?}{=} x_1^{q^T}, \dots y_n \stackrel{?}{=} x_n^{q^T}$, the verifier $\mathcal{V}$ samples n bits $b_1, \dots, b_n$ uniformly at random and sends them to the prover $\mathcal{P}$. Then both $\mathcal{P}$ and $\mathcal{V}$ compute a new statement

$$\prod_{i\in[1,n]} y_i^{b_i} \stackrel{?}{=} \left(\prod_{i\in[1,n]} x_i^{b_i}\right)^{q^T}$$

and then $\mathcal{P}$ proves only this new statement with a PoE. Assume that at least one of the initial statements is incorrect. It can be shown that then the new statement is also incorrect with probability at least $1/2$. Hence, the batching procedure has soundness error $1/2$. This error can be amplified by either repeating the batching procedure with fresh random bits λ many times, for statistical security parameter λ, or by increasing the size of the random challenges sent by $\mathcal{V}$. This is the second batching protocol presented by Rotem. In this protocol the verifier sends n uniformly random numbers $r_1, \dots, r_n \leftarrow \{1, \dots, 2^\lambda\}$ to $\mathcal{P}$. Then $\mathcal{P}$ and $\mathcal{V}$

[6] This section is taken essentially from [HHI25].

[7] At the moment the group of the VDF also changes every 10 minutes in Chia. This is due to their usage of class groups, whose security properties are not yet well understood.

compute a new statement

$$\prod_{i\in[1,n]} y_i^{r_i} \stackrel{?}{=} (\prod_{i\in[1,n]} x_i^{r_i})^{q^T},$$

which is then proven by a PoE. Soundness of this protocol is based on the low order assumption. In groups that have no low order elements the soundness error is $1/2^\lambda$.

In Chapter 4 we introduce two batch PoEs that outperform both proposals of Rotem and we evaluate their practicality. First, we show that the two batch PoEs of Rotem can be combined to improve the overall efficiency by at least a factor of two. Second, we revisit the work of Bellare, Garay, and Rabin [BGR98] on batch verification of digital signatures and show that, under the low order assumption, their bucket test can be securely adapted to the setting of groups of unknown order. The resulting batch PoE quickly outperforms the state of the art in the expected number of group multiplications with the growing number of instances, and it decreases the cost of batching by an order of magnitude already for hundreds of thousands of instances. Importantly, it is the first batch PoE that significantly decreases both the proof size and complexity of verification. Our experimental evaluations show that even a non-optimized implementation achieves such improvements, which would match the demands of real-life systems requiring large-scale PoE processing. Chapter 4 is based on [HHI25].

1.3 Short-Lived Proofs from VDFs

The third paper of the thesis constructs VDFs with special properties that are needed for another application of VDFs [HP25].[8] This application was presented by Arun, Bonneau and Clark in [ABC22]. The authors construct so called *short-lived proofs* and *short-lived signatures* from VDFs. Short-lived proofs and signatures are only valid for a prescribed amount of time T. After time T they are easy to forge by anyone and hence validity cannot be verified anymore. The authors of [ABC22] achieve this notion for any relation $\mathcal{R}$ by combining a proof system for $\mathcal{R}$ with a VDF computation using a simple OR statement: A short-lived proof is correct if either the proof for $\mathcal{R}$ is correct or a VDF computation has been performed. This way the proof for $\mathcal{R}$ is only valid before time T has passed since after time T anyone can output a valid proof by proving that they have performed the VDF computation. One useful property that short-lived proofs and signatures can have is *reusable forgeability*, which means that one slow computation enables efficient proof forgery for many statements.

In their constructions Arun, Bonneau and Clark need two variants of VDFs: zero-knowledge VDFs and watermarkable VDFs. Zero-Knowledge VDFs are VDFs that can verify that a prover $\mathcal{P}$ knows the result y of a VDF without revealing any other information about y to the verifier $\mathcal{V}$, i.e., instead of sending the result y to $\mathcal{V}$, $\mathcal{P}$ and $\mathcal{V}$ engage in a *zero-knowledge proof of knowledge* of y. The zero-knowledge VDF in [ABC22] is a zero-knowledge version of Wesolowski's VDF. Using this VDF in the OR construction described above to obtain a short-lived proof provides some form of reusable forgeability: After performing one slow computation of the delay function, one can forge proofs for multiple statements of the same sender by providing a re-randomized VDF proof for each statement. Using the zero-knowledge VDF in [ABC22] computing a re-randomized proof takes time roughly $T/\log(T)$ since this is the time it takes compute Wesolowski's VDF proof. Since Pietrzak's VDF proof can be computed in time $T/\sqrt{T}$, a zero-knowledge version of Pietrzak's VDF would enable much faster forging times. The authors of [ABC22] leave a construction of a zero-knowledge version of Pietrzak's VDF as an open problem.

[8] This section is taken essentially from [HP25].

Watermarkable VDFs are VDFs in which the proof can be watermarked, i.e., tied to a specific prover. They were informally introduced by Wesolowski in [Wes20], where he claims that his VDF can be watermarked by including a unique identifier in the computation of the random challenge. The authors of [ABC22] point out that security of this scheme is not proven since the proof of Wesolowski's VDF reveals the value x^s for a large s which may speed up the computation of $y = x^{q^T}$ and hence the computation of a proof with a different watermark. They propose to watermark the proof of their zero-knowledge VDF construction by including a unique identifier in the computation of the random challenge of the proof. Since the protocol is zero-knowledge, the watermarked proof does not reveal any information that might help computing a proof with a different watermark. However, this also means that the value $y = x^{q^T}$ cannot be revealed, which might be relevant in other applications of watermarkable VDFs.

In Chapter 5 we present the first constructions that transform *any* PoE in hidden order groups into a watermarkable VDF and into a zero-knowledge VDF. We note that this gives the first practical watermarkable VDF with a security proof, and the zkVDF solves the open problem stated in [ABC22] asking for a zkVDF based on Pietrzak's VDF. Instantiating the watermarkable VDFs with Pietrzak's PoE and using it in the [ABC22] construction of short-lived proofs (without reusable forgeability) we get proofs with significantly faster forging times and under different (arguably weaker) assumptions than the construction in [ABC22]. The results of Chapter [HP25] are published in [HP25].

1.4 Primality Testing from PoEs

Another application of PoEs, which is presented in the fourth paper of the thesis, is certifying non-primality of Proth numbers [HHKP23].[9] Proth's theorem [Pro78] states that $P_{k,n} = k2^n + 1$ is prime if and only if, for a quadratic non-residue x modulo $P_{k,n}$, it holds that

$$x^{k2^{n-1}} \equiv -1 \bmod P_{k,n}. \tag{1.2}$$

To date, the largest-known Proth prime is $10223 \cdot 2^{31,172,165} + 1$ [Pri16]. Since n is of the order of magnitude 10^7 and the square-and-multiply algorithm is the fastest way currently known to carry out exponentiation, the test roughly requires 10^7 squarings modulo a 10^7-digit modulus. Unfortunately, performing this test does not yield an immediate witness that certifies the correctness of the result – in particular, if $P_{k,n}$ is composite, the test does not find a divisor of $P_{k,n}$.[10] Until very recently, the standard way for another party to independently validate the test result was by recomputing the result of Equation (6.1). In 2020, Pavel Atnashev demonstrated that PoEs might be applicable in the context of these specialized primality tests to avoid the costly second recomputation.[11] Since the primality test using Proth's theorem amounts to iterated exponentiation, it seems immediate that one would attempt to exploit PoEs also towards efficient verifiability in the context of primality tests for giant numbers. The idea is for the volunteer to use the (non-interactive) PoE to compute, alongside the result of the test, a proof that helps any other party verify the result.

[9] This section is taken essentially from [HHKP23].

[10] Note that some primality tests, like, e.g., Miller-Rabin [Mil76, Rab80], can be modified to (sometimes) yield factors in case the number being tested is not a prime.

[11] More details can be found in this thread of mersenneforum.org. An implementation due to Atnashev is available on GitHub. The idea of using PoEs for certifying giant primes has been discussed also by Mihai Preda in another thread in the same forum already in August 2019

However, we already know that PoEs were constructed for groups whose order is hard to compute. If one party knows the group order, they can (in many groups) construct *false* Pietrzak PoEs that lead a verifier to accept proofs for false statements. In the context of primality testing the underlying group is $\mathbb{Z}_{P_{k,n}}$, so the group order is known whenever $P_{k,n}$ is prime. While this does not speed up the computation of the primality test (since the modulus is larger than the exponent), it removes the soundness guarantee of the protocol.

In Chapter 6 we show how to construct a *sound* practical proof of non-primality for Proth numbers. In particular, we show how to adapt Pietrzak's PoE at a moderate additional cost to make it a cryptographically-sound certificate of non-primality. That is, a volunteer can, parallel to running the primality test for $P_{k,n}$, generate an efficiently verifiable proof at a little extra cost certifying that $P_{k,n}$ is not prime. The results of this chapter are published in [HHKP23].

1.5 Other Related Work

Other candidate VDFs. In [HHKK23] a VDF based on the hardness of computing Lucas sequences over an RSA modulus is constructed. It is shown that there exist a reduction from repeated squaring to Lucas sequences. No reduction in the other direction is known. There are several candidate VDFs with sequentiality not based on iterated squaring, such as the permutation-polynomial based construction [BBBF18], the isogenies-based constructions [DMPS19, CSHT21, Sha19, CSRHT22] and the constructions from lattice problems [LM23, CLM23]. The constructions in [DMPS19, CSHT21, Sha19] work in the algebraic setting of isogenies of elliptic curves. Although these constructions provide a certain form of quantum resistance, they are presently far from efficient. Freitag et al. [FPS22] constructed VDFs from any sequentially hard function and polynomial hardness of learning with errors, the first from standard assumptions. The works of Cini, Lai, and Malavolta [LM23, CLM23] constructed the first VDF from lattice-based assumptions and conjectured it to be post-quantum secure. Finally, some VDF candidates rely on "arithmetization friendly" symmetric primitives and practically efficient SNARKs [BBBF18, SB19, KMT22]. However, the sequentiality of SNARK-based VDFs is yet to be understood, as shown, e.g., by the recent analysis of MinRoot [LMPR23]. While some of the above construction possibly provide post quantum security, they are currently not as efficient as the VDFs built from iterated squaring.

Existence of VDFs and PoSWs. There is evidence that to construct VDFs over groups, the reliance on the group order being unknown is inherent [RSS20, MSW20], which lends even more importance to PoE protocols from the perspective of efficient VDFs. PoSWs can be constructed from random oracles [MMV11]. However, VDFs do not exist in the random oracle model [MSW20, GRY24]. Finally, we point out that existence of VDFs has implications in complexity theory, in particular to the existence of average-case hardness in complexity classes of total search problems such as **PPAD** [EFKP20, LV20, CHK+19, BCH+22].

Variants of VDFs. In addition to the basic VDFs, refined variants of VDFs have also been explored. For a "continuous" VDF [EFKP20], it should be possible (loosely speaking) to take a proof and iterate it to produce a proof for the next iteration of the delay function (instead of having to recompute the proof for the new value from scratch). A "tight" VDF [DGMV20] necessitates that the amount of work that is required to generate a proof to be "comparable" to that required to just compute the function.

Batch verification. The idea of using batching to reduce the amortized cost per operation has been explored for a host of cryptographic primitives such as, e.g., key agreement [BY93], signatures [MN96], and public-key encryption [Fia97]. The problem of batching the verification of multiple *exponentiations* in arbitrary groups (not necessary of unknown order) was studied in [BGR98]. They make a heavy use of the random subset and random exponents technique (as pointed out in [Rot21]). Building on [BGR98], Rotem [Rot21] explored batch-verification of VDFs.

Assumptions in hidden order groups. The low order assumption in RSA groups is analyzed in [SB20]. The authors give equivalence results for weaker forms of the low order assumption and the factoring assumption for a non-negligible portion of moduli. The low order assumption in class groups is analyzed in [BKSW20]. The authors show that it is broken for Mersenne primes and other special forms of prime numbers.

CHAPTER 2

Preliminaries

In this thesis, we let λ denote a security parameter. We use $[n] := \{1, \ldots, n\}$ to denote the set of all positive integers smaller than or equal to n.

2.1 Relations and Interactive Proofs

A *relation* $\mathcal{R} \subset \mathcal{X} \times \mathcal{W}$ is a set of pairs (x, w), where x is called the instance and w is called the witness. The set of all values x for which there exists a witness w such that $(x, w) \in \mathcal{R}$ is called the *language* $L_{\mathcal{R}}$ for $\mathcal{R}$.

Definition 1 (interactive proof)**.** For a function $\varepsilon : \mathbb{N} \to [0, 1]$, an *interactive proof for a relation* $\mathcal{R}$ is a pair of interacting PPT algorithms $(\mathcal{P}, \mathcal{V})$, called the *prover* and the *verifier*, where $\mathcal{P}$ takes as input a pair $(x, w) \in \mathcal{R}$ and $\mathcal{V}$ takes as input x. We require the algorithms to satisfy the following properties

- **Completeness:** For every $x \in L_{\mathcal{R}}$, if $\mathcal{V}$ interacts with $\mathcal{P}$ on the common instance x, then $\mathcal{V}$ accepts with probability 1.
- **Soundness:** For every $x \notin L_{\mathcal{R}}$ and every cheating prover strategy $\widetilde{\mathcal{P}}$, the acceptance probability of the verifier $\mathcal{V}$ when interacting with $\widetilde{\mathcal{P}}$ is less than $\varepsilon(|x|)$, where ε is called the *soundness error*.

Definition 2 (proof of knowledge)**.** A *proof of knowledge* is an interactive proof that is *knowledge sound*, i.e., there exists an efficient extractor $\mathcal{E}$ such that for every (potentially malicious) prover $\mathcal{P}^*$ that makes $\mathcal{V}$ accept proof π for instance x of bit-length n with probability δ, the extractor $\mathcal{E}$, which can interact with $\mathcal{P}^*$, outputs a witness w such that $(x, w) \in \mathcal{R}$ with probability at least $(\delta - \varepsilon)/\text{poly}(n)$, where poly is some positive polynomial and $\varepsilon \in [0, 1]$ is called the *soundness error*.

Sometimes the knowledge extractor $\mathcal{E}$ only manages to output a witness for a relation that slightly differs to the relation for which completeness holds. In this case we say that the proof of knowledge soundness is *not tight*. This can affect the choice of parameters and assumptions needed for soundness.

To prove knowledge soundness it is often easier to show that the protocol satisfies *special soundness*. It is well known that special soundness implies knowledge soundness for 3-round protocols.

Definition 3 (special soundness)**.** A 3-round protocol is called *special sound* if there exist a polynomial time extractor that on input an instance x and two accepting transcripts (a, c, z) and (a, c', z') with common first message a and $c \neq c'$ outputs a witness $w \in \mathcal{R}$.

In this work we consider special honest verifier zero-knowledge which is a special case of honest verifier zero-knowledge. Note that proving zero-knowledge against an honest verifier is sufficient for us because the protocols will be made non-interactive via the Fiat-Shamir transform.

Definition 4 (special honest verifier zero-knowledge proof)**.** A *special honest verifier zero-knowledge proof* is an interactive proof that is *zero knowledge* when the verifier behaves according to the protocol. That means there exists an efficient simulator $\mathcal{S}$ that, given instance $x \in L_{\mathcal{R}}$ and a uniformly random value c from the randomness space of the verifier, can output an accepting transcript for x with verifier's message c which is indistinguishable from a real transcript with an honest verifier.

It is well known that any constant round interactive proof in which the verifier messages only consist of random elements can be transformed into a non-interactive proof via the Fiat-Shamir heuristic [FS87] by deriving the verifier's messages via a suitable hash function. If the interactive proof is honest verifier zero-knowledge, the non-interactive version is fully *zero-knowledge* (i.e., no assumption on the behavior of the verifier is needed) in the *random oracle model*. In the random oracle model the hash function is modelled as a publicly available random function $\mathcal{O}$. The simulator $\mathcal{S}$ has *programmable* access to $\mathcal{O}$, which means that it can set the output of $\mathcal{O}$ to a value of its choice as long as the distribution of the output values is uniform.

Definition 5 (proof of exponentiation)**.** A *proof of exponentiation* (PoE) in a group $\mathbb{G}$ is an interactive proof for the language

$$L = \{(x, y, e) \in \mathbb{G}^2 \times \mathbb{N} \mid x^{2^e} = y\}.$$

Definition 6 (batch proof of exponentiation)**.** A *batch proof of exponentiation* for m statements in a group $\mathbb{G}$ is an interactive proof for the language

$$L = \{\{(x_i, y_i, e)\}_{i\in[m]} \in \{\mathbb{G}^2 \times \mathbb{N}\}^m \mid x_i^e = y_i \text{ for all } i \in [m]\}.$$

In the thesis, we sometimes use the term *batching protocol* to refer to a batch proof of exponentiation.

2.2 Verifiable Delay Functions

Verifiable Delay Functions were introduced by Boneh et al. in [BBBF18].

Definition 7. A *verifiable delay function* (VDF) is a set of algorithms (`Setup`, `Eval`, `Prove`, `Verify`), where

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ on input statistical security parameter 1^λ and time parameter T outputs public parameters $\mathbf{pp}$.

$\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ on input $(\mathbf{pp}, x, T)$ outputs (y, α), where α is an advice string.

$\texttt{Prove}(\mathbf{pp}, x, y, \alpha) \to (y, \pi)$ outputs a proof π for y.

$\texttt{Verify}(\mathbf{pp}, x, y, \pi) \to \texttt{accept/reject}$ checks that $y = \texttt{Eval}(\mathbf{pp}, x)$.

The algorithm $\texttt{Eval}$ is deterministic and can compute the output y in T sequential steps. A VDF must additionally satisfy three properties:

Completeness: For all tuples $(\mathbf{pp}, x, y, \pi)$, where $y = \texttt{Eval}(\mathbf{pp}, x)$ and $\pi = \texttt{Prove}(\mathbf{pp}, x, y, \alpha)$, algorithm $\texttt{Verify}(\mathbf{pp}, x, y, \pi)$ outputs $\texttt{accept}$.

Sequentiality: Any parallel algorithm that uses at most $\text{poly}(\lambda)$ processors and outputs $y = \texttt{Eval}(\mathbf{pp}, x)$ with noticeable probability runs in time at least T.

Soundness: If $\texttt{Verify}(\mathbf{pp}, x, y, \pi)$ outputs $\texttt{accept}$, then the probability that $y \neq \texttt{Eval}(\mathbf{pp}, x)$ is negligible.

2.3 The Group of Signed Quadratic Residues

One example of a hidden order group that has useful properties is the group of signed quadratic residues [FS00, HK09] with a safe prime modulus. We call a prime number p *safe* if $p = 2p' + 1$ for a prime number p'. We say that $N = pq$ is a *safe prime modulus*, if both p and q are safe primes. Let $\mathbb{Z}_N^*$ denote the multiplicative group modulo N. The group of quadratic residues modulo N is defined as $\text{QR}_N := \{a^2 \mod N : a \in \mathbb{Z}_N^*\}$ and the group of signed quadratic residues is defined as

$$\text{QR}_N^+ := \{\mid b \mid : b \in \text{QR}_N\},$$

where $\mid b \mid$ is the absolute value of b when representing the elements of $\mathbb{Z}_N$ as $\{-(N-1)/2, \ldots, (N-1)/2\}$. QR_N^+ is a cyclic group with group operation $a \circ b :=\mid a \cdot b \mod N \mid$. Unlike in QR_N, membership in QR_N^+ can be efficiently tested: We have that $b \in \text{QR}_N^+$ if $0 \leq b \leq (N-1)/2$ and the Jacobi symbol of b modulo N is $+1$.

2.4 Assumptions

In this thesis we need the following well-known assumptions in hidden-order groups. In Sections 4.5.2, 5.3 and 5.6.2 we state the novel assumption that we need in the papers of this thesis.

Definition 8 (strong RSA assumption)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *strong RSA assumption* holds for $\texttt{GGen}$ if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$ and an element $a \leftarrow \mathbb{G}$.

2. $\mathcal{A}$ outputs a pair $(e, b) \in \mathbb{Z} \times \mathbb{G}$.
3. $\mathcal{A}$ wins if and only if $e \neq 1$ and $b^e = a$.

The following assumption, which was first formalized in [BBF24], states that it is (computationally) hard to find elements of low order. Note that our assumption is a bit stronger than theirs because our upper bound on the order is $2^{3\lambda+2}$, while they assume the upper bound 2^λ. This is only necessary for the results in Chapter 5. For the other results of the thesis the bound 2^λ would be sufficient. There are groups in which this assumption holds information theoretically because such elements do not exist: The group of signed quadratic residues QR_N^+ described above.

Definition 9 (low order assumption)**.** Let $\mathtt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *low order assumption* holds for $\mathtt{GGen}$ if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\mathtt{GGen}(1^\lambda)$.
2. $\mathcal{A}$ outputs a pair $(d, a) \in [2^{3\lambda+2}] \times \mathbb{G}$.
3. $\mathcal{A}$ wins if and only if $a \neq 1$ and $a^d = 1$.

The following assumption was first stated by Rivest, Shamir and Wagner [RSW96].

Definition 10 (iterated squaring assumption)**.** Let $\mathtt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *iterated squaring assumption* holds for $\mathtt{GGen}$ if, for any probabilistic parallel algorithm $\mathcal{A}$ that uses at most $\mathrm{poly}(\lambda)$ processors and runs in time less than T, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\mathtt{GGen}(1^\lambda)$, a random group element x and an integer T.
2. $\mathcal{A}$ wins if it outputs element $y = x^{2^T}$.

Remark 1. We note that, strictly speaking, the iterated assumption as stated above does not hold. Bernstein and Sorenson [BS07] showed that one can reduce the *sequential* time of computing an iterated squaring instance x^{2^T} from T to $T/\log\log(T)$ using at least T^2 processors. While this is a nice theoretical result, it is not practical in our setting. In practice the time parameter T will be at most 2^{32}, so the algorithm by Bernstein and Sorenson can reduce the sequential time by at most a factor of 6, for which it would need at least $T^2 = 2^{64}$ processors. For simplicity we will ignore this $\log\log(T)$ factor in the rest of the thesis.

In the decisional version of the iterated squaring assumption we consider an adversary that gets as input a pair of group elements (x, y) and needs to decide whether or not $y = x^{2^T}$. The YES instances are pairs (x, x^{2^T}) for a uniformly random group element x. The NO instances are pairs (x, z^2) for uniformly random group elements x and z. Note that it is necessary to square the element z because x^{2^T} is a square and in RSA groups one can rule out that an element is a square, whenever its (efficiently computable) Jacobi symbol is -1. The assumption was first stated explicitly in [MT19] and analyzed in the GGM in [RS20].

Instance: $(x, y, T, \mathbb{G})$, where $x, y \in \mathbb{G}$ and $T \in \mathbb{N}$ is even

Parameters: statistical security parameter λ

Statement: $x^{2^T} = y$ in $\mathbb{G}$

Protocol:

1. $\mathcal{V}$ samples $\ell \leftarrow \text{Primes}(\lambda)$ uniformly at random and sends it to $\mathcal{P}$.
2. $\mathcal{P}$ computes $q \in \mathbb{Z}_{\geq 0}$ and $0 \leq r < \ell$ such that $2^T = q\ell + r$ and sends $\pi = x^q$ to $\mathcal{V}$.
3. $\mathcal{V}$ computes $r = 2^T \mod \ell$ and checks if $y = \pi^\ell x^r$ in $\mathbb{G}$. It outputs `accept` or `reject` accordingly.

Figure 2.1: Wesolowski's PoE [Wes20]. $\text{Primes}(\lambda)$ denotes the set of the first λ prime numbers.

Definition 11 (decisional iterated squaring assumption)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *decisional iterated squaring assumption* holds for $\texttt{GGen}$ if, for any probabilistic parallel algorithm $\mathcal{A}$ that uses at most $\text{poly}(\lambda)$ processors and runs in time less than T, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$, a random group element x, an integer T and a group element y which, with probability $1/2$ each, takes one of the following two forms: either $y = z^2$ for a uniformly random group element z or $y = x^{2^T}$.
2. $\mathcal{A}$ outputs 0 or 1 indicating whether or not $y = x^{2^T}$.
3. $\mathcal{A}$ wins if it outputs the correct bit with probability greater than $1/2$.

2.5 Known Proofs of Exponentiation and Batching Protocols

We present Wesolowski's PoE in Figure 2.1, Pietrzak's PoE in Figure 2.2 and Block et al.'s PoE in Figure 2.3. We present the Random Subsets Protocol by [BHR+21, Rot21] in Figure 2.4a and the Random Exponents Protocol by [Rot21] in Figure 2.4b. Both protocols are adaptations of the protocols from [BGR98] to the setting of hidden order groups. For ease of exposition, we present all protocols in their interactive form. Non-interactive PoEs and batch PoEs can be obtained via the Fiat-Shamir heuristic [FS87], i.e., by deriving the verifier's challenges from the statements and current transcript via a suitable hash function. We note that in the interactive version of Wesolowski's PoE the challenges space needs to be increased to $\text{Primes}(2\lambda)$. See [BBF24, Section 3.3] for an attack on the non-interactive version with smaller challenge space.

Instance: $(x, y, T, \mathbb{G})$, where $x, y \in \mathbb{G}$ and $T \in \mathbb{N}$ is even

Parameters: statistical security parameter λ

Statement: $x^{2^T} = y$ in $\mathbb{G}$

Protocol:

1. For $T = 1$:
   - If $x^2 = y$, output `accept`.
   - Else, output `reject`.
2. For $T > 1$:
   a) $\mathcal{P}$ sends $v = x^{2^{T/2}}$ to $\mathcal{V}$.
   b) If $v \notin \mathbb{G}$, $\mathcal{V}$ outputs `reject`. Otherwise, $\mathcal{V}$ samples $r \leftarrow \{0, 1, \ldots, 2^\lambda - 1\}$ uniformly at random and sends it to $\mathcal{P}$.
   c) $\mathcal{P}$ and $\mathcal{V}$ compute $x' := x^r v$ and $y' := v^r y$ in $\mathbb{G}$.
   d) If $T/2$ is even, $\mathcal{P}$ and $\mathcal{V}$ run the protocol on instance $(x', y', T/2, \mathbb{G})$. If $T/2$ is odd, $\mathcal{P}$ and $\mathcal{V}$ run the protocol on instance $(x', y'^2, (T+1)/2, \mathbb{G})$.

Figure 2.2: Pietrzak's PoE [Pie19].

Parameters: statistical security parameter λ

Instance: $\{(x_i, y_i, T, \mathbb{G})\}_{i \in [\lambda]}$, where $x_i, y_i \in \mathbb{G}$ for all $i \in [\lambda]$ and $T \in \mathbb{N}$ is even

Statement: $x_i^{2^T} = y_i$ in $\mathbb{G}$ for all $i \in [\lambda]$

Protocol:

1. For $T = 1$:
   - If $x_i^2 = y_i$ for all $i \in [\lambda]$, output `accept`.
   - Else, output `reject`.
2. For $T > 1$:
   a) $\mathcal{P}$ sends $v_i = x_i^{2^{T/2}}$ to $\mathcal{V}$ for all $i \in [\lambda]$.
   b) If for some $i \in [\lambda]$, it holds $v_i \notin \mathbb{G}$, then $\mathcal{V}$ outputs `reject`. Otherwise, $\mathcal{V}$ samples $S \leftarrow \{0,1\}^{2\lambda \times \lambda}$ uniformly at random and sends it to $\mathcal{P}$.
   c) For all $j \in [\lambda]$, $\mathcal{P}$ and $\mathcal{V}$ compute
   $$x'_j := \prod_{i \in [\lambda]} (x_i v_i)^{S_{i,j}} \text{ and } y'_j := \prod_{i \in [\lambda]} (v_i y_i)^{S_{i,j}} \text{ in } \mathbb{G}.$$
   d) If $T/2$ is even, $\mathcal{P}$ and $\mathcal{V}$ run the protocol on instance $\{(x'_i, y'_i, T/2, \mathbb{G})\}_{i \in [\lambda]}$.If $T/2$ is odd, $\mathcal{P}$ and $\mathcal{V}$ run the protocol on instance $\{(x'_i, y'^2_i, (T+1)/2, \mathbb{G})\}_{i \in [\lambda]}$.

Figure 2.3: Block et al.'s PoE [BHR+21].

Parameters: $e, m, \rho \in \mathbb{N}$, $\mathbb{G}$, and PoE
Statements: $\left\{y_i \stackrel{?}{=} x_i^e\right\}_{i\in[m]}$ in $\mathbb{G}$
Protocol:

1. $\mathcal{V}$ samples a matrix $B \leftarrow \{0,1\}^{\rho\times m}$ uniformly at random and sends it to $\mathcal{P}$.
2. $\mathcal{V}$ and $\mathcal{P}$ construct new statements $\left\{y'_i \stackrel{?}{=} (x'_i)^e\right\}_{i\in[\rho]}$, where
$$y'_i = \prod_{j\in[m]} y_j^{B_{i,j}}, \quad x'_i = \prod_{j\in[m]} x_j^{B_{i,j}}.$$
3. $\mathcal{V}$ and $\mathcal{P}$ run ρ many PoE on $\left\{y'_i \stackrel{?}{=} (x'_i)^e\right\}_{i\in[\rho]}$ in parallel.

(a) Random Subsets [BHR+21, Rot21].

Parameters: $e, m, \kappa \in \mathbb{N}$, $\mathbb{G}$, and PoE
Statements: $\left\{y_i \stackrel{?}{=} x_i^e\right\}_{i\in[m]}$ in $\mathbb{G}$
Protocol:

1. $\mathcal{V}$ samples a vector $r \leftarrow [2^\kappa]^m$ uniformly at random and sends it to $\mathcal{P}$.
2. $\mathcal{V}$ and $\mathcal{P}$ both construct one new statement $\tilde{y} \stackrel{?}{=} (\tilde{x})^e$, where
$$\tilde{y} = \prod_{i\in[m]} (y_i)^{r_i}, \quad \tilde{x} = \prod_{i\in[m]} (x_i)^{r_i}.$$
3. $\mathcal{V}$ and $\mathcal{P}$ run PoE on statement $\tilde{y} \stackrel{?}{=} (\tilde{x})^e$.

(b) Random Exponents [Wes20, Rot21].

Figure 2.4: Known batch PoEs.

CHAPTER 3

Statistically Sound Proofs of Exponentiation

3.1 Introduction

In this chapter we construct practical statistically sound proofs of exponentiation in any group. From the discussion in Section 1.1, one may conclude that the PoE of Block et al. [BHR+21] is the only reasonable option when we need statistical guarantees and want to avoid trusted set-ups. However, it suffers from the drawback that its proof-size is large, $\lambda \log T$ to be precise. In this work, we present an efficient PoE that enjoys statistical soundness in all groups, but only for exponent q that is of a special structured form – for our basic protocol, q is set as the product of all primes less than some *bound* $B \in \mathbb{N}$. The size of our proof is

$$\lambda \log T / \log B,$$

which is *smaller* than in [BHR+21] by a (multiplicative) factor of $\log B$. It is, however, not possible to choose B to be arbitrarily large in our protocol as this would adversely affect the verifier's (computational) complexity. An illustration of how the proof-size and verifier-complexity of our protocol change with B can be found in Figure 3.1. In our most basic protocol, the verifier's complexity when $B = 521$ is roughly the same as in [BHR+21] (Figure 3.1.(b)). For this B, we get the proof for each of the $\log T$ rounds down from $\lambda = 80$ to $9 = \lceil 80/\log B \rceil$ group elements (Figure 3.1.(a)). In practice, this means, e.g., that for a time parameter $T = 2^{32}$ and instantiation in a group with elements of size 2048 bits, the proof-size drops from 655KB to 74KB.

Finally, for the application to VDFs and SNARKs, we argue that our special choice of exponent q does not really matter. In the construction of SNARKs in [BHR+21], it is possible to use any q as long as it is sufficiently large.[1] As for VDFs, one typically just sets the exponent $q = 2$, and exponentiation, therefore, is tantamount to squaring. For a more general q, one adjusts the time parameter accordingly, as explained next. For an arbitrary q, one can use the square-and-multiply algorithm, so each exponentiation induces $\lfloor \log(q) \rfloor$ (not just one) sequential squarings with some multiplications in-between. Note that if q was a power of 2

[1] Note that our structured exponent q is always even. However, many results in [BHR+21] are stated only for odd values of q. In Appendix 3.5, we show that such restriction is not necessary and that their results, in fact, hold for all values q.

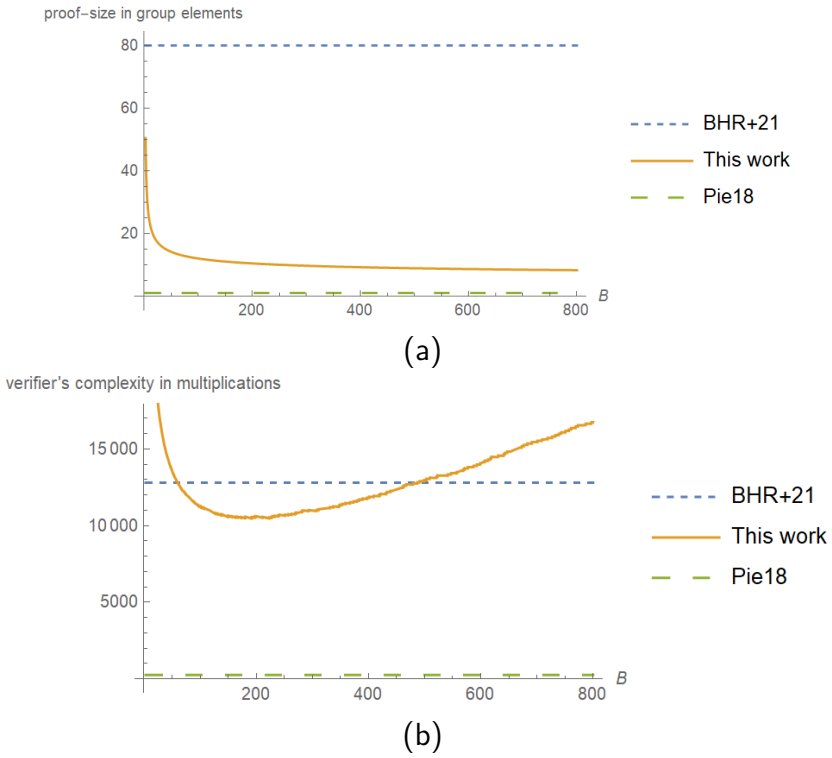


Figure 3.1: For 80-bit security, (a) the number of (group) elements sent by the prover *per round* and (b) the number of (group) multiplications carried out by verifier, also per round, plotted for different values of the bound B. The dotted blue line, the solid orange curve, and the dashed green line represent, respectively, [BHR+21], our protocol, and [Pie19]. In Figure 3.3 we dissect the solid orange curve in (b).

(which it is not in our case), say 2^k, the initial exponentiation would be of the form $x^{(2^k)^T}$, so one would set the time parameter to $T = T'/k$ in order to get a challenge that takes time T' to compute. Similarly, for our choice of q, one sets the time parameter to $T = \lceil T'/\log(q) \rceil$ to get a challenge that takes sequential time T' to compute.

3.1.1 Technical Overview

In this section, we first describe a basic version of our protocol where the verifier's running time is not optimal. This allows us to introduce the core ideas behind our PoE. Then we explain how to improve the verifier's efficiency. We refer the readers to Sections 3.2 and 3.4, respectively, for the technical details.

Basic Protocol and Proof Idea

Our starting point is Pietrzak's PoE, in particular, an observation on the fine-grained nature of its soundness which we exploit in our protocol. Therefore, we start with its high-level description.

Pietrzak's PoE and its soundness. The protocol in [Pie19] presented in Figure 2.2 is recursive in the time parameter T and involves $\log T$ rounds of interaction. To prove a (true) statement '$y = x^{q^T}$', the (honest) prover $\mathcal{P}$, in the first round, sends the "midpoint"

$\mu := x^{q^{T/2}}$ to the verifier $\mathcal{V}$. This results in two *intermediate* statements

$$\mu \stackrel{?}{=} x^{q^{T/2}} \text{ and } y \stackrel{?}{=} \mu^{q^{T/2}}, \tag{3.1}$$

but relative to half the original time parameter T. Next, $\mathcal{V}$ sends a random challenge r to $\mathcal{P}$, and they merge these two intermediate statements into a *new* statement

$$y' \stackrel{?}{=} (x')^{q^{T/2}}, \text{ where } x' := x^r \cdot \mu \text{ and } y' := \mu^r \cdot y.$$

The above steps constitute the "halving" sub-protocol, which is repeated $\log T$ times, halving the time parameter each time, until $\mathcal{P}$ and $\mathcal{V}$ arrive at a (base) statement for $T = 1$. At this point, $\mathcal{V}$ can efficiently check the correctness on its own by performing a single exponentiation.

To explain our observation, it suffices to focus on the first round of the protocol (but it applies also to later rounds). Assume that a cheating prover $\tilde{\mathcal{P}}$ tries to cheat with a (false) statement $\tilde{y} \stackrel{?}{=} x^{q^T}$ that is "α-false", by which we mean $\tilde{y} = y \cdot \alpha$ for some $\alpha \in \mathbb{G} \setminus \{\pm 1\}$ and $y := x^{q^T}$. The soundness of the halving sub-protocol depends on the order of α in $\mathbb{G}$, denoted $\mathrm{ord}(\alpha)$. For simplicity, let's restrict our attention throughout this section to the case where $\mathrm{ord}(\alpha)$ is a *prime power* p^e, for a prime $p \in \mathbb{N}$ and an exponent $e \in \mathbb{N}$ – the case of prime power already captures all interesting aspects. Our observation is that if $\mathrm{ord}(\alpha) = p^e$ for the original statement then the new statement is *still* false with probability $1 - 1/p^e$ (over the choice of r). In other words, the soundness error of this round is $1/p^e$. More generally, if $\mathrm{ord}(\alpha) = p^e$ for the original statement then for any $d \leq e$, $\mathcal{P}$ and $\mathcal{V}$ derive a new statement that is α'-false with probability $1 - 1/p^d$, where $\mathrm{ord}(\alpha') = p^{e'}$ for $e' \geq e - d$. We formalise this observation in Lemma 1.

Dealing with low-order elements by parallel repetition. Note that the above analysis also explains why Pietrzak's PoE is unsound when the group contains low-order elements: when, e.g., a statement is α-false for α such that $\mathrm{ord}(\alpha) = 2$, the soundness guaranteed is only $1/2$. To generate a cheating proof, it suffices to find such an α (see [BBF24] for details of the attack) and, hence, the necessity of low-order assumption. The PoE of Block et al. [BHR+21] gets around this issue essentially by direct amplification of the weak soundness, i.e., by running λ-many instances of the PoE *in parallel*, where λ is the security parameter. The protocol is presented in Figure 2.3. We provide an overview of their protocol next.

As in Pietrzak's PoE, the protocol in [BHR+21] is recursive in the time parameter T and involves $\log T$ rounds of interaction. At the start of the protocol, $\mathcal{P}$ and $\mathcal{V}$ have a tuple of λ (possibly identical) statements of the form $y \stackrel{?}{=} x^{q^T}$, with the same exponent q and time parameter T. In the first round, for each statement $y \stackrel{?}{=} x^{q^T}$ in the tuple, $\mathcal{P}$ sends the midpoint μ to $\mathcal{V}$, which results in two intermediate statements as in Equation (3.1). Altogether, there are 2λ intermediate statements at this point. Next, $\mathcal{V}$ sends a random challenge $(S_1, \ldots, S_\lambda)$ to $\mathcal{P}$, using which the intermediate statements are randomly merged into λ *new* statements, each of the form $y \stackrel{?}{=} x^{q^{T/2}}$. To be precise, the challenge S_i determines a subset of the intermediate statements, which are multiplied to obtain the i-th new statement. The above steps constitute the halving sub-protocol for [BHR+21] repeated $\log T$ times, halving the time parameter T each time, until $\mathcal{P}$ and $\mathcal{V}$ arrive at a statement for $T = 1$. At this point, $\mathcal{V}$ can efficiently check the correctness on its own by performing λ exponentiations.

To see why elements of low order do not affect soundness of the halving sub-protocol any more, suppose that one of the original statements is α-false with, say, $\mathrm{ord}(\alpha) = 2$ (which

is the worst case). Just like in [Pie19], we now have the guarantee that *at least* one of the 2λ intermediate statements is also false. [BHR+21] show that merging using the "random subset product" technique described above guarantees that each new statement is individually false with probability at least $1/2$ and, by independence of choice of S_is, at least one of the new statements is false with probability $1 - 2^{-\lambda}$. Thus, a false statement is in some sense "propagated" till the end, at which point it gets detected by $\mathcal{V}$.

Reducing the number of repetitions using structured exponents. Our basic protocol is similar to [BHR+21], except for two modifications: (i) we set q to be the product of all primes (strictly) less than some bound B; and (ii) we repeat only $\rho = \lambda/\log B$ times in parallel. To see why such a structured q helps reduce the number of repetitions from λ to ρ (while maintaining statistical soundness), we have to appeal to our above observation on the security of the PoE from [Pie19] and apply it to the setting of [BHR+21]. Suppose, again, that one of the original λ statements at the start of the protocol is α-false. There are three cases to be considered.

1. If $\mathrm{ord}(\alpha)$ has a "large" prime divisor $p > B$ then we apply our observation to infer that each new statement is α-false with probability $(1 - 1/p)$, and the independence of merging now implies a soundness error $p^{-\rho} \leq B^{-\rho}$. Since $\rho = \lambda/\log B$, we get soundness error $2^{-\lambda}$ as [BHR+21].

2. Otherwise, $\mathrm{ord}(\alpha)$ only has a "small" prime divisor $p < B$ and suppose that $\mathrm{ord}(\alpha) = p^e$.

   a) If the exponent e of the prime power p^e is "large" – to be precise $e > C$, where $C := \log T \log B$ – then we can apply our observation again. There are $\log T$ rounds and, by an averaging argument, there must exist a round i such that the prime power drops by at least $\log B$. However, by our observation, even for $p = 2$ this can only happen with probability at most $2^{-\log B} = 1/B$. Now, it can be similarly argued that the probability that there exists at least one false statement in the next round is $1 - B^{-\rho}$, and we get the same soundness error as in Item 1.

   b) Otherwise, the exponent e of the prime power p^e is "small", i.e., $e \leq C$. To handle this case, we modify the protocol so that the statement that (honest) $\mathcal{P}$ and $\mathcal{V}$ start with is
   $$y^{q^{-C}} \stackrel{?}{=} x^{q^{T-C}}$$
   instead of $y \stackrel{?}{=} x^{q^T}$, and then make $\mathcal{V}$ compute the final exponentiations $y^{q^{-C}} \to y$ *on its own*. To see why this helps with soundness, assume that $\tilde{\mathcal{P}}$ tries to cheat with an α-false statement $\tilde{y} \stackrel{?}{=} x^{q^{T-C}}$ in the modified protocol, i.e., $\tilde{y} = x^{q^{T-C}} \cdot \alpha$. Since $\mathrm{ord}(\alpha) = p^e$ divides q^C, we have $\alpha^{q^C} = 1$ and, therefore, $\mathcal{V}$'s final exponentiation leads to outright rejection:
   $$\tilde{y}^{q^C} = (x^{q^{T-C}} \cdot \alpha)^{q^C} = x^{q^T} \cdot \alpha^{q^C} = x^{q^T} \neq y.$$

To summarise our approach, the modification in Item 2b forces a cheating prover to cheat with α-false statements that are "far from being true" in the sense that $\mathrm{ord}(\alpha)$ has a divisor that is either a large prime or a prime power with large exponent. Moreover, it is possible to catch cheating with such statements building on existing techniques (Items 1 and 2a), aided by a fine-grained analysis. In Section 3.2, we extend the above analysis to accommodate α of arbitrary order (Theorem 1).

Improving Verifier's Complexity

The basic protocol that we just outlined decreases the number of parallel repetitions and, thus, the proof-size in the non-interactive case, by a factor $\log B$. But the verifier has to carry out some extra work as it must compute the final exponentiation $y^{q^{-C}} \to y$ on its own. This can be quite expensive, especially if we batch many proofs together. In the same group and for the same T, both protocols of Pietrzak and Block et al. can handle many PoEs basically at the price of a single PoE plus a small additive complexity overhead for each proof (this is, in fact, exploited in the SNARKs from [BHR+21]). In this work, we show that such batching works even for different values of T. Though, one problem for our new PoE is that, while this batching works also for the first phase of our protocol, the final exponentiation of the verifier cannot be trivially batched and, thus, it must be performed for each statement individually.

We thus further improve the protocol in two ways getting mostly rid of the extra cost for the final exponentiation. The first improvement leverages the observation that, by setting q to be not just the product of all primes (strictly) less than B but taking each prime p with power $\log B/\log(p)$, we can already decrease the exponent C for the final exponentiation from $\log T \log B$ to $\log T$. The second improvement comes from the observation that the final exponentiation $y^{q^{-C}} \to y$ can be replaced by just another PoE and, using our batching, this statement itself can be just batched together with the original statement. As the exponent ($C = \log T$ with the first improvement) is much smaller than T, the final exponentiation now only needs $\log(C) = \log\log T$ rounds. Iterating this idea $\log^*(T)$ times, which is at most

$$5 = \log^*(2^{2^{2^{2^2}}}) = \log^*(2^{65536})$$

in practice, we get the number of exponentiations down to 1 with a modest increase (from $\rho \cdot \log T$ to $\rho \cdot (\log T + \log^*(T))$ group elements) in proof-size. This batching argument only works so conveniently for T of a special form, basically powers of 2: T in the (relevant) range $2^{17} < T < 2^{65536}$ should be of the form $T = 2^t + 2^{16} + 2^4 + 2^2 + 1$. For general T the verifier's cost grows with basically the Hamming weight of $\log T$. In Appendix 3.5.3 we analyse the gain in efficiency of the polynomial commitment in [BHR+21] when we use this improved version of our PoE as a building block instead of the PoE proposed in [BHR+21].

3.2 Basic Protocol

Block et al. [BHR+21] constructed a statistically-sound PoE in any group of unknown order using the PoE from [Pie19] as starting point (which was described in Section 3.1.1). To achieve λ bits of security, their construction requires a multiplicative factor of λ in proof-size compared to [Pie19]. Below, we first explain the PoE from [BHR+21] in a bit more detail (than in Section 3.1.1), and then we explain how our protocol reduces this overhead. For now we just focus on improving the proof-size, but the verifier complexity of our protocol will increase, especially in settings where we batch many proofs – later, in Section 3.4, we will show how to get down the verifier's complexity.

Statistical PoE from [BHR+21]. To interactively prove the statement $y \stackrel{?}{=} x^{q^T}$, the prover $\mathcal{P}$ and verifier $\mathcal{V}$ first make λ copies of the statement.[2] In every round of the protocol,

[2]Note that the protocol works also when the starting statements are different *as long as* the exponent q and time parameter T match. This is the case for our protocol too.

the original statements are reduced to "smaller" statements by reducing the exponent q^{T_i} to $q^{T_{i+1}} := q^{T_i/2}$ as follows. The i-th round starts with a set of λ statements

$$\{y_i \stackrel{?}{=} x_i^{q^{T_i}}\}_{i\in[1,\lambda]}.$$

Then, $\mathcal{P}$ sends λ many "midpoints"

$$\{\mu_i := x_i^{q^{T_i/2}}\}_{i\in[1,\lambda]}$$

resulting in 2λ intermediate statements of the form

$$\{v_i \stackrel{?}{=} u_i^{q^{T_i/2}}\}_{i\in[1,2\lambda]}.$$

To avoid a blow-up in the number of statements, $\mathcal{V}$ sends a random subset $S \subseteq [1, 2\lambda]$ to $\mathcal{P}$ and, then, $\mathcal{P}$ and $\mathcal{V}$ use S to recombine these 2λ intermediate statements into one using subset-product:

$$\Pi_{i\in S} v_i \stackrel{?}{=} \Pi_{i\in S} u_i^{q^{T_i/2}}.$$

To ensure soundness, it is required to perform λ many of such recombinations using independent subsets $S_1, \ldots, S_\lambda$, and the round ends with λ many new smaller statements. We next explain why the recombination step must be performed λ many times. Suppose only one of the 2λ intermediate statements is false before the recombination step (which is the worst case). Then, with probability $1/2$, the false statement is not among the selected statements in the random subset used during the recombination step, and the resulting new statement is true. If all new statements are true, then the verifier falsely outputs accept at the end of the protocol and, therefore, the verifier must perform λ many independent recombinations to ensure λ bit security.

Our protocol. We give a formal description of our protocol in Figure 3.2 – for ease of presentation, we assume that $T = 2^t + C$ for some $t \in \mathbb{N}$.[3] Moreover, since the exponent q is fixed throughout the protocol, we sometimes drop it and use the short-hand (x, y, T) to denote the statement $y \stackrel{?}{=} x^{q^T}$. Its soundness is then proved in Section 3.2.1. Below, we give a high-level overview, slightly more detailed than in Section 3.1.1. We start by listing the major differences between our protocol and [BHR+21].

1. Instead of sampling a subset $S \subseteq [1, 2\lambda]$ to construct a new statement using subset-product, we take each intermediate statement to a random exponent in $\{0, 1, \ldots, 2^\kappa - 1\}$, where κ is some small integer, and then multiply them together: see Equation (3.3).

2. We set
$$q := \prod_{\text{prime } p<B} p, \tag{3.2}$$
where B is some fixed *bound*, which can be chosen depending on the application of the PoE.

3. We define a *constant* C such that the prover gives a proof for the statement $y' \stackrel{?}{=} x^{q^{T-C}}$ (i.e., a q^C-th root of the original statement) and the verifier computes the final check $(y')^{q^C} = y$ itself.

[3] The case where $T - C$ is not a power of 2 can be handled by a standard approach similar to [Pie19, Section 3.1].

The above changes allow us to reduce the number of repetitions from λ to $\rho := \lambda / \log B$ (for λ bits security). At a first glance, it could seem like the first change is sufficient to avoid the need for λ independent recombinations since the probability that a false statement is part of a new statement is not $1/2$ any more but seemingly $1/2^\kappa$. Unfortunately, it is not the case that taking κ-bit exponents for the recombination step achieves such a drastic improvement in the bound on the probability of accepting a false statement. Note that the process of raising a false statement to some exponent can also result in a true statement. This is indeed very likely if a false statement $y \stackrel{?}{=} x^{q^T}$ is "close" to the true one in the sense that y is the correct value multiplied by a low-order element α. If, e.g., this element α is of order two and the statement is raised to an even exponent, say two, the resulting statement $(y\alpha)^2 \stackrel{?}{=} (x^{q^T})^2$ will be true. This observation underlies an attack on [Pie19] that was first described[4] in [BBF24] and it is also the reason why [Pie19] is statistically-sound only in groups that have no elements of small order.

To circumvent the above attack using low-order elements, we introduce the second and third change in the protocol: instead of the original statement $y \stackrel{?}{=} x^{q^T}$, the (honest) prover only proves the (smaller) modified statement $y' \stackrel{?}{=} x^{q^{T-C}}$, where $y' := x^{q^{T-C}}$, and the verifier checks whether $y = (y')^{q^C}$ *by itself* as the final step. Moreover, to ensure that all the low orders are covered, we define q to be the product of all small prime numbers up to a certain bound B as in Equation (3.2). Now, a cheating prover that tries to cheat on an original statement by proving a false modified statement[5] *will* get caught in the final exponentiation *as long as* the false modified statement is "close" to the true one, where "close" means that the correct value can be multiplied by an element α whose order only has small prime divisors (prime numbers less than B) and the prime divisors have small exponents (integers up to C). To see this, observe that if the modified statement is $y'\alpha \stackrel{?}{=} x^{q^{T-C}}$ (which is false), the final exponentiation with q^C leads to rejection since

$$(\alpha y')^{q^C} = 1 \cdot (x^{q^{T-C}})^{q^C} = x^{q^T} \neq y,$$

where $\alpha^{q^C} = 1$ holds in $\mathbb{G}$ because of our assumption that it has low order. The above changes allow us to restrict to cheating provers that try to convince the verifier of statements that are "far from true", i.e., where the correct value is multiplied by an element whose order either has a large prime divisor or a divisor which is a small prime number with a large exponent. However, in this case the probability that the protocol ends with only true statements and the verifier wrongly accepts at the end of the protocol is less than $\log T \cdot 2^{-\lambda}$ for parameters $C = \log T \log B$ and $\rho = \lambda / \log B$, where ρ takes the role of λ in [BHR+21], i.e., it is basically the number of parallel repetitions of Pietrzak's protocol.

3.2.1 Soundness

We show that our protocol is statistically-sound for arbitrary groups of unknown order. In particular, soundness holds against cheating provers that can compute group elements of small order.

[4] The observation that random batching can be attacked using low-order elements was already made in [BP00].

[5] If the (cheating) prover does not cheat on the modified statement, the verifier will anyway catch it during the final exponentiation.

Parameters: (determined in the analysis)

1. bound $B \in \mathbb{N}$, which defines the exponent $q := \prod_{\text{prime } p<B} p$
2. constant for exponentiation $C \in \mathbb{N}$
3. number of parallel repetitions $\rho \in \mathbb{N}$
4. size of individual random coin $\kappa \in \mathbb{N}$

Statement: $y \stackrel{?}{=} x^{q^T}$ in $L_{\mathbb{G}}$

Protocol: For ease of presentation, we assume that $T = 2^t + C$. The protocol consists of t rounds described in Item 3 below.

1. The prover $\mathcal{P}$ sends $y' = x^{q^{T-C}}$ to the verifier $\mathcal{V}$, defining the initial ρ statements $\{(x_{0,j}, y_{0,j}, T_0)\}_{j\in[1,\rho]}$, where $T_0 := T - C$ and, for $j \in [1,\rho]$, $x_{0,j} := x$ and $y_{0,j} := y'$.
2. In round $i \in [1,t]$, $\mathcal{P}$ and $\mathcal{V}$ engage in the following halving sub-protocol:
   a) Let $\{(x_{i-1,j}, y_{i-1,j}, T_{i-1} = 2^{t-i+1})\}_{j\in[1,\rho]}$ be the statement from round $i-1$.
   b) $\mathcal{P}$ sends $\mathcal{V}$ the midpoints $\left\{\mu_{i,j} := x_{i-1,j}^{q^{T_{i-1}/2}}\right\}_{j\in[1,\rho]}$, which defines 2ρ intermediate statements
   $$\{(x_{i-1,j}, \mu_{i,j}, T_i := T_{i-1}/2)\}_{j\in[1,\rho]} \quad \text{and} \quad \{(\mu_{i,j}, y_{i-1,j}, T_i)\}_{j\in[1,\rho]},$$
   which we denote $\{(u_{i,k}, v_{i,k}, T_i)\}_{k\in[1,2\rho]}$.
   c) $\mathcal{V}$ sends a random challenge $\{r_{i,j,k}\}_{j\in[1,\rho],k\in[1,2\rho]}$ to $\mathcal{P}$, where $r_{i,j,k} \leftarrow \{0,1\}^\kappa$ independently for all $j \in [1,\rho]$ and $k \in [1,2\rho]$.
   d) $\mathcal{P}$ and $\mathcal{V}$ set $\{(x_{i,j}, y_{i,j}, T_i)\}_{j\in[1,\rho]}$ as the statement for the next round, where
   $$x_{i,j} := \prod_{k\in[1,2\rho]} u_{i,k}^{r_{i,j,k}} \quad \text{and} \quad y_{i,j} := \prod_{k\in[1,2\rho]} v_{i,k}^{r_{i,j,k}}, \tag{3.3}$$
   and proceed to the next round.
3. $\mathcal{V}$ accepts if and only if $x_{t,j}^q = y_{t,j}$ and $(y')^{q^C} = y$ for all $j \in [1,\rho]$.

Figure 3.2: Our basic Proof of Exponentiation.

Theorem 1. *Let B be any prime number such that $q := \prod_{\text{prime } p<B} p$ and $\rho \in \mathbb{N}$ be the number of repetitions per round. If we set $C = \log T \log B$ and let $\kappa \to \infty$, the verifier $\mathcal{V}$ will output* `accept` *on a false statement $(x, y, T = 2^t + C)$ with probability at most t/B^ρ.*

A parameter of our PoE is the bit-size κ of each random element sampled by the verifier. In the statement of Theorem 1, we consider the limit case with κ approaching infinity for the sake of readability. Note that if r is sampled from a randomness space of size 2^κ we have $\Pr[p \text{ divides } r] = 1/p + 1/2^\kappa$. In the limit case $\kappa \to \infty$, the probability is $1/p$. In practice, κ needs to be chosen carefully such that the protocol is still efficient but the probability of the above event is close enough to $1/p$. We discuss this point further in Section 3.2.2. Before proving Theorem 1, we analyse in Lemma 1 how the order of a group element precisely affects soundness; next we provide an overview.

Fine-grained soundness. Let $x^{q^{T-C}} = y'$ but a cheating prover $\tilde{\mathcal{P}}$ claims that $x^{q^{T-C}} = y'\alpha$. In the execution of the protocol, $\tilde{\mathcal{P}}$ first sends a midpoint μ, which results in two intermediate statements. Note that no matter what the value of μ is, one of the two statements will be false, so for now let's assume that $\tilde{\mathcal{P}}$ sends a correct midpoint $\mu = x^{q^{(T-C)/2}}$. Therefore the intermediate statements are

$$\mu \stackrel{?}{=} x^{q^{(T-C)/2}}, \quad \text{and} \quad y'\alpha \stackrel{?}{=} \mu^{q^{(T-C)/2}},$$

and in particular, the second statement is α-wrong. In the protocol, we copy each statement ρ many times, raise each copy to a random exponent r_k and then multiply the 2ρ statements together. This results in a new statement that is true whenever

$$\alpha^{r_1}\alpha^{r_2}\dots\alpha^{r_\rho} = \alpha^{r_1+r_2+\dots+r_\rho} = 1.$$

This is the case when $r_1 + r_2 + \dots + r_\rho \equiv 0 \mod \operatorname{ord}(\alpha)$, which happens with probability $1/\operatorname{ord}(\alpha)$ if we assume that the randomness space is large enough (see Section 3.2.2 for discussion on the size of the randomness). This means that whenever $\operatorname{ord}(\alpha)$ is large, it is unlikely that the statement is transformed into a true statement after a single round. However, the order of the element that makes the statement false can also decrease round by round until the statement is transformed into a true one. To prove this intuition, we use the following well-known fact about the order of group elements. A proof can be found in any standard textbook on group theory (e.g., [DF03, Proposition 5]).

Proposition 1. *Let $\mathbb{G}$ be a group, $\alpha \in \mathbb{G}$ a group element and m a positive integer. It holds that*

$$\operatorname{ord}(\alpha^m) = \frac{\operatorname{ord}(\alpha)}{\gcd(\operatorname{ord}(\alpha), m)}.$$

By Proposition 1 we get that $\operatorname{ord}(\alpha^{r_1+r_2+\dots+r_\rho}) < \operatorname{ord}(\alpha)$ whenever $r_1 + r_2 + \dots + r_\rho \equiv 0 \mod d$, where d is a divisor of $\operatorname{ord}(\alpha)$. If the order decreases in all of the ρ many new statements obtained this way, a cheating prover has a better chance to end up with a true statement in one of the following rounds. We want to bound the probability that after some round of the protocol all of the statements are true. To this end, we need the following lemma which bounds the probability that recombining a set of $m > \rho$ statements, where at least one statement is false, gives ρ true statements. In the proof of Theorem 1 we always have $m = 2\rho$. Later in Section 3.4 we show how to prove many statements simultaneously so we will use the lemma with different values m.

Lemma 1. *Let $\{(x_i, y_i, T)\}_{i\in[1,m]}$ be a set of m statements such that at least one of the statements is α-false for some $\alpha \in \mathbb{G}$. Let $\{(\tilde{x}_j, \tilde{y}_j, T)\}_{j\in[1,\rho]}$ be a set of ρ statements defined as*

$$\tilde{x}_j := \prod_{i\in[1,m]} x_i^{r_{j,i}} \quad \textit{and} \quad \tilde{y}_j := \prod_{i\in[1,m]} y_i^{r_{j,i}}$$

with independently sampled $r_{j,i} \leftarrow \mathbb{Z}_{2^\kappa}$ uniformly at random for all $i \in [1, m]$ and $j \in [1, \rho]$. Let B be any prime number. If we let $\kappa \to \infty$, the new statements satisfy the following properties with probability at least $1 - (1/B)^\rho$:

1. *If for some prime $p \geq B$ we have $p \mid \mathrm{ord}(\alpha)$, at least one of the statements $\{(\tilde{x}_j, \tilde{y}_j, T)\}_{j\in[1,\rho]}$ is $\tilde{\alpha}$-false and $p \mid \mathrm{ord}(\tilde{\alpha})$.*
2. *If for some prime $p < B$ and some integer $e \geq \log B$ we have $p^e \mid \mathrm{ord}(\alpha)$, at least one of the statements $\{(\tilde{x}_j, \tilde{y}_j, T)\}_{j\in[1,\rho]}$ is $\tilde{\alpha}$-false and $p^{e-\log B+1} \mid \mathrm{ord}(\tilde{\alpha})$.*

Proof. Since we want to lower bound the probabilities of the above events, it is sufficient to consider the case where $\mathrm{ord}(\alpha)$ has a single prime divisor. So, we assume $\mathrm{ord}(\alpha) = p^e$ for some prime p and integer e. Using α, we can express the statements $\{(x_i, y_i, T)\}_{i\in[1,m]}$ equivalently in the form $\{(x_i, h_i\alpha^{a_i}, T)\}_{i\in[1,m]}$, where $x_i^{q^T} = h_i$ are the correct values for all $i \in [1, m]$, $a_i \in \mathbb{Z}$ and at least one of the $a_i = 1$. A new statement $(\tilde{x_j}, \tilde{y_j}, T)$ is computed as

$$\tilde{x}_j := \prod_{i\in[1,m]} x_i^{r_{j,i}} \text{ and } \tilde{y_j} := \prod_{i\in[1,m]} (h_i\alpha^{a_i})^{r_{j,i}}.$$

Let $\tilde{\alpha} := \prod_{i\in[1,m]} \alpha^{a_i \cdot r_{j,i}}$. By Proposition 1, the order of $\tilde{\alpha}$ is

$$\frac{p^e}{\gcd(p^e, \sum_{i=1}^m a_i r_{j,i})} = p^{e-s}$$

for some $s \in \{0, 1, \ldots, e\}$. The probability that $s \geq k$ for any $k \in \{0, 1, \ldots, e\}$ is

$$\Pr[s \geq k] = \Pr\left[\sum_{i=1}^m a_i r_{j,i} \equiv 0 \mod p^k\right] = \frac{1}{p^k}.$$

To prove the first claim of the lemma, we set $e = 1$ and $p = B$: the probability that the new statement is true is the probability that $s = 1$, which is $1/B$ and, therefore, the probability that all of the ρ new statements are true is $1/B^\rho$.

We prove the second claim of the lemma by setting $e \geq \log B$ and observing that the probability of $s \geq \log B$ is $1/p^{\log B} \leq 1/2^{\log B} = 1/B$ – the probability that this is the case for all ρ statements is at most $1/B^\rho$. □

Proof of Theorem 1. Assume that the correct value in Step 3 of the protocol is $y' := x^{q^{T-C}}$ but a cheating prover $\tilde{\mathcal{P}}$ claims that $y'\alpha \stackrel{?}{=} x^{q^{T-C}}$ (i.e., makes a statement that is α-false). Notice that in the case where $\mathrm{ord}(\alpha) \mid q^C$ we have that $(y'\alpha)^{q^C} = (y')^{q^C} = y$ and, hence, the verifier $\mathcal{V}$ ends up rejecting after Step 4 of the protocol. It follows that if $\tilde{\mathcal{P}}$ wants to convince $\mathcal{V}$ that the result is not y, then it needs to choose an element α such that $\mathrm{ord}(\alpha)$ does not divide q^C. In this case, $\tilde{\mathcal{P}}$ wins if all of the ρ statements are true after t rounds of the protocol. From the discussion above, we know that the best option for $\tilde{\mathcal{P}}$ is either picking an element of order 2^{C+1} or an element of order p, where p is the smallest prime not dividing q^C. We analyse the two cases separately.

Case 1: Let $\mathrm{ord}(\alpha) = p$. Assume that in round i of the protocol we have ρ many statements

$$\{(x_{i-1,j}, y_{i-1,j}\alpha^{a_{i-1,j}}, T_{i-1})\}_{j\in[1,\rho]} \tag{3.4}$$

where $a_{i-1,j} \in \mathbb{Z}$ for all $j \in [1,\rho]$. If $a_{i-1,j} \equiv 0 \mod p$, the statement is true. Otherwise it is false and, by Proposition 1 and the primality of p, we know that $\alpha^{a_{i-1,j}}$ has order p. We assume that at least one of the $a_{i-1,j}$ is not divisible by p and we bound the probability that all of the statements are true in round $i+1$.

In Step 3 of the protocol, $\tilde{\mathcal{P}}$ sends midpoints $\mu_{i,j}$ which results in 2ρ statements

$$\{(x_{i-1,j}, \mu_{i,j}, T_i = T_i/2)\}_{j\in[1,\rho]} \quad \text{and} \quad \{(\mu_{i,j}, y_{i-1,j}\alpha^{a_{i-1,j}}, T_i)\}_{j\in[1,\rho]}, \tag{3.5}$$

which we denote by

$$\left\{(u_{i,k}, v_{i,k}\alpha^{b_{i,k}}, T_i)\right\}_{k\in[1,2\rho]}. \tag{3.6}$$

Note that at least one of the $b_{i,k}$ is non-zero modulo p, no matter which elements $\mu_{i,j}$ the prover sends. Hence, the assumption of Lemma 1 is satisfied, so the probability that all of the statements in round $i+1$ are true is at most $1/B^\rho$. By the union bound, we get that the probability that all statements are true after t rounds is t/B^ρ.

Case 2: Let $\mathrm{ord}(\alpha) = 2^{C+1}$ where $C = t\ell$ for some $\ell \geq \log B$. In order to end up with a true statement after t rounds, $\tilde{\mathcal{P}}$ has to decrease the order of the false element by a factor of 2^ℓ on average per round. In particular (by an averaging argument) there has to be one round where the order decreases by at least 2^ℓ.

Assume that in round i of the protocol we have ρ statements given in Equation (3.4). Without loss of generality, let $\alpha^{a_{i-1,1}}$ have the largest order of all $\alpha^{a_{i-1,j}}$. The prover sends midpoints $\mu_{i,j}$ which results in 2ρ statements given in Equation (3.5) which we then denote as in Equation (3.6). We note that no matter the value of midpoint $\tilde{\mathcal{P}}$ sends, the order of the element that makes one of the two statements

$$\mu_{i,1} \stackrel{?}{=} x_{i-1,1}^{q^{T_i}} \quad \text{and} \quad y_{i-1,1}\alpha^{a_{i-1,1}} \stackrel{?}{=} \mu_{i,1}^{q^{T_i}}$$

false is at least $\mathrm{ord}(\alpha^{a_{i-1,1}})$. To see this, assume that $\mu_{i,1}$ is the correct midpoint but $\tilde{\mathcal{P}}$ sends $\mu_{i,1}\beta$ for some group element β. Then the second statement becomes

$$y_{i-1,1}\alpha^{a_{i-1,1}}\beta^{-q^{T_i}} \stackrel{?}{=} \mu_{i,1}^{q^{T_i}},$$

which is γ-false for $\gamma := \alpha^{a_{i-1,1}}\beta^{-q^{T_i}}$. Since $\alpha^{a_{i-1,1}} = \gamma\beta^{q^{T_i}}$ we have that $\mathrm{ord}(\alpha^{a_{i-1,1}})$ divides $\mathrm{lcm}(\mathrm{ord}(\gamma), \mathrm{ord}(\beta^{q^{T_i}}))$. It follows that $\mathrm{ord}(\alpha^{a_{i-1,1}})$ divides either $\mathrm{ord}(\gamma)$ or $\mathrm{ord}(\beta^{q^{T_i}})$ (and hence $\mathrm{ord}(\beta)$) because the order of $\alpha^{a_{i-1,1}}$ is a power of 2.

By Lemma 1, we get that the probability that none of the statements in round $i+1$ is $\tilde{\alpha}$-false, where $\tilde{\alpha}$ is some element with order divisible by $\mathrm{ord}(\alpha^{a_{i-1,1}})/2^{\ell-1}$, is at most $1/B^\rho$. By the union bound, we conclude that $\tilde{\mathcal{P}}$ wins after t rounds with probability at most t/B^ρ.

Cases 1 and 2 together yield Theorem 1. □

Corollary 1. *For $C := t \log B$ the Fiat-Shamir transform of our PoE yields a sound non-interactive protocol in the random-oracle model.*

Proof. As we have seen above, a cheating prover is able to convince the verifier of a false statement only if there is one round where at least one of the following two events happens depending on which attack is chosen:

- an α-false statement where $\mathrm{ord}(\alpha)$ has a prime divisor of size at least B is transformed into a true one or
- the order of the false element decreases by at least $2^{C/t}$.

We know that the probability that the output of a random oracle results in such an event is $(1/B)^\rho$ since, by our choice of C, we have

$$1/2^{\rho C/t} = (1/B)^\rho.$$

By a union bound, the probability that a cheating prover that makes up to Q queries to the random oracle will find such a query is at most $Q \cdot (1/B)^\rho$. □

3.2.2 Efficiency

In this section, we analyse the efficiency of the Fiat-Shamir transform of our PoE for proving a statement of the form $y \stackrel{?}{=} x^{q^T}$ with $T = 2^t + C$.

Randomness space. In order to keep the cost of exponentiation with random coins low, we need to make the size of the randomness space as small as possible while ensuring that divisibility by B is almost uniformly distributed. For concreteness, we use $\log B + 5$ random bits. Then it holds for any prime $p > B$ and $c \in \mathbb{Z}_p$ that

$$\Pr_{r \leftarrow \mathbb{Z}_{2^{\lceil \log B \rceil + 5}}}[r = c \mod p] < \frac{1}{B} + \frac{1}{B \cdot 2^5} \approx \frac{1.03}{B}.$$

Verifier's efficiency. The work for the verifier consists of two parts: 1) the interactive part, which is dominated by $t \cdot 4\rho^2$ exponentiations (with exponents of size $\log B + 5$) and ρ exponentiations with q, and 2) the final exponentiation with q^C. Each exponentiation with a z-bit exponent via the square-and-multiply algorithm costs about $1.5z$ multiplications (i.e., z plus the Hamming weight of the exponent), so the small exponentiations have complexity $6t\rho^2(\log B + 5)$. Additionally, the verifier performs $2t\rho^2$ multiplications to recombine the statements. The exponentiation with q^C takes $C \cdot \log(q)$ multiplications. If we set $C = t \cdot \log B$, the total of multiplications performed by the verifier is approximately

$$t \cdot ((6 \log B + 32)\rho^2 + \log B \cdot \log(q)) + \rho \log(q) \approx t \log B(6\rho^2 + 2B) + 2\rho B,$$

where we use the upper bound $q = \prod_{\text{prime } p<B} p < 4^B$ of Erdős [Erd32] to bound $\log(q)$ by $2B$. As an example, consider an implementation where $t = 32$, $B = 521$, and $\rho = \lceil 80/\log(521) \rceil$=9. Then we have $\log(q) \approx 703$, so the cost for the verifier is around 426000 multiplications.

In Figure 3.3, we plot the complexity of the verifier *in a single round* of the interactive protocol for different values of B. Additionally, we consider the curves for the verifier's complexity of only the interaction with the prover and only the final exponentiation separately. Observe that, for $B < 227$, the total complexity decreases as B increases due to the fact that the number

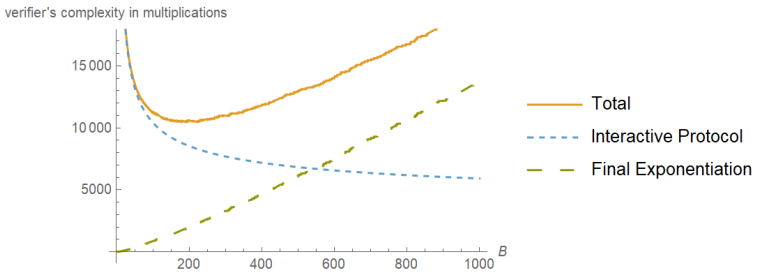


Figure 3.3: Number of multiplications of the verifier in one round for 80-bit security depending on the bound B. The orange solid curve is the total verifier's complexity for one round, the blue dotted graph is the cost of the interactive part of the protocol and the green dashed graph is the cost of the final exponentiation divided by the number of rounds (i.e., we amortize the cost of the final exponentiation over the number of rounds).

of repetitions $\lambda/\log B$ decreases faster than the increasing cost of the final exponentiation with q^C (the latter increases linearly with B). Beyond $B = 227$, it is the other way round and, thus, the total cost increases. Note that $B = 227$ implies $q \approx 2^{287}$. If an application requires either a value q that is much larger than this or PoEs for multiple statements (e.g., in [BHR+21], where λ many PoEs are needed in each round), then the final exponentiation of the verifier becomes too expensive. We present two modifications of the protocol that improve this complexity significantly: In Section 3.3, we show how to replace $C = \log T \log B$ with $C = \log T$ by slightly modifying how we set q. In Section 3.4, we show how to compute the last step interactively without increasing the number of rounds.

Prover's efficiency. The prover needs to compute x^{q^T} and the midpoints $\mu_{i,j}$. Computing x^{q^T} takes $\log(q) \cdot T$ multiplications. If the prover stores the value $x^{q^{T/2}}$ during that computation, then computing the midpoints takes another $\rho \cdot \log(q) \cdot (T/4 + T/8 + \ldots + 1) \approx \rho \cdot \log(q) \cdot T/2$ multiplications. This number can be significantly reduced by storing a few more elements during the computation of x^{q^T} similarly to [Pie19, Section 6.2]. For sufficiently large values of T, the cost for computing the proof can be made small compared to the cost of the T exponentiations required to compute the output and, moreover, the computation of the proof can be easily be parallelized. For this reason we mostly ignore the prover's complexity in the comparisons.

Communication complexity. The communication complexity from the prover to the verifier is of interest as it equals the proof-size after using the Fiat-Shamir heuristic. In each of the t rounds, our prover sends ρ many midpoints which are of size $\log N$. If $\log N = 2048$, $t = 32$, and $\rho = 9$ then the communication complexity is approximately 2^{19} bits.

Comparison with alternative PoEs. In Table 6.1, we compare our protocol with the proofs of exponentiation from [Pie19], [BHR+21], and [Wes20]. We list the proof-size and verifier's complexity. Prover's complexity is omitted since the main computation for the prover in all the protocols is dominated by the same factor, i.e., the cost of T sequential exponentiations to compute the output.

We observe that [Wes20] is the most efficient PoE regarding verifier's complexity and proof-size. However, it is not statistically-sound. [Pie19] introduces only a minor increase in

PoE	statistically-sound	Verifier's complexity	$\lvert\pi\rvert$
Our PoE	yes	$(6(\frac{\lambda}{\log B})^2 + 2B)\log B \log T + \frac{2\lambda}{\log B}$	$\frac{\lambda}{\log B}\log T$
[BHR+21]	yes	$2\lambda^2 \log T + 2\lambda \log(q)$	$\lambda \log T$
[Pie19]	in some $\mathbb{G}$	$3\lambda \log T$	$\log T$
[Wes20]	no	$\log T + 3\lambda$	1

Table 3.1: Comparison of different PoEs. Verifier's complexity is measured in the number of multiplications and proof-size $|\pi|$ in the number of group elements. By λ, we denote the statistical security parameter. [Pie19] is statistically-sound only in groups without elements of small order.

overhead, but it has the drawback that it is only statistically-sound in groups with no low-order elements other than the identity. The PoE from [BHR+21] and our PoE are both statistically-sound in all groups, while the proof-size of our PoE improves by a factor of $\log B$ upon [BHR+21] and we compare the communication complexity per round for different values of B in Figure 3.1.(a).

The verifier's efficiency of our PoE depends on the choice of the bound B which also determines the size of q. In Figure 3.1.(b), we compare the number of multiplications per round for the verifier in both protocols for different choices of B. Additionally to the work in each round, the verifier computes λ many exponentiations with q in the last round of [BHR+21] and ρ many exponentiations with q in the last round of our interactive protocol. We see that the verifier's complexity improves for $B \in (59, 499)$, which corresponds to $q \in (2^{71}, 2^{685})$.

It is important to note that this is the verifier's complexity for proving a single statement. The PoE in [BHR+21] achieves the same verifier's efficiency for proving λ many different statements with the same exponent simultaneously. Our protocol incurs additional $\log T \log(q)$ multiplications for every new statement, since the verifier has to compute the final exponentiation individually for every statement. In Section 3.4, we give a batching protocol that reduces the cost of the final exponentiation to $\log(q)$, which enables us to prove arbitrarily many statements simultaneously without significantly increasing the proof-size and verifier's complexity.

3.3 Reducing Verifier's Complexity by Modifying q

In Figure 3.1.(b) we see that for large values of B and q the verifier's complexity increases because the final computation $(y')^{q^C}$ becomes expensive. The cost of this computation is $C \cdot \log(q)$, where so far we have set $C = t\log(B)$. We can reduce this number to $C = t\log(B)/2$ by setting q to

$$q = 2^2 \cdot 3^2 \cdot \prod_{3<p<B} p. \tag{3.7}$$

It is straightforward to check that this does not affect our soundness bound, but it has a notable effect on verifier's efficiency as shown in Figure 3.4.

This approach can be generalized to setting $C = t\log(B)/k$ for any integer $k \leq \log(B)$. To ensure soundness we need to modify q as follows: Let m be the largest prime number such that $m < 2^k$. Then we set

$$q = 2^k \cdot 3^{\lceil k/\log(3)\rceil} \cdot 5^{\lceil k/\log(5)\rceil} \cdots m^{\lceil k/\log(m)\rceil} \cdot \prod_{m<p<B} p.$$

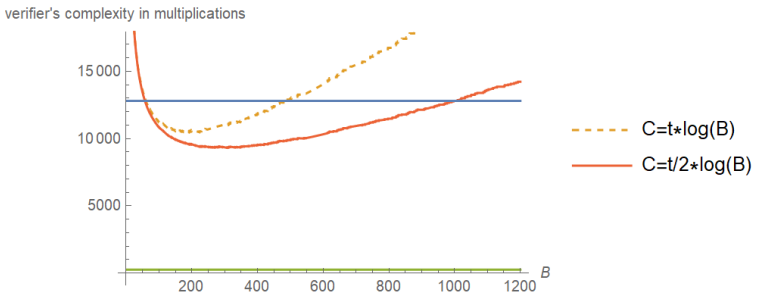


Figure 3.4: Number of multiplications of the verifier in one round for 80-bit security depending on the bound B. The solid blue line represents the number of multiplications in [BHR+21], the dotted orange curve represents the complexity of our protocol with $C = t\log(B)$, the solid red curve is the complexity in our protocol with $C = t\log(B)/2$ and the solid green line represents the verifier's complexity in [Pie19].

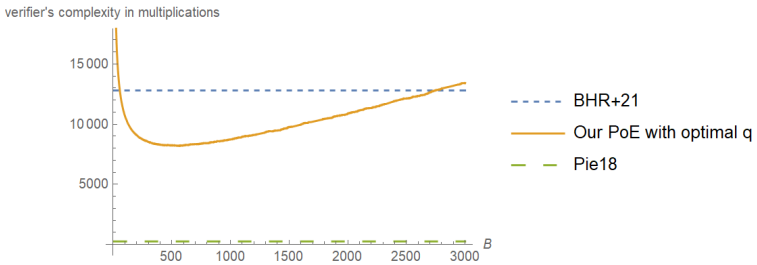


Figure 3.5: Number of multiplications of the verifier in one round for 80-bit security depending on the bound B. The dotted blue line represents the number of multiplications in [BHR+21], the solid orange curve is the complexity of our protocol with $C = t$ and q as above and the dashed green line is verifier's complexity in [Pie19] (which is 240 multiplications).

In particular, the choice of q that optimizes verifier's efficiency for large values of B is

$$q = \prod_{p<B} p^{\lceil \log(B)/\log(p) \rceil}$$

for which we can set $C = t$. The cost for the verifier with this parameters is shown in Figure 3.5. We conclude that the verifier's complexity of our scheme improves upon [BHR+21] for values of B from 59 up to 2749, which corresponds to values of q between approximately 2^{71} and $2^{400 \cdot \log(2749)} \approx 2^{3167}$.

3.4 Reducing (Verifier's) Complexity by Batching

In this section, we show how to prove arbitrary many statements simultaneously without increasing the number of rounds. This batching protocol serves two purposes:

1. Efficiently proving multiple independent statements. This is needed for example in the polynomial commitment scheme of [BHR+21], where in each round λ many statements need to be proven;

2. Reducing the verifier's complexity of the final exponentiation with q^C in our basic protocol. Instead of performing the computation locally, the verifier can request an

additional PoE for the statement $(y')^{q^C} = y$ and verify it simultaneously with the original PoE. While now we need to do a final exponentiation for the new statement, the constant in the exponent drops from $\log T$ to $\log \log T$.

In [Rot21] Rotem gives a batching technique for arbitrary PoEs, where the statements have the same exponent. We describe a batching technique for our PoE, where the statements can have different exponents. Furthermore, the protocol can be easily adapted to the PoEs in [Pie19] and [BHR+21].

3.4.1 The Protocol

Assume the prover wants to prove two statements in the same group $\mathbb{G}$:

$$h_1 \stackrel{?}{=} g_1^{q^{2^t}+C_1} \quad \text{and} \quad h_2 \stackrel{?}{=} g_2^{q^{2^s}+C_2}.$$

The statements can either be independent or one of them is the statement from the final verifier exponentiation of the other. The two statements can be proven simultaneously as follows. First the prover sends the statements

$$h_1' \stackrel{?}{=} g_1^{q^{2^t}} \quad \text{and} \quad h_2' \stackrel{?}{=} g_2^{q^{2^s}}.$$

We can assume that $t = \ell + s$ for some $\ell \in \mathbb{N}$. Begin with the proof of the first statement. After executing the protocol for $\ell - 1$ rounds and the prover sending midpoints in round ℓ, we have 2ρ statements

$$\left\{ v_j \stackrel{?}{=} u_j^{q^{2^s}} \right\}_{j \in [2\rho]}.$$

The prover makes this $2\rho + 1$ statements by adding $h_2' \stackrel{?}{=} g_2^{q^{2^s}}$ to them. Next the verifier sends $\rho \cdot (2\rho + 1)$ random coins and both parties create ρ new statements similarly to the original protocol. Then they proceed with the PoE protocol. Note that this process neither reduces soundness of the proof of the first statement nor of the second statement since by Lemma 1 we only need one of the statements that are being combined to be false. In the end the verifier checks if $h_1 = (h_1')^{q^{C_1}}$ and $h_2 = (h_2')^{q^{C_2}}$. This process can be extended to arbitrary-many statements of the form $h_i \stackrel{?}{=} g_i^{q^{2^r}+C_i}$ with the protocol given in Figure 3.6. Note that in Step 4 we do not specify whether the verifier checks $h_i = (h_i')^{q_i^C}$ by carrying out the computation locally or by appending it to the statements. This depends on the size of C and on the application.

Remark 2. In the case where the exponents of q are not powers of 2, one can simply divide a statement of the form $y \stackrel{?}{=} x^{q^S}$ for $S \in \mathbb{N}$ into smaller statements as follows. Let $(s_0, s_1, \ldots, s_m)$ be the binary representation of S. Then we have

$$x^{q^S} = x^{q^{\sum s_k \cdot 2^k}} = x^{\prod q^{s_k \cdot 2^k}} = y.$$

This gives at most $m+1$ smaller statements $y_1 \stackrel{?}{=} x^{q^{s_0}}$ and $y_{i+1} \stackrel{?}{=} y_i^{q^{s_i \cdot 2^i}}$ for $i \in [1, m]$ where $y_{m+1} = y$. Again these statements can be proven simultaneously with the batching protocol.

The theorem below follows immediately from the description of the batching protocol and Remark 2.

Parameters: Same as in Figure 3.2
Statements: $\left\{h_i \stackrel{?}{=} g_i^{q^{2^{t_i}+C_i}}\right\}_{i\in[1,m]}$ in $L_{\mathbb{G}}$ with and $t_1 > t_2 > \ldots > t_m \in \mathbb{N}$
Protocol:

1. The prover $\mathcal{P}$ sends $h'_i := g_i^{q^{2^{t_i}}}$ for all $i \in [1,m]$ to $\mathcal{V}$.
2. Execute Step 3 of the PoE protocol for $(g_1, h'_1, 2^{t_1})$ for $t_1 - t_2 - 1$ rounds.
3. In round $i \in [1, m-1]$ of the batching protocol we have ρ statements of the form $\{(x_j, y_j, 2^{t_{i+1}+1})\}_{j\in[1,\rho]}$:
   a) The prover $\mathcal{P}$ sends ρ midpoints $\{\mu_j\}_{j\in[1,\rho]}$, which results in 2ρ statements $\{(u_k, v_k, 2^{t_{i+1}})\}_{k\in[1,2\rho]}$
   b) The prover $\mathcal{P}$ and verifier $\mathcal{V}$ append $(g_{j+1}, h'_{j+1}, 2^{t_{i+1}})$ to the statements resulting in $2\rho+1$ statements of the form $\{(\tilde{u}_k, \tilde{v}_k, 2^{t_{i+1}})\}_{k\in[1,2\rho+1]}$.
   c) The verifier $\mathcal{V}$ sends the random challenge $\{r_{j,k}\}_{j\in[1,\rho],k\in[1,2\rho+1]}$, where $r_{j,k} \in \{0,1\}^\kappa$.
   d) They both set $\{(\tilde{x}_j, \tilde{y}_j, 2^{t_{i+1}})\}_{j\in[1,\rho]}$ as the statement for the next execution of the PoE protocol, where
   $$\tilde{x}_j := \prod_{k\in[1,2\rho+1]} \tilde{u}_k^{r_{j,k}} \quad \text{and} \quad \tilde{y}_j := \prod_{k\in[1,2\rho+1]} \tilde{v}_k^{r_{j,k}}$$
   e) If $i < m-1$: Execute Step 3 of the PoE protocol for $t_{i+1} - t_{i+2} - 1$ rounds. Else: Execute Step 3 of the PoE protocol for t_m rounds until the statements are of the form $\left\{(x_j^*, y_j^*, 1)\right\}_{j\in[1,\rho]}$.
4. At the end of $m-1$ rounds, the verifier $\mathcal{V}$ accepts if and only if $(x_j^*)^q = y_j^*$ for all $j \in [1,\rho]$ and $(h'_i)^{q^{C_i}} = h_i$ for all $i \in [1,m]$.

Figure 3.6: Batching protocol for PoE.

Theorem 2. *For any $m \in \mathbb{N}$ the statements $\{(g_i, h_i, S_i + C_i)\}_{i\in[1,m]}$ can be proven in at most $1 + \max_i \log(S_i)$ rounds where additionally to one execution of the PoE protocol the following computations need to be performed:*

1. *$\mathcal{P}$ and $\mathcal{V}$ perform*
   $$2\rho \sum_{i=1}^{m} h(S_i)$$
   additional exponentiations with exponents of size $\log B + 5$. Here $h(S_i)$ denotes the hamming weight of S_i;
2. *$\mathcal{V}$ performs $m-1$ additional exponentiations with exponents q^{C_i} for $i \in [1,m] \setminus \{\arg\max_i S_i\}$;*

and the communication complexity increases by $m-1$ group elements.

Soundness of the protocol follows immediately from Lemma 1 and Theorem 1 since in the statement of Lemma 1 we consider a set of arbitrary many statements of the form (x_i, y_i, T) in any round. This means that the proof of Theorem 1 also holds when new statements are added during the execution of the protocol.

Theorem 3. *Let B be any prime number such that $q := \prod_{\text{prime } p<B} p$ and $\rho \in \mathbb{N}$ be the number of repetitions per round. If we set $C = \log T \log B$ and let $\kappa \to \infty$, the verifier $\mathcal{V}$ will output* `accept` *on statement $\{(g_i, h_i, 2^{t_i} + C_i)\}_{i\in[1,m]}$, where $t_1 \geq t_2 \geq \ldots \geq t_m$ and at least one statements is false, with probability at most t_1/B^{ρ}.*

3.4.2 Improving Verifier's Efficiency

In this section we analyse how the batching protocol reduces the number of multiplications for verifying a statement of the form $y \stackrel{?}{=} x^{q^T}$. In Section 3.5.3 we analyse the gain in efficiency of the polynomial commitment in [BHR+21] when we use this improved version of our PoE as a building block instead of the PoE proposed in [BHR+21].

The first prover message is the value $y' = x^{q^{T-C}}$, where $C \geq \log T$. The key idea is that the verifier does not carry out the last exponentiation with q^C but the prover gives an interactive proof of the statement $(y')^{q^C} = y$ (a "smaller" PoE). This reduces the final exponentiation to $(y'')^{q^{C'}} = y$, where y'' is the first prover message in the smaller PoE and $C' \geq \log(C)$ is much smaller than C. This statement can again be proven interactively by an even smaller PoE. In fact, this trick can be applied recursively until the verifier only has to perform a single exponentiation with q in the final step. We make two assumptions in this section:

1. We have $q = \prod_{\text{prime } p<B} p^{\lceil \log B/\log(p)\rceil}$ such that the constant C in the PoE protocol is lower bounded only by $\log T$ and not $\log T \log B$. This is the trick we discussed in Section 3.3. This assumption is needed to reduce the exponent from q^C to q and should be adopted in practice if one wants to make use of the recursion.

2. Instead of setting C to exactly $\log T$, we set $C = 2^{2^{2^2}} + 2^{2^2} + 2^2 + 1$, which will always be larger than $\log T$ in practice. This assumption is mainly for the ease of presentation and need not be adopted in practice.

Reducing the exponent from q^C to $q^{\log(C)}$. We know that exponentiation with q^C takes $C \log(q)$ multiplications. In order to reduce this cost for the verifier, we slightly modify the protocol in the following way: Instead of the verifier performing the last exponentiation locally, the verifier and the prover run the batching protocol with statements

$$\{(x, y, T = T_0 + C), (y', y, C = S_0 + C')\},$$

where $C' = \log(C)$. This modification introduces $3\rho \cdot h(S_0)(\log B + 5)$ additional multiplications during the interactive part of the protocol (by Theorem 2) *but* reduces the complexity of the final exponentiation to

$$C' \log(q) = \log(C) \log(q) \approx \log\log T \log(q).$$

By our special choice of C we have $h(S_0) = 1$ so we can ignore it in the remainder of the section

Applying the recursion. As we have seen, the exponent q^C can be reduced to $q^{C'}$. Now, the verifier can either perform the final exponentiation with $q^{C'}$ or apply the above procedure recursively until the verifier only has to do a single exponentiation with q in the final step. We denote the number of recursions needed until the exponent is reduced to q by $\log^*(C)$. We have that the entire recursion adds at most $3\log^*(C)\rho\cdot(\log B+5)$ multiplications during the interactive part of the protocol but reduces the work of the final exponentiation from $\log T\log(q)$ to $\log(q)$.

In Section 3.2.2 we saw that the verifier's complexity without any batching is

$$\log T\cdot((6\log B+32)\rho^2+\log(q))+\rho\log(q).$$

Our batching protocol reduces the number of multiplications for verifying the proof of a single statement to approximately

$$\log T(6\log B+32)\rho^2+3\log^*(C)\rho\cdot(\log B+5)+(\rho+1)\log(q)$$

and increases the proof-size to $\log^*(C)+\rho\log T$ group elements.

Proving multiple statements. With this optimization of the cost of verifying a single statement we can now compute the complexity of verifying m statements with our improved protocol. Each additional statement that either has exponent q^T or a smaller power of q adds $\log(q)$ multiplications to compute the final exponentiation, $3\log^*(C)\rho\cdot(\log B+5)$ multiplications during the interactive part and increases the proof-size by at most $\log^*(C)$ elements. We conclude that m many statements can be proven with verifier's complexity

$$\log T(6\log B+32)\rho^2+3m\log^*(C)\rho\cdot(\log B+5)+(\rho+m)\log(q)$$

and communication complexity $m\log^*(C)+\rho\log T$.

3.5 Application in Polynomial Commitments

In this section, we discuss the application of our protocol to the polynomial commitment scheme in [BHR+21]. In particular we show in Section 3.5.2 that one can choose the parameter q to be even and in Section 3.5.3 we analyse the gain in efficiency when we use our PoE as a building block instead of the one proposed in [BHR+21].

A polynomial commitment scheme [KZG10] allows one party – *the committer* – to commit to a (low-degree) polynomial P. Another party – *the verifier* – can later ask the committer for an evaluation $y=P(x)$ along with a proof that helps it (efficiently) verify that the evaluation is consistent with the initial commitment c. There are two main properties that the polynomial commitment must satisfy: correctness and binding. Loosely speaking, a polynomial commitment scheme is correct if the (honest) committer can convince the verifier of the value $y=P(x)$ of the polynomial on *any* point x on its domain, whereas it is (computationally) binding if no (computationally-bounded) cheating prover can convince the verifier of a wrong evaluation $y'\neq P(x)$. Since [BHR+21, BFS20] use their polynomial commitment scheme to build an *argument of knowledge*, they require the stronger property called *knowledge soundness* instead of just binding – i.e., the committer must know a polynomial $P(x)$ such that $y=P(x)$ and c is a commitment to $P(x)$ (formalised via an extractor). The ideas above can be naturally extended to *multilinear* polynomials.

3.5.1 [BHR+21] Polynomial Commitments

The time- and space-efficient (zero-knowledge) argument of knowledge from [BHR+21] is built on top of a time- and space-efficient (multilinear) polynomial commitment scheme. We first provide an overview of this polynomial commitment scheme, and then highlight the key properties that it should satisfy.

Commitment. To commit to a degree-n multilinear polynomial $P : \mathbb{F}^n \to \mathbb{F}$ over a finite field $\mathbb{F}$ of order p, the committer evaluates P over the Boolean hypercube $\{0,1\}^n$ to obtain a sequence of field elements $(z_0, \ldots, z_{N-1}) \in \mathbb{F}^N$, where $N := 2^n$. This sequence is then interpreted as a sequence of digits $\mathcal{Z}$ base (large-enough) $q \in \mathbb{N}$ of an integer $z \in \mathbb{Z}$ — z is said to be the *integer encoding* of the polynomial P (see Algorithm 1). The commitment, finally, is obtained by computing the exponent $c = g^z$, where g is a random element in a group of unknown order (e.g., RSA group or class groups of imaginary quadratic field). [BHR+21] show how to carry this out using time $\tilde{O}(2^n)$ and space $\mathbf{poly}(n)$ given multi-pass streaming access to the evaluations of P on the Boolean hypercube.

Evaluation. Let $P : \mathbb{F}^n \to \mathbb{F}$ be a degree-n multilinear polynomial with integer encoding z and, for $b \in \{0,1\}$, let $P_b : \mathbb{F}^{n-1} \to \mathbb{F}$ be defined as $P(b, \cdot)$. Once committed to $c = g^z$, to prove in a verifier-efficient manner that $P(\boldsymbol{\zeta}) = \gamma$ for some $(\boldsymbol{\zeta} = (\zeta_1, \zeta_2, \ldots, \zeta_n), \gamma) \in \mathbb{F}^n \times \mathbb{F}$, the committer and the verifier proceed interactively. In the first round the committer computes, for $b \in \{0,1\}$, $c_b := g^{z_b}$, where $z_b \in \mathbb{Z}$ is the integer encoding of P_b and $\gamma_b := P_b(\zeta_2, \ldots, \zeta_n)$. It sends $(c_0, c_1, \gamma_0, \gamma_1)$ to the verifier. The verifier checks whether

1. $\gamma = \zeta_1\gamma_1 + (1 - \zeta_1)\gamma_0$ (which should hold since $P(\boldsymbol{\zeta}) = \zeta_1 P_1(\zeta_2, \ldots, \zeta_n) + (1 - \zeta_1)P_0(\zeta_2, \ldots, \zeta_n)$); and
2. $c_0(c_1)^{q^{N/2}} = g^{z_0 + q^{N/2} z_1} = g^z = c$.

Note that the second equality in Item 2 relies on the homomorphic property of the integer encoding and, in turn, the commitment. Since checking $c_0(c_1)^{q^{N/2}} = c$ involves computing $(c_1)^{q^{N/2}}$ which can be expensive to the verifier, a PoE is employed to prove $c_0/c = (c_1)^{q^{N/2}}$. Now, note that checking the validity of $P(\boldsymbol{\zeta}) = \gamma$ has been reduced to checking the validity of *two* degree-$n-1$ expressions $P_b(\zeta_2, \ldots, \zeta_n) = \gamma_b$. Since recursing on both expressions is too expensive, the committer and verifier *fold* them into a single statement via random linear combination: the verifier sends a random $\alpha \in \mathbb{F}$ and the new statement is $P(\boldsymbol{\zeta}') = \gamma'$ with commitment $c' = c_0 c_1^{\alpha}$, where $\boldsymbol{\zeta}' := (\zeta_2, \ldots, \zeta_n)$ and $\gamma' = \gamma_0 + \alpha\gamma_1$. The knowledge soundness (and hence binding) of the commitment scheme is relies on the *hidden order assumption* in groups of unknown order.

Requirements from the PoE. Note that the use of the PoE in the [BHR+21] polynomial commitment is more or less black-box. However, there are two important criteria that it should satisfy.

1. Firstly, the PoE has to satisfy statistical soundness so that the knowledge soundness of the polynomial commitment built upon it can be argued ([BHR+21, Lemma 6.4]).[6]

[6] To be precise, it suffices for the soundness of the PoE to be based on a hardness assumption that is *at most* as strong as the hardness assumption that is used for showing the binding or knowledge soundness of the polynomial commitment.

Our PoE satisfies statistical soundness.

2. Secondly, the exponent q used in the PoE protocol is borrowed *from* the polynomial commitment. In order for the polynomial commitment to satisfy its homomorphic properties, [BHR+21] set it to be a *large, odd* integer – in particular, they require $q \gg p \cdot 2^{n^{\mathbf{poly}(\lambda)}}$. This requirement that q be large, as we saw in Section 3.2 is advantageous for our PoE. On the other hand, the requirement that q be odd is in conflict with our trick of choosing an *even* q as in Equation (3.2). However, we show in the next section that the requirement that q be odd is not necessary in [BHR+21].

3.5.2 Polynomial Commitments with Structured Base

Recall that the exponent q in the PoE protocol [BHR+21] is borrowed from the polynomial commitment scheme built upon it. We first observe that none of the claims pertaining to the integer encoding $(\mathsf{Enc}_q, \mathsf{Dec}_q)$ in the polynomial commitment of [BHR+21] and its use in extraction rely on the exponent q being odd. In fact, the assumption in [BHR+21] that q be odd is an artefact of [BFS20] (as confirmed in a personal communication with the authors of [BHR+21]). We show in Lemmas 2 and 3 that the properties of the encoder and decoder that are necessary for the polynomial commitment of [BHR+21] to work also hold for even – and hence arbitrary – q. This allows us to use structured exponents of the form required in Section 3.2 (e.g., q as in Equation (3.2)). We first describe (for self-containment) the integer encoding from [BHR+21] in Algorithm 1 and then prove that they are consistent over $\mathbb{Z}(q/2)$ for any $q \in \mathbb{N}$ (Lemma 2). We then prove that the homomorphic properties of the decoding algorithm holds for all q (Lemma 3). Since the rest of the proofs pertaining to the polynomial commitment are unaffected by the change in exponent, [BHR+21, Theorem 4.2] can be proven also based on our PoE.

Algorithm 1 Integer encoding from [BHR+21, BFS20].

Common parameters:

1. Base $q \in \mathbb{N}$
2. Degree $n \in \mathbb{N}$ with $N := 2^n$

procedure Enc($\mathcal{Z}$)
  Parse $\mathcal{Z} =: z_0, \ldots, z_{N-1} \in \mathbb{Z}(q/2)^N$
  return $v := \sum_{\boldsymbol{b} \in \{0,1\}^n} q^{\boldsymbol{b}} z_{\boldsymbol{b}}$
end procedure

procedure Dec(v)
  for $k \in [0, N]$ **do**
    $S_{k-1} := v \mod q^k$
    if $S_{k-1} > q^k/2$ **then** $S_{k-1} := S_{k-1} - q^k$ **end if**
    $S_k := v \mod q^{k+1}$
    if $S_k > q^{k+1}/2$ **then** $S_k := S_k - q^{k+1}$ **end if**
  end for
  return $\mathcal{Z} := (z_0, \ldots, z_{N-1})$
end procedure

Lemma 2 (Bijectivity of encoder for all q, restatement of [BHR+21, Fact 5.1] and [BFS20, Fact 1]). *Let $q, N \in \mathbb{N}$ with $q \geq 2$. For any $v \in \mathbb{Z}(q^N/2)$, there exists a unique sequence $\boldsymbol{z} \in \mathbb{Z}(q/2)^N$ such that $v = \mathsf{Enc}_q(\boldsymbol{z})$. Furthermore, $\boldsymbol{z} = \mathsf{Dec}_q(v)$.*

Proof. The proof follows by inspection of that from [BHR+21]. That is, we argue that:

1. the domain and range have the same size; and
2. the composition of decoding and encoding functions is identity.

Since, $\mathbb{Z}(B) := \{x \in \mathbb{Z} : -B \leq x < B\}$, it follows that $\left|\mathbb{Z}(q^N/2)\right| = \left|\mathbb{Z}(q/2)^N\right| = q^N$. To show Item 2, we proceed by induction on the elements of a sequence $\mathcal{Z} := z_0, \ldots, z_{N-1} \in \mathbb{Z}(q/2)^N$.

Base case. To see that the decoder correctly recovers the first element z_0 from $\mathsf{Enc}(\mathcal{Z})$, we note that its first iteration (i.e., $k = 0$) is simply the modulo operation base q followed by a conditional shift by $q/2$. By taking the encoding function $\mathsf{Enc}(\mathcal{Z})$ modulo q, note that the higher powers of q disappear and only $z_0 \mod q$ remains. The correct representative from $[q/2, q/2)$ is then recovered by the conditional shift.

Induction hypothesis. Assume that the first k elements $z_0, \ldots, z_{k-1}$ have been correctly recovered. In particular, this implies the sequence $S_{-1}, S_0, \ldots, S_k$ has been correctly computed.

Induction. To see that the decoder correctly recovers z_{k+1}, we take $\mathsf{Enc}(\mathcal{Z})$ modulo q^{k+1}. Since S_k has been correctly computed, and since $z_{k+1}q^k + S_k = v = S_{k+1} \mod q^{k+1}$, it follows from the description of the decoder (i.e., $z_{k+1} := (S_{k+1} - S_k)/q^k$) that it recovers the correct representative of z_{k+1} after the conditional shift.

□

Lemma 3 (Homomorphism of decoder for all q, restatement of [BHR+21, Claim 5.2]). *Let $q, N \in \mathbb{N}$ with $q \geq 2$. Also let $\ell \in \mathbb{N}$ and $B_1, B_2 \geq 1$ be such that $B_1 \cdot B_2 \leq q/(2\ell)$. Then, for every $a_1, \ldots, a_\ell \in \mathbb{Z}(B_1)$, and integers $z_1, \ldots, z_\ell \in \mathbb{Z}(q^N/2)$ such that $\mathsf{Dec}_q(z_i) \in \mathbb{Z}(B_2)^N$,*

$$\mathsf{Dec}_q\left(\sum_{i\in[1,\ell]} a_i \cdot z_i\right) = \sum_{i\in[1,\ell]} a_i \cdot \mathsf{Dec}_q(z_i) \tag{3.8}$$

Sketch. Once Lemma 2 has been reproved for even q, the argument is the same as in[BHR+21]. That is, one argues that:

1. Encoding of LHS and RHS in Equation (3.8) are equal; and
2. Encoding of LHS and RHS in Equation (3.8) are in $\mathbb{Z}_q^N$. □

3.5.3 Efficiency

In this section we analyse the improvement in efficiency of the polynomial commitment scheme in [BHR+21] using our PoE, the batching protocol and the optimization in Section 3.3. In the polynomial commitment scheme the PoE protocol is used to prove statements of the form $x_i^{q^{2^{n-k-1}}} = y_i$ for every $i \in [\lambda]$ and every $k \in \{0, 1, \ldots, n-1\}$.

Communication complexity. In [BHR+21] the communication complexity of proving λ many statements with the same exponent is $\lambda(n-k-1)$ group elements. This gives a total PoE proof-size of

$$\lambda \sum_{k=0}^{n-1}(n-k-1) = \frac{\lambda}{2}(n-1)n.$$

As we have seen in Section 3.4.2, in our PoE the cost of proving λn statements, in which the largest exponent is q^{n-1}, is

$$\lambda n \log^*(n-1) + \frac{\lambda}{\log(B)}(n-1).$$

We conclude that we decrease the proof-size of the polynomial commitment by a factor of approximately $n/(2\log^*(n-1))$. This number can be increased to $n/2$ at the cost of a higher verifier complexity. More generally, the number of recursive steps explained in Section 3.4.2 can be used to choose a trade-off between proof-size and verifier efficiency.

Verifier's efficiency. In [BHR+21] the verifier's complexity of proving λ many statements with the same exponent is $2\lambda^2(n-k-1) + \lambda\log(q)$ multiplications. This gives a total verifier's complexity of

$$2\lambda^2 \sum_{k=0}^{n-1}((n-k-1) + \lambda\log(q)) = (\lambda\log(q) + 2\lambda^2(n-1))n.$$

As we have seen in Section 3.4.2, in our PoE the cost of verifying λn statements, in which the largest exponent is q^{n-1}, is

$$(n-1)(6\log(B)+32)\rho^2 + 3\lambda n\log^*(C)\rho\cdot(\log(B)+5) + (\rho+\lambda n)\log(q) \approx 15\lambda^2 n + \lambda n\log(q).$$

Since in practice we have $n \approx 32$, we conclude that the verifier's efficiency of the polynomial commitment scheme implemented with our PoE is comparable to that in [BHR+21].

3.6 Conclusion

In this chapter, we presented a new construction of statistically sound PoEs which is secure in arbitrary groups, even when the group order is known. By leveraging structured exponents – specifically, the product of all primes up to a chosen bound – we were able to reduce the number of parallel repetitions required for soundness, significantly improving over the concrete efficiency of previous construction by Block et al. While Pietrzak's PoE in groups of signed quadratic residues is still more efficient than our PoE, those groups require trusted setups. An interesting open problem is to construct a statistically sound PoE that is as efficient as Pietrzak's PoE but does not require any restriction on the type of hidden order group.

CHAPTER 4

Batching Proofs of Exponentiation

4.1 Introduction

One practically important feature of PoEs is that many statements can be efficiently aggregated to a single statement. In particular, m statements $y_1 \stackrel{?}{=} x_1^e, \ldots, y_m \stackrel{?}{=} x_m^e$ can be represented by a single aggregate statement $\prod_{i=1}^m y_i \stackrel{?}{=} (\prod_{i=1}^m x_i)^e$. This property suggests that one might be able to construct *batch* Proofs of Exponentiation, i.e., secure aggregation protocols that allow one to verify many statements by checking a single PoE. Clearly, the above naive aggregation does not have any meaningful security properties – it is easy to produce incorrect statements that would be aggregated into a valid statement, e.g., by randomly permuting the correct y_i's. However, a sound batching procedure can be constructed by randomizing the aggregation using a suitable vector of exponents $r_1, \ldots, r_m \in \mathbb{Z}_p$ and considering the batched statement $\prod_{i=1}^m y_i^{r_i} \stackrel{?}{=} (\prod_{i=1}^m x_i^{r_i})^e$.

Notice that, in the extreme case where the randomization exponents are uniformly random bits $r_1, \ldots, r_m \in \mathbb{Z}_2$, the procedure amounts to the naive aggregation performed on a random subset of the original statements. However, it already achieves soundness error $\frac{1}{2}$ for an arbitrary collection of statements.[1] There are two natural approaches for boosting the soundness (i.e., reducing the soundness error). In applications requiring statistical security, λ parallel instances of the random subset approach can be run to get soundness error $2^{-\lambda}$. We refer to the parallel repetition variant of the random subset approach as the *Random Subsets Protocol*. It was independently suggested by Block et al. [BHR+21] and Rotem [Rot21].

The overhead stemming from the λ parallel repetitions can be avoided when one is willing to settle for computational soundness guarantees from the batching process. In particular, one can perform a single randomized batching using exponents from a sufficiently large set (i.e., $r_1, \ldots, r_m \in \mathbb{Z}_{2^\ell}$), which is computationally sound, assuming that the adversary cannot efficiently produce group elements of low order. The corresponding batch PoE, which we call the *Random Exponents Protocol*, was first proposed by Wesolowski [Wes20] for his VDF construction and later analysed by Rotem [Rot21] for general PoEs.

[1] This can be seen by the following argument: Suppose that the i-th statement is false and batch all statements except for the i-th one. Denote by b the uniformly random bit determining whether the i-th statement is multiplied to the batched statement or not. If the batched statement is correct, then it will remain correct if and only if $b = 0$. If it is wrong, one needs at least that $b = 1$ for it to become correct. In both cases, the probability of resulting in a correct statement is at most $\frac{1}{2}$.

The practical efficiency of the resulting batch PoE ultimately depends on the number of group multiplications performed. When raising the instances to uniformly random κ-bit exponents before computing their product, one has to perform 1.5κ multiplications in expectation per each exponentiation. Thus, for $\kappa = \lambda$, the Random Subsets Protocol based on computing products of λ random subsets results (in expectation) roughly in half as much work as in the Random Exponents Protocol that raises each statement to a random λ exponent before computing their product. However, when batching via the Random Subsets Protocol, λ proofs have to be constructed and verified, while the Random Exponents Protocol results in a single PoE. Note that proofs can be efficiently verified, but the construction is not as efficient, so proving λ many statements instead of one yields a significant loss in the prover's efficiency in practice. Hence, the above two batch PoEs from the literature give a trade-off for the complexities of the verifier and prover.

To evaluate the improvement achieved by the above batch PoEs, we need to compare them to the baseline when one disregards any attempt at batching and simply verifies all PoEs, which, e.g., avoids any implementation overhead of batching. Clearly, batch PoEs minimize the storage requirement, as instead of storing m proofs, one can store λ or even a single proof. Below, we discuss the number of expected multiplications per instance in the PoEs from Pietrzak [Pie19] and Wesolowski [Wes20] to compare with the verification complexity in the known batch PoEs.

For Wesolowski's proof, presented in Figure 2.1, the verifier checks one group identity $y = \pi^{\ell} x^{r}$ and uses at most $3\kappa + 1$ multiplications in expectation. Setting $\kappa = \lambda$ yields a verification cost similar to the Random Exponents Protocol for m instances. However, the non-interactive variant requires $\kappa = 2\lambda$ due to an attack by Boneh, Bünz, and Fish [BBF24], making the Random Exponents Protocol roughly twice as fast in practice. Additional overheads reported by Attias et al. [AVD22] may further favor the Random Exponents Protocol.

For Pietrzak's proof, presented in Figure 2.2, which proceeds in $\log T$ rounds, each round requires constructing new statements and costs at most $3\kappa+2$ multiplications in expectation. A final group identity check adds one multiplication. Thus, the total cost is about $(3\kappa + 2)\log T$, and the batching benefits increase as the time delay parameter T increases.

In this chapter, we present two new approaches for batch PoEs that improve over the known Random Subsets and Random Exponents protocols in various ways. Additionally, we show how to use a secure batch PoE for remote benchmarking of parallelism in a publicly verifiable manner.

4.1.1 New batching protocols

The first approach, which we call the *Hybrid Protocol*, is a natural combination of the two known approaches we discussed. It only slightly increases the work performed by the verifier compared to the Random Subsets Protocol but results in a single PoE proof that has to be constructed and verified. Specifically, using the Random Subsets Protocol described above, m instances are first batched to λ instances, which are then batched to a single PoE instance via the Random Exponents approach with λ-bit exponents. Our experiments show that this new batch PoE, while very easy to implement, is already roughly twice as fast as the Random Exponents Protocol.

The second approach, which we call the *Bucket Protocol*, can be seen as an optimization of the Hybrid Protocol that, for a large enough number of instances, is one order of magnitude

faster than the random exponents approach and five times faster than the random subsets approach. It is based on a protocol of Bellare, Garay and Rabin [BGR98] for batch verification of signatures in prime-order groups. The main difference to our construction is that we ultimately aggregate all statements into one, such that only one proof needs to be computed and verified in the end, while the original batching of Bellare et al. yields multiple proofs that need to be verified in parallel.

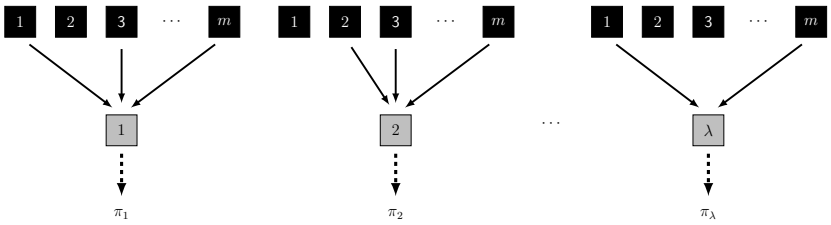
The main idea of the Bucket Protocol is to optimize the number of multiplications incurred in the Random Subsets Protocol by decreasing the number of parallel repetitions as follows: In the Random Subsets Protocol, a new PoE statement is constructed by taking a random subset of the m original PoE statements and then computing the product of the statements in the subset. Since we select half of the statements in a random subset in expectation, starting with m instances and repeating this step λ times to amplify the soundness results in $\lambda m/2$ multiplications in expectation. In the Bucket Protocol, each of the m instances is assigned uniformly at random to one of $K = 2^k$ buckets, where $k \in \mathbb{N}$ is a parameter of the protocol. Each bucket then gives rise to a new statement by taking the product of the statements that are assigned to that bucket, and the resulting K instances are then batched via the Random Exponents Protocol using random exponents of size k. The soundness analysis shows that, in this case, $\rho \approx \lambda/k$ parallel repetitions are sufficient, reducing the number of statement multiplications from $\lambda m/2$ to ρm, whereas the number of exponentiations is independent of m. In particular, for any specific number of statements m, we can find the best k that minimizes the number of group multiplications. We discuss the optimal choice of k and its effect on performance in Section 4.4.

In Figures 4.1 and 4.2, we visualise the various batching approaches. The squares represent PoE instances. All protocols start with m PoE instances represented by the black squares. The thickness of full arrows highlights the cost in group operations, i.e., thicker arrows correspond to less efficient aggregation. Dashed arrows represent computation of the final PoE(s).

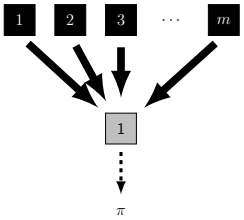
The soundness analysis. Our soundness analysis of the Bucket Protocol differs from that of Bellare et al. because they focus on prime-order groups, while we consider groups of hidden order. Our main technical tool is Lemma 5, which informally states that an adversary can only succeed in the following game with probability at most 1 over the randomness space unless it breaks the low order assumption: The adversary outputs a set of statements, at least one of which is incorrect. The statements are raised to random exponents and multiplied together, and the adversary wins if the resulting statement is correct.

Intuitively, if a statement $y \stackrel{?}{=} x^e$ is incorrect, then y can be viewed as the correct result multiplied by a "bad" element. Since every element in a group has an inverse, this factorization is always possible. Raising the statement to a random exponent r makes this bad element disappear from the equation only if r is a multiple of its order, an event that occurs with probability about one over the order.

Rotem [Rot21] proved a similar lemma, but his reduction from the low order assumption involves guessing the element's order, resulting in a quadratic loss in winning probability. We observe that the reduction can efficiently determine the order whenever the adversary's success probability is non-negligible, thereby preserving the adversary's original winning probability. Additionally, Rotem's lemma depends on the maximum order of low-order elements. Our analysis removes this dependency, showing it is unnecessary and clarifying the roles of the various parameters involved.

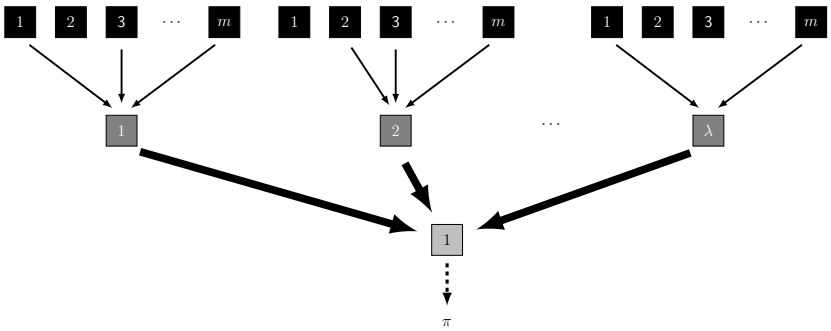


(a) Random Subsets Protocol [BHR+21, Rot21] (Figure 2.4a)

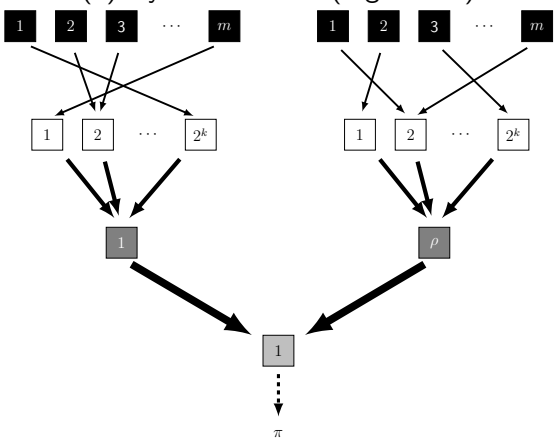


(b) Random Exponents Protocol [Rot21] (Figure 2.4b)

Figure 4.1: Depiction of the known batch PoEs.



(a) Hybrid Protocol (Figure 6.3)



(b) Bucket Protocol (Figure 4.4)

Figure 4.2: Depiction of the new batch PoEs.

Leveraging efficient multi-exponentiation. Group multi-exponentiation, i.e., evaluation of products of the form $\prod_{i=1}^{k} g_i^{e_i}$, has long been recognized as an important algorithmic task in cryptography. Various optimized algorithms for multi-exponentiation were proposed going

back to Straus [Str64] and Pippenger [Pip80] (see Möller [Möl01] and Henry [Hen10] and the references therein for a historical exposition). The most direct way of computing $\prod_{i=1}^{k} g_i^{e_i}$ via k exponentiations takes roughly λk group multiplications when the bit-length of the exponents is λ. The *asymptotically* optimal algorithm due to Pippenger reduces the multiplication cost by a multiplicative factor $\log(\lambda k)$, i.e., the dominating factor in the number of multiplications is $\frac{\lambda k}{\log(\lambda k)}$ instead of λk, which is clearly a major improvement.

However, all known multi-exponentiation algorithms induce a significant implementation overhead, and, in the end, the practical gains depend on many factors such as the specific hardware architecture, available parallelism, (non-trivial) choice of parameters for the algorithm, and the specific λ and k. For example, Attias et al. [AVD23] recently presented an experimental evaluation of the known multi-exponentiation algorithms on various architectures. In their experiments, they observed a maximal speedup by a factor 1.75 compared to the basic approach when measuring the *total time* of the multi-exponentiation.

When using any of the batching protocols in practice, one would (and should) likely experiment with all protocols, combining them with some efficient multi-exponentiation algorithm. Nevertheless, in the rest of the chapter, we measure the multiplication cost via the basic multiexponentiation algorithm. We stress that our aim is not to give an unfair advantage to our protocols in the comparisons. Instead, we wish to use a consistent complexity measure that seems as the most representative in predicting algorithmic bottlenecks while applying any of the batching protocols. We leave a deeper implementation study of our protocols for future work as it is out of the scope of the current work. Next, we discuss the effects of efficient multi-exponentiation algorithms on our protocols.

Our Hybrid Protocol cannot take significant advantage of advanced multi-exponentiation algorithms. The bulk of its computation is to evaluate products of random subsets of the m instances, and it applies the Random Exponents Protocol always only to λ instances (i.e., roughly 128 instances), which is too small to yield any significant practical improvement. Therefore, it would achieve smaller advantage over an implementation of the Random Exponents Protocol that uses the best known multi-exponentiation algorithm for the specific number of instances and architecture.

However, any improvement in the complexity of multi-exponentiation that would speed up the Random Exponents Protocol should provide similar gains also to our Bucket Protocol. Specifically, the parameter k in the Bucket protocol allows for a trade-off between the number of parallel repetitions $\rho = \lceil \lambda/(k-2) \rceil$ and the size of multi-exponentiations performed on the $K = 2^k$ buckets. Thus, faster multi-exponentiation would allow to increase the number of buckets and, correspondingly, decrease further the number of parallel repetitions.

4.1.2 Efficient remote benchmarking of parallelism from batch PoEs

In Section 4.5, we give a solution to the remote parallel benchmarking problem discussed in Section 4.1. Recall that the goal of the prover in such a scheme is to convince the verifier that it has m parallel processors. We show that any efficient PoE batching procedure $\texttt{Batch}$ can be used for this purpose based on the following observation. Suppose the prover solves m iterated squaring challenges $x_1, \ldots, x_m$ and knows the corresponding $y_1 = x_1^{2^T}, y_2 = x_2^{2^T}, \ldots, y_m = x_m^{2^T}$. Then, given an arbitrary batched challenge $\tilde{x} = \texttt{Batch}((x_1, \ldots, x_m); r)$ derived using randomness r (e.g., the exponents $r_1, \ldots, r_m$ in the Random Exponents Protocol), it can produce the correct $\tilde{y} = \tilde{x}^{2^T}$ fast in a much shorter time than T by running

the batching procedure on $y_1, \ldots, y_m$ and outputting $\tilde{y} = \texttt{Batch}((y_1, \ldots, y_m); r)$. On the other hand, if the prover does not know some of the y_i's then, intuitively, it needs time T to produce the correct $\tilde{y}$. This intuition gives rise to a simple protocol with optimal communication complexity. The verifier sends the prover m iterated squaring challenges and, after time T, fresh randomness r for the batching procedure defining an iterated squaring challenge $\tilde{x}$. The prover must send back the corresponding $\tilde{y}$ fast. Then, they use any PoE to verify the correctness of $\tilde{y}$ efficiently. Importantly, the prover sends only one group element $\tilde{y}$ and computes a single PoE. The verifier's complexity is the same as the complexity of the verifier in the chosen batching protocol as it only needs to compute $\tilde{x} = \texttt{Batch}((x_1, \ldots, x_m); r)$ and verify the PoE.

To understand the security guarantees of the above protocol, we need to understand the hardness of solving m parallel repeated squaring instances, which, to the best of our knowledge, has not been considered in the literature. Note that, in our setting, the verifier must distinguish a prover with m processors from anyone with $m-1$ processors or less. In particular, there must be a significant gap between solving the m instances on m machines, which takes time T, compared to solving the m instances on $m-1$ machines. Since the instances are uniformly random, we assume that the computation performed for one instance cannot be reused for speeding up other instances.[2] Then a trivial lower bound for solving the m instances on $m-1$ machines is $T + T/(m-1)$: If we disregard the sequentiality of the task, then we still need to divide mT steps among $(m-1)$ machines. Is there a better lower bound for m *sequential* tasks? Upon a closer inspection, one realizes that we are facing a generalization of the "three toasts" puzzle, asking to toast three slices of bread from both sides as fast as possible on a pan that can fit only two slices at a time (see, e.g., "Quick and Toasty" in Mason, Burton, and Stacey [MBS82]). The natural approach is to toast two slices on one side then flip the first and swap the second for the third. After finishing toasting the first, we can take it out and use its slot for finishing toasting the second simultaneously with the third one. It is trivial to extend the same approach to m instances of repeated squaring on $m-1$ processors, matching the lower bound $T + T/(m-1)$. Thus, we can only assume that solving m repeated squaring instances with delay parameter T on $m-1$ processors takes time at least $T + T/(m-1)$.

To prove the security of our benchmarking protocol, we additionally rely on a recent generalization of the iterated squaring assumption introduced in [HP25]. Their generalization states that the sequential hardness of iterated squaring is preserved even for the task of computing $y = (x^r)^{2^T}$, where $r \neq 0$ is chosen by the prover adaptively after seeing the challenge x and the delay parameter T. The formal definition of our *generalized parallel iterated squaring assumption* incorporating the above consideration about solving parallel iterated squaring instances in the assumption from [HP25] is presented in Definition 19.

Note that, to some extent, the remote benchmarking problem could be addressed using Proof of Work (PoW) by setting the hardness of the PoW puzzle to match the claimed parallelism of the prover. However, such a scheme would only measure the hash-rate and would not necessarily prove much about the parallelism of the prover. A prover using specialized hardware for fast hashing to convince the verifier in a PoW-based scheme is unlikely to be of much use to the verifier. On the other hand, the ability to perform group exponentiations fast in parallel attested via our protocol would be of interest for example when one wishes to outsource proving in modern zk-SNARKs relying on variants of the KZG polynomial commit-

[2] Since $m, T \in \textbf{polylog}(\mathrm{ord}(\mathbb{G}))$, this seems like a reasonable assumption. It even holds, e.g., in idealized models such as the generic group model [Sho97, Mau05].

ment [KZG10], where it is estimated that the majority of proving time is spent performing Multi-Scalar Multiplication (i.e., group multi-exponentiations)[Xav22].

Additionally, the probabilistic nature of PoW comes with various downsides. For example, even if the prover has the claimed parallel capacity, it might be unlucky and not find a good enough PoW solution within the time limit, leading to increased latency of such schemes. In our solution, any honest prover is guaranteed to always succeed in convincing the verifier within the prescribed time if it has the claimed parallelism.

4.1.3 More Related Work

Batch verification of digital signatures. Fiat [Fia97] initiated a related line of work on batch verification of many digital signatures. See Camenish, Hohenberger, and Pedersen [CHP12] for a historical overview of the known results. Bellare, Garay, and Rabin [BGR98] presented a set of protocols for batch verification of modular exponentiation in prime-order groups. For a generator g of a group $\mathbb{G}$, they consider the problem of verifying statements of the form $y_1 \stackrel{?}{=} g^{e_1}, \ldots, y_m \stackrel{?}{=} g^{e_m}$. Importantly, they work in groups with a publicly known order, and wish to batch verify exponentiations with a fixed base and varying exponents. However, their protocols can also be adapted to the setting relevant for VDFs based on PoEs. Di Crescenzo et al. [CKKS17] extended the small exponent protocol of [BGR98] to the setting of safe prime RSA groups. However, their proof size is linear in the number of statements to be batched.

Performance of PoEs in Practice. Attias, Vigneri, and Dimitrov [AVD20] reported on the practical performance of verification for two PoEs in the context of Verifiable Delay Functions in RSA groups. The verification time for Wesolowski's VDF can be further optimized in RSA groups, e.g., as suggested by Attias et al. [AVD22].

4.2 The Hybrid Protocol

In this section, we present our first protocol that improves the number of multiplications over the Random Exponents Protocol (Figure 2.4b). We exploit the fact that the Random Subsets Protocol (Figure 2.4a) results in λ instances that can be securely batched under computational hardness assumptions. The resulting Hybrid Protocol (Figure 4.3) reduces the number of PoEs needed to one while applying the random exponents technique solely to λ instances, i.e., the complexity of the more costly technique is made independent of the number of batched instances. Its completeness follows by inspection of the protocol. In the rest of the section, we prove its soundness based on the low order assumption (Definition 9).

Theorem 4. *Let* $\mathtt{PoE}$ *be a proof of exponentiation with soundness error* γ *and let* $\mathbb{G}$ *be a group output by* $\mathtt{GGen}(\lambda)$*. Assuming the low order assumption for* $\mathtt{GGen}$ *with soundness error* μ*, the hybrid batching protocol presented in Figure 4.3 has soundness error at most* $\gamma + \mu + 2^{-\rho} + 2^{-\lambda} + 2^{-\kappa}$.

Before proving Theorem 4, we state two lemmas we use in the proof. For the proof of Lemma 4, see Block et al. [BHR+21].

Lemma 4 ([BHR+21, Fact 8.1])**.** *For any group* $\mathbb{G}$ *and any* $x_1, \ldots, x_n \in \mathbb{G}$*, where at least one of them is not the identity element, we have* $\Pr_{S \leftarrow 2^{[n]}}\left[\prod_{i \in S} x_i = 1\right] \leq \frac{1}{2}$.

Parameters:

- group $\mathbb{G}$, exponent e, number of statements m, number of repetitions of subset multiplications ρ, size of random coins κ, a proof of exponentiation PoE

Statements: $\left\{y_i \stackrel{?}{=} x_i^e\right\}_{i\in[m]}$ in $\mathbb{G}$

Protocol:

1. $\mathcal{V}$ samples a matrix $B \leftarrow \{0,1\}^{\rho\times m}$ and a vector $r \leftarrow [2^\kappa]^\rho$ uniformly at random and sends both to $\mathcal{P}$.
2. $\mathcal{V}$ and $\mathcal{P}$ both construct new statements $\left\{y'_i \stackrel{?}{=} (x'_i)^e\right\}_{i\in[\rho]}$, where
$$y'_i = \prod_{j\in[m]} y_j^{B_{i,j}} \quad \text{and} \quad x'_i = \prod_{j\in[m]} x_j^{B_{i,j}}.$$
3. $\mathcal{V}$ and $\mathcal{P}$ both construct one new statement $\tilde{y} \stackrel{?}{=} (\tilde{x})^e$, where
$$\tilde{y} = \prod_{i\in[\rho]} (y'_i)^{r_i} \quad \text{and} \quad \tilde{x} = \prod_{i\in[\rho]} (x'_i)^{r_i}.$$
4. $\mathcal{V}$ and $\mathcal{P}$ run PoE on statement $\tilde{y} \stackrel{?}{=} (\tilde{x})^e$.

Figure 4.3: Our Hybrid Protocol.

The second lemma is similar to [Rot21, Lemma 6.2] but the proof differs. Importantly, Rotem's reduction incurs a quadratic security loss, and our reduction is tight.

Lemma 5. *Let $\mathbb{G}$ be a group and $n, e, \kappa \in \mathbb{N}$ arbitrary positive integers. Let $\mathcal{A}$ be an algorithm that runs in time t and, with probability ε, outputs group elements $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathbb{G}$ such that, for at least one $i \in [n]$, we have $x_i^e \neq y_i$ and*

$$\Pr_{r_1,\ldots,r_n\leftarrow[2^\kappa]}\left[\prod_{i\in[n]} (x_i^{r_i})^e = \prod_{i\in[n]} y_i^{r_i}\right] = \frac{1}{2^\kappa} + \delta.$$

Then there exists an algorithm $\mathcal{B}$ that runs in time $\mathrm{poly}(t, \log(e), n, \log(\delta^{-1})\delta^{-1})$ and, with probability ε, outputs an element of order at most δ^{-1} in $\mathbb{G}$ together with its order.

Proof. We let $\mathcal{B}$ run $\mathcal{A}$ and find one pair (x_{i^*}, y_{i^*}) for which $x_{i^*}^e \neq y_{i^*}$ among the group elements output by $\mathcal{A}$. This takes time $\mathrm{poly}(t, \log(e), n)$. Set $z = x_{i^*}^e / y_{i^*}$. We claim that z has order at most δ^{-1}. This means that $\mathcal{B}$ can find the order in time $\mathrm{poly}(\log(\delta^{-1})\delta^{-1})$ and output z together with its order. Clearly, $\mathcal{A}$ and $\mathcal{B}$ have the same success probability. It remains to prove the bound on the order of z. Let $(x_{i_1}, y_{i_1}), \ldots, (x_{i_s}, y_{i_s})$ be the pairs for which $x_{i_j}^e \neq y_{i_j}$. Assume without loss of generality that $x_{i_j}^e = y_{i_j} z_j$ for some group elements $z_1, \ldots, z_s \in \mathbb{G}$. Note that one of those elements is equal to z and denote the corresponding

index by j^*. We have

$$\frac{1}{2^\kappa} + \delta = \Pr\left[\prod_{i\in[n]} (x_i^{r_i})^e = \prod_{i\in[n]} y_i^{r_i}\right] \tag{4.1}$$

$$= \Pr\left[\prod_{j\in[s]} \left(x_{i_j}^{r_{i_j}}\right)^e = \prod_{j\in[s]} y_{i_j}^{r_{i_j}}\right] \tag{4.2}$$

$$= \Pr\left[\prod_{j\in[s]} (y_{i_j} z_j)^{r_{i_j}} = \prod_{j\in[s]} y_{i_j}^{r_{i_j}}\right] \tag{4.3}$$

$$= \Pr\left[\prod_{j\in[s]} z_j^{r_{i_j}} = 1\right] = \Pr\left[z^{r_{i^*}} = \prod_{j\in[s], j\neq j^*} z_j^{-r_{i_j}}\right], \tag{4.4}$$

where the probability is taken over $r_1, \ldots, r_n \leftarrow [2^\kappa]$. The first equation holds by assumption. In Equation (4.2), we only take the product over the incorrect statements, since the correct statements can be cancelled out in the equation. Equation (4.3) holds by definition of z_j's. Equation (4.4) follows from first cancelling out correct statements and then moving all elements z_j different from z to the other side of the equation. The element on the right side of the equation is either an element of the subgroup generated by z or not. If it is not an element of the subgroup, the probability of the event is 0. Otherwise, let $\alpha < \mathrm{ord}(z)$ be such that $\prod_{j\in[s], j\neq j^*} z_j^{-r_{i_j}} = z^\alpha$ and let s be the largest integer such that $s \cdot \mathrm{ord}(z) \leq 2^\kappa$. Denote $\Pr := \Pr_{r_{i^*}\leftarrow[2^\kappa]}$. Then we have

$$\begin{aligned}
(4.4) &\leq \Pr\left[z^{r_{i^*}} = z^\alpha\right] \\
&= \Pr\left[z^{r_{i^*}} = z^\alpha \mid r_{i^*} \leq s \cdot \mathrm{ord}(z)\right] \cdot \Pr\left[r_{i^*} \leq s \cdot \mathrm{ord}(z)\right] \\
&\quad + \Pr\left[z^{r_{i^*}} = z^\alpha \mid r_{i^*} > s \cdot \mathrm{ord}(z)\right] \cdot \Pr\left[r_{i^*} > s \cdot \mathrm{ord}(z)\right] \\
&\leq \frac{s \cdot \mathrm{ord}(z)}{2^\kappa} \cdot \frac{1}{\mathrm{ord}(z)} + \frac{2^\kappa - s \cdot \mathrm{ord}(z)}{2^\kappa} \cdot \frac{1}{2^\kappa - s \cdot \mathrm{ord}(z)} \\
&\leq \frac{1}{\mathrm{ord}(z)} + \frac{1}{2^\kappa}.
\end{aligned}$$

Hence, we obtain that $2^{-\kappa} + \delta \leq 1/\mathrm{ord}(z) + 2^{-\kappa}$ and the claim follows. □

Proof of Theorem 4. Assume that at least one of the initial statements $\left\{y_i \stackrel{?}{=} x_i^e\right\}_{i\in[m]}$ is incorrect. An adversary that tries to break soundness of the protocol needs to be successful in one of the Steps 2,3,4, which means that either in Step 2 or 3 it starts with at least one wrong statement and at the end of the step all of the statements are correct or it breaks the soundness of the PoE in Step 4. By Lemma 4, we have that after Step 2 of the protocol at least one of the statements $\left\{y_i' \stackrel{?}{=} (x_i')^e\right\}_{i\in[\rho]}$ is incorrect except with probability at most $1/2^\rho$. Assume that at least one of those statements is incorrect and consider Step 3. Applying Lemma 5 to the statements $\left\{y_i' \stackrel{?}{=} (x_i')^e\right\}_{i\in[\rho]}$, we get that $\tilde{y} \stackrel{?}{=} (\tilde{x})^e$ is incorrect except with probability at most $\mu + 2^{-\lambda} + 2^{-\kappa}$. Otherwise, one could find an element of order smaller than 2^λ with probability larger than μ, which contradicts the low order assumption. In Step 4 of the protocol, $\mathcal{V}$ and $\mathcal{P}$ run a PoE with soundness error γ. Taking the union bound, we get that the soundness error of the protocol is at most $\gamma + \mu + 2^{-\rho} + 2^{-\lambda} + 2^{-\kappa}$ assuming the μ-low order assumption. □

As we have just seen in the proof of Theorem 4, the summand $2^{-\lambda}$ in our upper bound in the soundness error of the Hybrid Protocol comes from the assumption that it is hard to find elements of order less than 2^{λ} (see Definition 9). Note that a weaker variant of the low order assumption would suffice to get meaningful security. To weaken the assumption, one can restrict the bound on the order of elements the adversary can output while winning the game, for example, to $\lambda^{\log \lambda}$. This would increase the corresponding term in the bound on the soundness error from $2^{-\lambda}$ to $\lambda^{-\log \lambda}$, which is, however, still negligible. Similarly, a weaker variant of the low order assumption suffices also for our soundness analysis of the Bucket Protocol presented next. For simplicity of presentation, we stick to the bound of $2^{-\lambda}$ in the low order assumption throughout the rest of the chapter.

4.3 The Bucket Protocol

In this section, we present an adaptation of the bucket test of Bellare, Garay, and Rabin [BGR98] to the setting of batch proofs of exponentiation. The protocol is presented in Figure 4.4 and illustrated in Figure 4.2b. Starting with m statements, the following process is repeated $\rho = \lceil \lambda/(k-2) \rceil$ times: The prover and verifier uniformly at random place the m statements into $K = 2^k$ buckets and then compute the product of the statements in each bucket to obtain K new statements, which are aggregated to a single statement using the Random Exponents Protocol with k-bit exponents. After the ρ repetitions, the prover and verifier aggregate the resulting ρ statements using the Random Exponents Protocol with λ-bit exponents, and they run a proof of exponentiation on the final statement. This last aggregation step differs from the bucket test in [BGR98], where the ρ instances are verified directly in parallel instead of aggregating them to a single statement.

Note that both the prover and verifier must run in polynomial time in λ, and, thus, they could not keep track of 2^{λ} buckets. Therefore, without loss of generality, we use 2^{λ} as an upper bound on the number of buckets $K = 2^k$ in the proof of the following theorem.

Theorem 5. *Let* $\mathtt{PoE}$ *be a proof of exponentiation with soundness error* γ *and let* $\mathbb{G}$ *be a group output by* $\mathtt{GGen}(\lambda)$*. Assuming the low order assumption for* $\mathtt{GGen}$ *with soundness error* $\mu \leq 2^{-k} - 2^{-\lambda}$*, the bucket batching protocol presented in Figure 4.4 has soundness error at most* $\gamma + \mu + 2^{-\lambda+1} + 2^{-\kappa}$*.*

Proof. Suppose that there is at least one incorrect statement among $\left\{ y_i \stackrel{?}{=} x_i^e \right\}_{i \in [m]}$. An adversary that tries to break the soundness of the protocol needs to be successful in one of the Steps 2,3,4,5, which means that either in Step 2, 3 or 4 it starts with at least one wrong statement and at the end of the step all of the statements are correct or it breaks the soundness of the $\mathtt{PoE}$ in Step 5.

Following the analysis in [BGR98], we show that after Steps 2 and 3 at least one of the statements $\left\{ y_i'' \stackrel{?}{=} (x_i'')^e \right\}_{i \in [\rho]}$ is incorrect except with probability $2^{-\lambda}$. Fix one round $i \in [\rho]$ and assume that all statements have been assigned to a bucket except for one incorrect statement. We call a bucket *good* if the product of all statements in that bucket yields a correct statement. Otherwise, we call it *bad*. The event that all K buckets are good after round i can only occur if all but one bucket so far are good: If more than one bucket is bad, at least one of them cannot become good after the final missing statement is assigned. If all buckets are good, assigning the final missing statement will make its bucket bad. In order for

Parameters:

- group $\mathbb{G}$, common exponent e, number of statements m, number of buckets $K = 2^k$, where $k \leq \lambda$, number of repetitions of bucketings $\rho = \lceil \lambda/(k-2) \rceil$, size of random coins κ, a proof of exponentiation PoE

Statements: $\left\{ y_i \stackrel{?}{=} x_i^e \right\}_{i \in [m]}$ in $\mathbb{G}$

Protocol:

1. $\mathcal{V}$ samples two matrices $B \leftarrow [K]^{\rho \times m}$ and $R \leftarrow [K]^{\rho \times K}$ and a vector $r \leftarrow [2^\kappa]^\rho$ uniformly at random and sends both to $\mathcal{P}$.
2. $\mathcal{V}$ and $\mathcal{P}$ both construct new statements $\left\{ y'_{i,b} \stackrel{?}{=} (x'_{i,b})^e \right\}_{i \in [\rho], b \in [K]}$, where
$$y'_{i,b} = \prod_{j \in [m], B_{i,j} = b} y_j \quad \text{and} \quad x'_{i,b} = \prod_{j \in [m], B_{i,j} = b} x_j.$$
3. $\mathcal{V}$ and $\mathcal{P}$ both construct new statements $\left\{ y''_i \stackrel{?}{=} (x''_i)^e \right\}_{i \in [\rho]}$, where
$$y''_i = \prod_{b \in [K]} (y'_{i,b})^{R_{i,b}} \quad \text{and} \quad x''_i = \prod_{b \in [K]} (x'_{i,b})^{R_{i,b}}.$$
4. $\mathcal{V}$ and $\mathcal{P}$ both construct one new statement $\tilde{y} \stackrel{?}{=} (\tilde{x})^e$, where
$$\tilde{y} = \prod_{i \in [\rho]} (y''_i)^{r_i} \quad \text{and} \quad \tilde{x} = \prod_{i \in [\rho]} (x''_i)^{r_i}.$$
5. $\mathcal{V}$ and $\mathcal{P}$ run PoE on statement $\tilde{y} \stackrel{?}{=} (\tilde{x})^e$.

Figure 4.4: Our Bucket Protocol based on the bucket test from Bellare et al. [BGR98].

all buckets to be good in the end, we at least need the final missing statement to fall into the only bad bucket. This occurs with probability $1/K = 2^{-k}$. Now consider Step 3. Applying Lemma 5 to statements $\left\{ y'_{i,b} \stackrel{?}{=} (x'_{i,b})^e \right\}_{b \in [K]}$ for a fixed $i \in [\rho]$, where at least one of the statements is incorrect, we get that the resulting statement $y''_i \stackrel{?}{=} (x''_i)^e$ is incorrect except with probability at most $\mu + 2^{-\lambda} + 2^{-k}$. Otherwise, one could find an element of order smaller than 2^λ with probability larger than μ, which contradicts the low order assumption. Taking the union bound, we get that at least one of the statements $\left\{ y''_i \stackrel{?}{=} (x''_i)^e \right\}_{i \in [\rho]}$ is wrong except with probability at most $(\mu + 2^{-\lambda} + 2^{-k} + 2^{-k})^\rho \leq (2^{-k+2})^\rho \leq 2^{-\lambda}$ since $\mu \leq 2^{-k} - 2^{-\lambda}$ by assumption. Assume that at least one of those statements is incorrect and consider Step 4. Applying Lemma 5 to the statements $\left\{ y''_i \stackrel{?}{=} (x''_i)^e \right\}_{i \in [\rho]}$, we get that $\tilde{y} \stackrel{?}{=} (\tilde{x})^e$ is incorrect, except with probability at most $\mu + 2^{-\lambda} + 2^{-\kappa}$. Otherwise, one could find an element of order smaller than 2^λ with probability larger than μ, which contradicts the low order assumption. In

Table 4.1: The complexity of various batch PoEs for m instances with security parameter λ, and 2^k buckets in the Bucket Protocol.

Protocol	# multiplications	# proofs
No batching	0	m
Random Subsets	λm	λ
Random Exponents	$(3\lambda+2)m$	1
Hybrid	$\lambda(m+3\lambda+2)$	1
Bucket	$\left\lceil \frac{\lambda}{k-2} \right\rceil \left(2m+(3k+2)2^k+(3\lambda+2)\right)$	1

Step 5 of the protocol $\mathcal{V}$ and $\mathcal{P}$ run a $\texttt{PoE}$ with soundness error γ. Taking the union bound, we get that the soundness error of the protocol is at most $\gamma+\mu+2^{-\lambda+1}+2^{-\kappa}$. □

4.4 Comparison

A comparison of no batching and the four batching approaches is summarised in Table 4.1. The number of proofs to verify is clear in all approaches. Next, we explain the expected number of multiplications for each approach. This computation excludes the multiplication cost of the verification of the PoE proof(s) as that depends on the specific PoE.

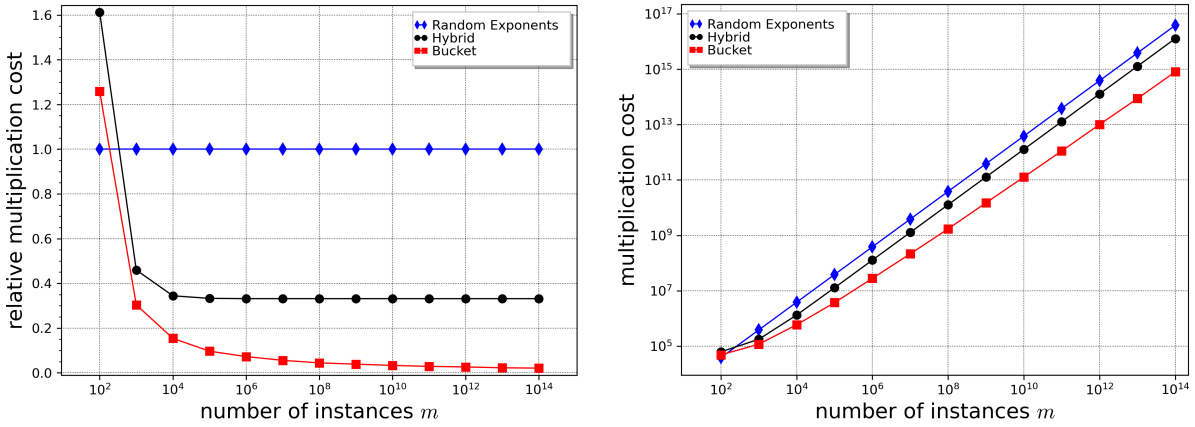
Random Subsets: In expectation, the Random Subsets Protocol selects subsets of $[m]$ of size $m/2$, and, thus, it computes two products of size $m/2$ per each of the λ parallel repetitions.

Random Exponents: The m instances are raised to random λ-bit exponents, which is performed using $1.5\lambda \cdot 2m$ multiplications in expectation. Finally, it needs to compute two products of m group elements (to construct the resulting x_i''s and y_i''s).

Hybrid Protocol: In expectation, the protocol in Figure 6.3 performs the λm multiplications as in the Random Subsets Protocol. Then, it applies the Random Exponents Protocol to the resulting λ instances.

Bucket Protocol: The protocol in Figure 4.4 performs $\rho = \lceil \lambda/(k-2) \rceil$ repetitions. In total, it takes $2\rho m$ multiplications to produce the instances corresponding to the buckets as, in each repetition, every instance participates in exactly one bucket. Then, in each repetition, the $K = 2^k$ bucket instances are aggregated using the Random Exponents Protocol with coins of size k. Finally, the resulting ρ instances are merged using the Random Exponents Protocol with coins of size λ.

Note that the approaches are ordered on the basis of their efficiency. The Random Exponents Protocol increases the number of multiplications compared to the Random Subsets Protocol. However, it is the first protocol with a single PoE proof. The Hybrid Protocol achieves the same number of multiplications as the Random Subsets compiler up to an additive overhead independent of the number of instances m. Finally, the Bucket Protocol is parameterized by the number of buckets and enables a trade-off between the number of multiplications that depend on the number of instances and that are independent of the number of instances.



(a) (b)

Figure 4.5: The relative (4.5a) and absolute (4.5b) numbers of multiplications on 100 to 10^{14} instances compared to the random exponent batching approach from Rotem [Rot21] with the security parameter $\lambda = 128$ and optimal choice of k in the Bucket Protocol.

Table 4.2: The measured times (in seconds) for PoE batching approaches in a 2048-bit RSA group. The last row is extrapolated for the first two approaches due to excessive times.

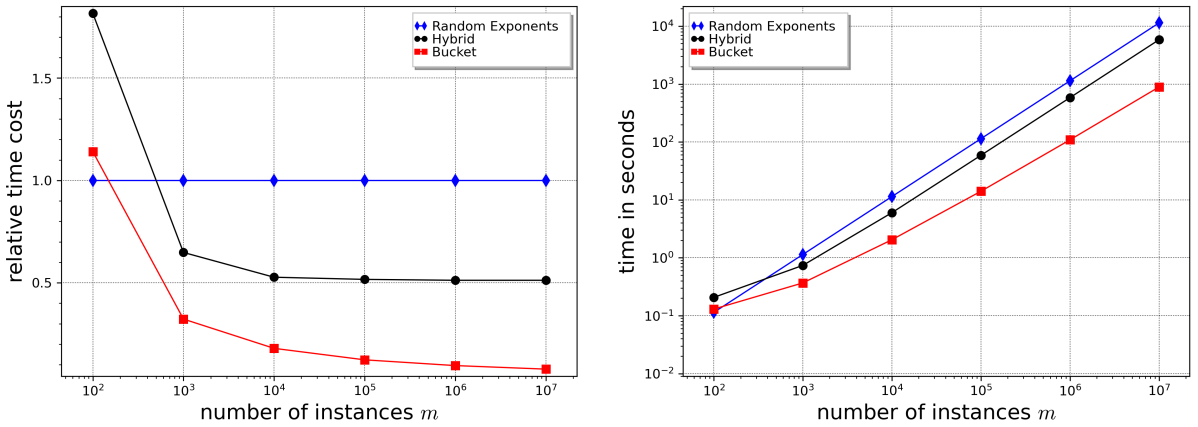
# instances	Random Exponents	Hybrid	Bucket
100	0.114	0.207	0.130
1 000	1.14	0.739	0.368
10 000	11.4	6.01	2.05
100 000	114	58.9	14.1
1 000 000	1140	584	109
10 000 000	*11400*	*5840*	892

For $\lambda = 128$ and varying m, Figures 4.5a and 4.5b show the relative and total group multiplications. For the Bucket Protocol, we use the optimal k for each m. At $m = 1{,}000$, both the Hybrid and Bucket Protocols significantly reduce expected multiplications compared to the Random Exponents Protocol. The Hybrid Protocol achieves a roughly threefold decrease at tens of thousands of instances, as expected based on Table 4.1. With varying k, the gap between the Random Exponents and Bucket Protocols grows. The Bucket Protocol achieves a threefold improvement at $m = 1{,}000$ and reduces the expected number of multiplications by an order of magnitude at $m = 100{,}000$.

4.4.1 Experimental Evaluation

In this section, we present experimental results that compare the practical performance of known approaches for PoE batching.

We have implemented all approaches in SageMath 10.1. We have timed the performance on a machine with a four core 3.60GHz Intel Xeon CPU E5-1620 0 processor with 64 GB of RAM. For our experiments, we generated 10M random PoE instances of the form $y = x^{2^{25}}$ with a common 2048-bit RSA modulus (roughly 12 GB of data). We then timed the performance of the Random Exponents Protocol, Hybrid Protocol, and Bucket Protocol using the SageMath timeit function. The timing results on multiples of ten from one hundred to ten million



(a) (b)

Figure 4.6: The relative (4.6a) and absolute (4.6b) time performance on 100 to 10M instances compared to the random exponent batching approach from Rotem [Rot21], with the security parameter $\lambda = 128$ and optimal choice of k in the Bucket Protocol.

instances are presented in Table 4.2 and Figures 4.6a and 4.6b.

When comparing Figures 4.5a and 4.6a, we see that the speed-up of the Bucket Protocol behaves as expected from the theoretical analysis: already at a thousand instances, we see a significant speed-up, which, with a growing number of instances m quickly converges to a speed-up of one order of magnitude. However, the speed-up of the Hybrid Protocol is smaller than one could expect based solely on the number of group multiplications. It is roughly twice as fast as the Random Exponents protocol, whereas the number of multiplications is almost reduced by a factor of three. We conjecture that this is due to the complexity of creating the subsets in the protocol's first step. In our implementation, we loop through λ many uniformly random bit strings of length m and always check if the current bit b is 0 or 1. We leave optimizing the implementation open for future work.

4.5 Communication-Optimal Remote Parallel Benchmarking via Batch PoEs

In this section, we construct an efficient proof of sufficient parallel resources using batch PoEs. With our protocol, a prover can convince a verifier that it has enough resources to solve m repeated squaring instances in parallel. The proof consists of one group element and one PoE. In particular, its size is independent of m. The protocol is presented in Figure 4.7. It can be instantiated with any PoE `PoE` and any batching protocol `Batch` presented in this chapter that only requires the verifier to check one PoE.

The verifier chooses m random elements $x_1, \ldots, x_m$ from the hidden order group $\mathbb{G}$ and sends them to the prover, along with a time parameter T. The verifier then waits for time T while the prover computes $y_i = x_i^{2^T}$ in parallel for all i. At time T, the verifier sends a random value r to define a batching challenge and starts a new timer. Using r, the prover aggregates the y_i's into a single value $\tilde{y}$ and returns it to the verifier. Upon receiving $\tilde{y}$, the verifier stops the timer and checks if the elapsed time is less than t for some $t \ll T$. If the prover took too long, the verifier rejects. If the timing check passes, both parties use r to locally combine the x_i's into a single instance $\tilde{x}$. Finally, the prover sends a PoE proof for the statement $\tilde{y} \stackrel{?}{=} \tilde{x}^{2^T}$,

Parameters:

- group $\mathbb{G}$, time parameter T, number of statements m, another time parameter $t < T/(m-1)$, a proof of exponentiation `PoE`, a PoE batching protocol `Batch`

Protocol:

1. $\mathcal{V}$ samples m group elements $x_1, \ldots, x_m \leftarrow \mathbb{G}$, sends them to $\mathcal{P}$, and starts a timer.
2. $\mathcal{P}$ computes $y_i = x_i^{2^T}$ for all $i \in [m]$ in parallel.
3. After time T, $\mathcal{V}$ samples randomness r for `Batch` and sends it to $\mathcal{P}$.
4. $\mathcal{P}$ computes $(\tilde{x}, \tilde{y}) = \texttt{Batch}((x_1, \ldots, x_m), (y_1, \ldots, y_m); r)$ defining the statement $\tilde{y} \stackrel{?}{=} \tilde{x}^{2^T}$ and sends $\tilde{y}$ to $\mathcal{V}$.
5. $\mathcal{V}$ stops the timer and checks that it is less than $T + t$. Otherwise $\mathcal{V}$ rejects.
6. $\mathcal{V}$ computes $\tilde{x} = \texttt{Batch}((x_1, \ldots, x_m); r)$ locally and $\mathcal{V}$ and $\mathcal{P}$ run `PoE` on statement $\tilde{y} \stackrel{?}{=} \tilde{x}^{2^T}$.

Figure 4.7: A proof of sufficient resources to solve m repeated squaring instances in time less than $T + t$.

and the verifier then accepts or rejects based on the correctness of the PoE.

By soundness of the PoE, we have that if the prover sends a valid $\tilde{y}$ after receiving r in time less than t, then (except with a negligible probability) it has computed all m iterated squaring instances in time less than $T + t$. In particular, if t is small enough, this shows that the prover has enough computing power to solve m repeated squaring instances in parallel. The proof size is only one group element plus the size of the PoE. The verifier's complexity is the same as the complexity of the verifier in the chosen batching protocol.

Before we prove soundness of the protocol, we make some remarks on how to optimize and adapt the protocol for different applications:

To optimize communication complexity of the protocol, we can replace the m group elements sent by the verifier with a uniformly sampled key k for a pseudo-random function. With this key k, both the prover and the verifier can sample the challenges $x_1, \ldots, x_m$ locally without further interaction.

To remove interaction, we can replace the second message from the verifier using a randomness beacon b. This beacon b simultaneously serves as the source for the randomness of the challenge r of the batching protocol and as the starting of the timer. If the prover sends $\tilde{y}$ in time less than t after the beacon was released and the PoE proof is valid, the verifier accepts the proof. Note that, in order for the proof to remain publicly verifiable after time $T + t$, the message $\tilde{y}$ from the prover needs to be timestamped.

In some applications, public verifiability of the proof of resources is not necessary. For example, if a company wants to privately benchmark the computing power of prospective contractors paid for outsourced computation. In this case, we can adapt the protocol as follows: In the first step, the verifier $\mathcal{V}$ constructs an RSA group with a trapdoor, samples the m instances

from this group and sends the instances and the group to $\mathcal{P}$. Steps 2-5 remain the same. In Step 6, the verifier can efficiently check that $\tilde{y}$ is the correct element using the trapdoor of the group, avoiding the need for the final PoE

Before we prove the security of our protocol in Section 4.5.2, we recall a result by Attema, Cramer, and Kohl [ACK21] we use in our proof. The presentation follows the exposition from [BDH+25].

4.5.1 An efficient tree sampling algorithm by [ACK21]

To prove the knowledge soundness of interactive protocols, a *tree of transcripts* is often helpful. Attema, Cramer, and Kohl [ACK21] showed how to sample such a tree efficiently.

Definition 12 (tree of transcripts)**.** A $(\alpha_1, \ldots, \alpha_n)$-*tree of transcripts* for an n-round interactive protocol is a set of $\prod_{i=1}^{n} \alpha_i$ transcripts that can be arranged in the following tree structure. Consider transcripts of the form $(x, c_1, \ldots, c_n, a_1, \ldots, a_n)$, where x denotes the instance, $c_1, \ldots, c_n$ the verifier's challenges, and $a_1, \ldots, a_n$ the prover's messages. Each transcript corresponds to exactly one path from the root node to a leaf node. The tree's root is labeled by the instance x, the remaining vertices on the path to a leaf are labeled by $a_1, \ldots, a_n$, where the vertex at level $i+1$ gets the label a_i. The edges on the path are labeled by $c_1, \ldots, c_n$, where the edge connecting the vertices on levels i and $i+1$ are labeled by c_i. Each vertex at level i has precisely α_i children, corresponding to α_i distinct choices for c_i that label the edges.

Attema et al. reduced the problem of extracting a tree of transcripts for an interactive protocol to a slightly more general game about finding a "tree of 1-entries" in a 0/1-tensor, which we describe next. For an n-round protocol, set R as the size of the prover's randomness space and N as the size of the verifier's challenge space and fix a statement x. The $(n+1)$-dimensional tensor $H_x \in \{0,1\}^{R \times N \times \cdots \times N}$ is then defined such that, for any fixed randomness $r \in [R]$ and any vector of challenges $c_1, \ldots, c_n \in [N]$, the entry $H_x[r, c_1, \ldots, c_n]$ equals 1 if and only if the verifier eventually accepts the corresponding transcript. Then, a $(\alpha_1, \ldots, \alpha_n)$-tree of 1-entries in H_x corresponds to a $(\alpha_1, \ldots, \alpha_n)$-tree of accepting transcripts for the protocol. We present the tree sampling algorithm TreeFind of Attema, Cramer, and Kohl [ACK21] in Algorithm 2 that achieves the following.

Lemma 6 ([ACK21])**.** *For any $n, R, N \in \mathbb{Z}_{>0}$, let $H \in \{0,1\}^{R \times N \times \cdots \times N}$ be an $(n+1)$-dimensional tensor and let $\varepsilon \in \mathbb{R}$ be the fraction of 1-entries in H. The algorithm* $\mathsf{TreeFind}^H(\alpha_1, \ldots, \alpha_n)$ *outputs a $(\alpha_1, \ldots, \alpha_n)$-tree of 1-entries in H with probability at least*

$$\varepsilon - \frac{1}{N} \sum_{i=1}^{n} (\alpha_i - 1).$$

The expected sample complexity of $\mathsf{TreeFind}^H(\alpha_1, \ldots, \alpha_n)$ *is at most $\prod_{i=1}^{n} \alpha_i$.*

Remark 3 (On sampling a tree of 1-round transcripts with structured challenges)**.** In their result, Attema, Cramer, and Kohl consider n round protocols, i.e., protocols in which the verifier sends n challenges that are each answered individually by a prover message. However, note that the algorithm TreeFind is oblivious to the number of prover's messages. It only queries the tensor H and thus only learns whether or not the verifier accepts at the end of the protocol. We can therefore use Algorithm 2 to obtain a tree of 1-round transcripts with structured challenges as follows: Let the verifier challenge of the protocol be a vector

Algorithm 2 The tree-finding algorithm TreeFind from [ACK21].

TreeFind$^H(\alpha_1, \ldots, \alpha_n)$:

1: $a \leftarrow R$
2: return TreeFind$_0^H(a)$ with the root labeled by a

TreeFind$_n^H(a, x_1, \ldots, x_n)$:

1: **if** $H[a, x_1, \ldots, x_n] = 1$ **then**
2: return new leaf labeled by $H[a, x_1, \ldots, x_n]$
3: **else**
4: return $\perp$
5: **end if**

TreeFind$_i^H(a, x_1, \ldots, x_i)$: ▷ for $i \in \{0, \ldots, n-1\}$

1: $x_{i+1} \leftarrow N$
2: v a new vertex
3: $\mathcal{T} =$ TreeFind$_{i+1}^H(a, x_1, \ldots, x_{i+1})$
4: **if** $\mathcal{T} = \perp$ **then**
5: return $\perp$
6: **else**
7: connect v and the root of $\mathcal{T}$ by an edge labeled by x_{i+1}
8: **while** v has $< \alpha_i$ children **do**
9: $x_{i+1} \leftarrow N$ (without replacement)
10: $\mathcal{T} =$ TreeFind$_{i+1}^H(a, x_1, \ldots, x_{i+1})$
11: **if** $\mathcal{T} \neq \perp$ **then**
12: connect v and the root of $\mathcal{T}$ by an edge labeled by x_{i+1}
13: **end if**
14: **if** all possibilities were tried **then**
15: return $\perp$
16: **end if**
17: **end while**
18: **end if**

of the form $s = (s_1, \ldots, s_m) \in [S]^m$ for some $S \in \mathbb{Z}_{>0}$ and let $R \in \mathbb{Z}_{>0}$ be the size of the prover's randomness space. For a fixed statement x, we define an $(m+1)$-dimensional tensor $H_x \in \{0,1\}^{R \times S \times \ldots \times S}$ such that for any fixed randomness $r \in [R]$ and any challenge $s \in [S]^m$, the entry $H_x(r, s_1, \ldots, s_m)$ equals 1 if and only if the verifier accepts the prover's response. By giving TreeFind query access to H_x and input $(\alpha_1, \ldots, \alpha_m)$, we obtain a tree of 1-round transcripts, where the vertices at level $m+1$ are labeled by the prover's messages and the other vertices do not have a label. The edges are labeled by the different choices for $s_1, \ldots, s_m$ and each vertex at level i has precisely α_i children. Note that since s is sent at once, the order of the entries in s does not matter and we can assume that all the entries i that have $\alpha_i = 1$ are arranged in the lower levels of the tree.

4.5.2 Security Proof

We prove security based on a new assumption that extends the *generalized iterated squaring assumption* [HP25], which states that, given a uniform group element x, it takes time T to compute $(x^r)^{2^T}$ for any choice of $r \neq 0$. Our assumption (Definition 19), which we

call *strong* generalized *parallel* iterated squaring assumption, extends the generalized iterated squaring assumption to the parallel setting. It states that, given m random group elements $x_1, \ldots, x_m$ it is hard to perform the equivalent task in time $T + T/(m-1)$ on *less than* m processors. For technical reasons in our proof, we additionally give the adversary access to an oracle that on input two group elements (x, y) decides whether or not $x^{2^T} = y$. The access to such an oracle, for example, allows the prover to sample group elements and test whether they are solutions to some iterated squaring challenge in time much smaller than T. However, note that such test would succeed only with a negligible probability, and, in general, the deciding oracle does not seem to give the adversary any significant advantage in solving iterated squaring instances. Nevertheless, it is an interesting open problem to prove security of our protocol under the parallel generalized iterated squaring assumption (i.e., a weaker assumption without the deciding oracle) or even only under the generalized iterated squaring assumption (i.e., reducing to the hardness of a single iterated squaring instance).

Definition 13 (strong parallel generalized iterated squaring assumption)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *strong parallel generalized iterated squaring assumption* holds for $\texttt{GGen}$ if, for all $T, m \in \textbf{poly}(\lambda)$ and any probabilistic parallel algorithm $\mathcal{A}$ that uses at most $m-1$ processors and runs in time less than $T + T/(m-1)$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$, m random group elements $x_1, \ldots, x_m$ and an integer T. $\mathcal{A}$ additionally gets access to an oracle $\mathcal{O}$ that on input two group elements (x, y), decides whether or not $x^{2^T} = y$.
2. $\mathcal{A}$ outputs pairs $(y_1, r_1), \ldots, (y_m, r_m) \in \mathbb{G} \times \mathbb{Z}$.
3. $\mathcal{A}$ wins if and only if $r_1, \ldots, r_m \neq 0$ and $y_1 = (x_1^{r_1})^{2^T}, \ldots, y_m = (x_m^{r_m})^{2^T}$.

We prove security for the case where $\texttt{Batch}$ is the Bucket protocol, since this is the batching protocol with the best performance and because extracting $y_1, \ldots, y_m$ is the most complicated in this protocol. It is straightforward to use the techniques in the proof for proving security in the cases where $\texttt{Batch}$ is instantiated as the Random Exponents protocol or the Hybrid protocol.

To extract a fixed y_i, the extractor needs a tree of 8 transcripts that is structured as follows: The edges are labeled by the entries of the challenge vectors of the verifier and the leaves are labeled by the prover's output $\tilde{y}$ corresponding to the challenges on the path to this leaf. The tree is of depth $\rho \cdot (m + K + 1)$ and the vertices have either degree one or two. Recall that we can use algorithm TreeFind to find such a tree and assume that the vertices of degree one are in the lower levels and the vertices of degree two are in the higher levels of the tree (see Remark 3). In the following, let us consider the subtree in the last three levels where all vertices have degree two. We extract y_i as follows: The edges in the last level of the tree are labeled by the i-th entries of the challenges used in Step 4 of the bucket protocol. These challenges are four pairs of the following form:

$$r = (r_1, ..., r_{i-1}, r_i, r_{i+1}, ..., r_\rho) \text{ and } r' = (r_1, ..., r_{i-1}, r'_i, r_{i+1}, ..., r_\rho),$$

where all entries except for r_i and r'_i are fixed in the higher levels of the tree. Fix two neighboring leaves and their parent node. Denote the corresponding leaf labels by $\tilde{y}$ and $\tilde{y'}$. Then we have that $\tilde{y}/\tilde{y'} = g^{r_i - r'_i}$, where g denotes the corresponding element that an honest

prover computes in Step 3 of the bucket protocol. We store the element $\tilde{y}/\tilde{y}' = g^{r_i - r_i'}$ and repeat this procedure for the remaining three subtrees consisting of two neighboring leaves and their parent node. In the next step, we extract a power of the element computed in Step 2 for the bucket that x_i gets assigned to. We do this in a similar manner as before using the 4 elements of the form $\tilde{y}/\tilde{y}'$ and raising the elements to the correct exponents.

Finally, we can extract y_i by dividing the two elements we have obtained. This works because the challenges in the first round are the bucket assignments. Fix any assignment of buckets of the first challenge. Then the second challenge is the same except that x_i is assigned to a different bucket. This means that the extracted elements correspond to the product of (powers of) the same elements except that one of them is additionally multiplied by a power of $x_i^{2^T}$ and the other one is not. By dividing the former by the latter, we obtain a power of $x_i^{2^T}$. Note that we know exactly which power this is because it is a combination of the challenges the extractor sends. We repeat this procedure for every y_i and output the result together with the corresponding powers. Lemma 6 guarantees that TreeFind can sample a tree for a fixed y_i in expected time polynomial in the success probability of the adversary. Hence, the extractor obtains m trees in expected time polynomial in m and the success probability of the adversary.

We note that the algorithm TreeFind expects elements $\alpha_1, \ldots, \alpha_m$ as input that indicate the degree of the vertices of the tree of transcripts. Unfortunately, our extractor does not know which bucket element y_i will land in and thus has to guess which of the buckets K it needs two distinct challenges. Its success probability in one round is, therefore, reduced by a factor $1/K$. We believe that one can adapt TreeFind in such a way that it chooses some of the α_i's adaptively without any efficiency loss, which would remove the probability loss of $1/K$, but decided to use the algorithm from [ACK21] for simplicity of presentation.

Theorem 6. *Let* `PoE` *be a proof of exponentiation with soundness error* ϵ*. Consider an adversary* $\mathcal{A}$ *that uses at most* $m-1$ *processors and convinces the verifier to output accept in the protocol in Figure 4.7, where* `Batch` *is the Bucket protocol, with probability* $\delta > \epsilon$*. Then there exists an adversary* $\mathcal{B}$ *solving the strong parallel generalized iterated squaring game with* $m-1$ *processors in expected time*

$$T + \frac{8mt}{2(\delta - \epsilon)/K - 6/K^2}.$$

Finally, we discuss the implications of the above adversary on the security of our protocol presented in Figure 4.7. Theorem 6 implies that for any $\delta > 8mt(m-1)K/(2T) + 3/K + \epsilon$, the adversary $\mathcal{B}$ breaks the strong parallel generalized iterated squaring assumption. However, note that the lower bound on δ might not be negligible for some choice of parameters m, t, K and T, which means that there could exist an adversary breaking soundness of the protocol in Figure 4.7 with noticeable probability. In this case, the soundness error can be reduced to $(8mt(m-1)K/(2T) + 3/K + \epsilon)^\omega$ by repeating the protocol ω many times.

Proof of Theorem 6. We construct adversary $\mathcal{B}$ as follows:

1. $\mathcal{B}$ gets as input m generalized iterated squaring instances $x_1, \ldots, x_m$ and access to the oracle $\mathcal{O}$ described in Definition 19.

2. $\mathcal{B}$ sends $x_1, \ldots, x_m$ to $\mathcal{A}$ and starts the timer.

3. After time T, $\mathcal{B}$ starts the extraction procedure. Recall that the challenges sent by $\mathcal{V}$ in the Bucket Protocol contain two matrices $B \in [K]^{\rho\times m}$, $R \in [K]^{\rho\times K}$ and a vector $r \in [2^\ell]^\rho$. We can arrange the elements of the challenge in a long vector $c \in [K]^{\rho(m+K)} \times [2^\ell]^\rho$ by arranging the rows of the matrices consecutively. Denote by $d = \rho \cdot (m + K + 1)$ the dimension of the vector. To extract a power of y_i for a fixed $i \in [m]$, $\mathcal{B}$ invokes TreeFind with input $\alpha = (\alpha_1, \dots \alpha_d)$, where $\alpha_{d-\rho+1} = \alpha_i = 2$ and $\alpha_j = 2$, for some uniformly random $j \leftarrow [\rho \cdot m + 1, \rho \cdot m + K]$. All other entries in α are set to 1.

4. To answer the queries by TreeFind, $\mathcal{B}$ sends the corresponding challenge to $\mathcal{A}$. By assumption, $\mathcal{A}$ sends a response in time t with probability at least δ. If it does not send a response in time t, $\mathcal{B}$ answers the query with 0. If $\mathcal{A}$ outputs $\tilde{y}$ in time, $\mathcal{B}$ constructs the corresponding $\tilde{x}$ and queries $\mathcal{O}$ on $(\tilde{x}, \tilde{y})$. $\mathcal{B}$ forwards the response of $\mathcal{O}$ to TreeFind.

5. If TreeFind outputs a tree of transcripts, $\mathcal{B}$ proceeds to the next step. If TreeFind aborts, $\mathcal{B}$ goes back to Step 3.

6. At this step $\mathcal{B}$ has a tree of 8 transcripts corresponding to 8 challenges of the following form:

$$\{c^{(b_1,b_2,b_3)} = (c_1, \dots, c_i^{(b_1)}, \dots, c_{\rho\cdot m}, c_{\rho\cdot m+1}, \dots, c_j^{(b_1,b_2)}, \dots \\ \dots, c_{\rho\cdot m+\rho K}, c_{\rho\cdot m+\rho K+1}, \dots, c_{d-\rho+1}^{(b_1,b_2,b_3)}, \dots, c_d)\}_{b_1,b_2,b_3\in\{0,1\}}.$$

   First $\mathcal{B}$ checks if one of $c_i^{(0)}$ and $c_i^{(1)}$ is equal to j. If not, $\mathcal{B}$ goes back to Step 3. Otherwise, assume without loss of generality that $c_i^{(0)} = j$. Denote the corresponding messages output by $\mathcal{A}$ by $a_0^{(b_1,b_2,b_3)}$. To extract y_i, $\mathcal{B}$ does the following:

   a) For all $b_1, b_2 \in \{0, 1\}$, set $a_1^{(b_1,b_2)} := a_0^{(b_1,b_2,0)}/a_0^{(b_1,b_2,1)}$.

   b) For $b_1 \in \{0, 1\}$, set

$$a_2^{(b_1)} := \frac{(a_1^{(b_1,0)})^{c_{d-\rho+1}^{(b_1,1,0)} - c_{d-\rho+1}^{(b_1,1,1)}}}{(a_1^{(b_1,1)})^{c_{d-\rho+1}^{(b_1,0,0)} - c_{d-\rho+1}^{(b_1,0,1)}}}.$$

   c) Set

$$y_i := \frac{(a_2^{(0)})^{(c_{d-\rho+1}^{(1,1,0)} - c_{d-\rho+1}^{(1,1,1)})(c_{d-\rho+1}^{(1,1,0)} - c_{d-\rho+1}^{(1,1,1)})(c_j^{(1,0)} - c_j^{(1,1)})}}{(a_2^{(1)})^{(c_{d-\rho+1}^{(0,1,0)} - c_{d-\rho+1}^{(0,1,1)})(c_{d-\rho+1}^{(0,1,0)} - c_{d-\rho+1}^{(0,1,1)})(c_j^{(0,0)} - c_j^{(0,1)})}}$$

   and

$$r_i := \left(c_j^{(0,0)} - c_j^{(0,1)}\right)\left(c_j^{(1,0)} - c_j^{(1,1)}\right) \prod_{b_1,b_2\in\{0,1\}} \left(c_{d-\rho+1}^{(b_1,b_2,0)} - c_{d-\rho+1}^{(b_1,b_2,1)}\right).$$

7. $\mathcal{B}$ repeats the procedure starting from Step 3 for all $i \in [m]$.

8. $\mathcal{B}$ outputs $((y_1, r_1), \dots, (y_m, r_m))$.

We now show that the output of $\mathcal{B}$ is always correct. Fix any $i \in [m]$. By inspecting the bucket protocol we observe that the 8 $\tilde{x}$ values that $\mathcal{B}$ computes are of the form

$$\{((x_i^{b_1} h_1)^{c_j^{(b_1,b_2)}} h_2)^{c_{d-\rho+1}^{(b_1,b_2,b_3)}} h_3\}_{b_1,b_2,b_3\in\{0,1\}}$$

for some elements $h_1, h_2, h_3 \in \mathbb{G}$. Now if we perform the same operation on those values as on the values $a_0^{(b_1,b_2,b_3)}$ in Step 6 of the extraction, we obtain the element $x_i^{r_i}$. Since we know that $a^{(b_1,b_2,b_2)} = \tilde{x}^{2^T}$ for all $b_1, b_2, b_3 \in \{0,1\}$, we also have that $y_i = (x_i^{r_i})^{2^T}$.

It remains to compute the running time of $\mathcal{B}$. By Lemma 6 we know that TreeFind is successful in constructing a tree for $\mathcal{B}$ with probability at least $\delta - \epsilon - 3/K$ and the expected sample complexity of one run is at most 8. Since we choose $j \in [\rho \cdot m + 1, \rho \cdot m + K]$ uniformly at random, we have that the probability of a fixed $c_i^{(b_1)}$ being equal to j is $1/K$. Since TreeFind outputs two distinct values for $c_i^{(0)}$ and $c_i^{(1)}$, we have that the probability that one of the values is equal to j is $2/K$. Hence, we have that TreeFind outputs a tree that let's $\mathcal{B}$ extract y_i with probability at least $2/K \cdot (\delta - \epsilon - 3/K)$. Since $\mathcal{B}$ needs m trees of transcripts, we follow that the expected running time of $\mathcal{B}$ is at most $T + 8mt/(2(\delta-\epsilon)/K - 6/K^2)$. This means that $\mathcal{B}$ breaks the strong parallel iterated squaring assumption whenever $8mt/(2(\delta-\epsilon)/K - 6/K^2) < T/(m-1)$ and the claim follows. □

4.6 Conclusion

In this chapter, we introduced two new batch proofs of exponentiation that substantially improve over existing protocols in both theory and practice. First, we showed that combining the Random Subsets approach with the standard Random Exponents technique leads to a simple Hybrid Protocol that achieves a better balance of prover and verifier costs, while still producing a single proof for all aggregated statements. Second, we presented the Bucket Protocol, an adaptation of a classical bucket-test technique for signature verification, tailored to hidden-order groups. With an appropriate choice of parameters, the Bucket Protocol achieves an order-of-magnitude speedup over existing protocols for large batches of instances, significantly reducing both the proof size and verification overhead.

To demonstrate the practicality of these constructions, we provided an implementation and benchmark results showing that, even in a non-optimized setting, our protocols outperform earlier batch PoEs as the number of instances grows. Additionally, we proposed a new application for batch PoEs in remote attestation of parallel computational power. By combining parallel repeated-squaring challenges with our batch PoEs, a verifier can efficiently test whether a prover truly possesses a claimed number of parallel processors—while maintaining near-optimal communication and verification complexity.

Our work highlights both the importance of batching in real-world scenarios that require frequent generation of PoEs (e.g., verifiable delay functions in blockchain consensus) and the flexibility of bucket-based methods to optimize performance. We leave a deeper exploration of implementation optimizations, as well as investigating stronger security proofs under progressively weaker assumptions, as exciting directions for future research.

CHAPTER 5

Practical Short-Lived Proofs from Verifiable Delay Functions

5.1 Introduction

In this chapter we present the first constructions that transform *any* PoE in hidden order groups into a watermarkable VDF and into a zero-knowledge VDF. Our watermarkable VDF is practically efficient and only slightly increases the proof size of the PoE. However, the zero-knowledge VDF increases the proof size by roughly 4λ group elements, where λ is a statistical security parameter. While the proof size is still independent of the time parameter T, a blow-up by 4λ group elements is undesirable in practice.

To address this, we introduce the notion of zero-knowledge proofs of sequential work (zk-PoSW). We show that zkPoSW can replace zkVDFs in the construction of short-lived proofs and signatures with reusable forgeability from [ABC22], and also give a construction that transforms any PoE into a zkPoSW which only increases the proof size of the PoE by 3 group elements. As before, using Pietrzak's PoE we get short lived proofs and signatures with shorter forging times and different assumptions than [ABC22].

In this chapter we present all VDFs with exponent 2^T for simplicity. As before, the 2 could be replaced by any other constant q.

Watermarkable VDFs. We construct the first general watermarkable VDF scheme from any PoE in hidden order groups. In this construction the proof of the statement $y \stackrel{?}{=} x^{2^T}$ is computed as follows:

1. Sample a random $r \leftarrow \pm[2^\lambda]$.
2. Compute $x' := x^r$ and $y' := y^r$.
3. Compute a PoE proof π_{PoE} for the statement $y' \stackrel{?}{=} (x')^{2^T}$.
4. Compute a watermarked zero-knowledge proof of knowledge of r, denoted by π_{PoK}.
5. Publish $x, x', y, y', \pi_{\text{PoE}}$ and π_{PoK}.

Watermarking the proof of knowledge is done by including a unique identifier in the computation of the random challenge. Watermark unforgeability holds by soundness of the proof of knowledge, a decisional variant of the discrete log assumption in hidden order groups and a decisional variant of the iterated squaring assumption: If the proof of knowledge is sound, then the only two ways for an adversary to forge a proof with its own watermark are the following:

- Find r such that $x^r = x'$ and $y^r = y'$, copy π_{PoE} and honestly compute π_{PoK}. By finding r, the adversary finds a small discrete log of x' with base x.
- Compute a PoE for a new statement $y^{r'} \stackrel{?}{=} (x^{r'})^{2^T}$ faster than time T. If the adversary is able to do this, then in particular it can also recognize that y is indeed the result of x^{2^T} in time faster than T, which breaks the decisional iterated squaring assumption.

The construction in [ABC22] is only slightly more efficient than ours: It increases the proof size of Wesolowski's proof by one group element, whereas our construction increases the proof size of a PoE by four group elements. However, our construction can watermark *any* PoE, while the construction in [ABC22] cannot be generalized to other PoEs than Wesolowski's. Further, the watermarkable VDF in [ABC22] cannot reveal the output of the iterated squaring instance since it is based on a zero knowledge VDF, which may be undesirable for other applications.

Zero-Knowledge VDFs. We construct the first general zero-knowledge VDF from any PoE in hidden order groups. It is similar to the watermarkable VDF construction but instead of publishing the element y, the prover just proves knowledge of y. In this construction the proof of the statement $x^{2^T} = y$ is computed as follows:

1. Sample a random $r \leftarrow \pm[2^\lambda]$.
2. Compute $x' := x^r$ and $y' := y^r$.
3. Compute a PoE proof π_{PoE} for the statement $y' \stackrel{?}{=} (x')^{2^T}$.
4. Compute a zero-knowledge proof of knowledge of y and r, denoted by π_{PoK}.
5. Publish $x, x', y', \pi_{\text{PoE}}$ and π_{PoK}.

To prove that this scheme is zero-knowledge, the simulator needs a precomputed pair x^*, y^* and a PoE for the statement $y^* \stackrel{?}{=} (x^*)^{2^T}$, which we can include in the public parameters. Then it can output a simulated proof simply by forging the proof of knowledge of y and r. We need to rely on a decisional version of the discrete log assumption in hidden order groups so that the adversary cannot decide if there exists a small discrete log between elements x^* and x or not. The bottleneck of this construction is the zero-knowledge proof of knowledge of y and r. We obtain it by combing Schnorr's protocol with the Guillou-Quisquater protocol for proving knowledge of a s-root in hidden order groups. In our setting we can only prove soundness of this scheme when using challenge space $\{0, 1\}$ and running λ many repetitions, which makes the scheme impractical.

Zero-Knowledge Proofs of Sequential Work. We salvage the efficiency of the above scheme by dropping the requirement of a proof of knowledge of y. This means that our construction is not a VDF anymore: The prover might not know the unique result $y = x^{2^T}$ since it can also just compute $y' = (x^r)^{2^T}$ and output a valid proof. However, we assume that computing the proof still requires T steps, which is sufficient for a proof of sequential work. We call the assumption that computing $(x^r)^{2^T}$ for an adversarially chosen $r \neq 0$ takes T sequential steps *generalized iterated squaring assumption*. In this construction, given x, the proof of sequential work is computed as follows:

1. Compute $y = x^{2^T}$ together with advice string α.
2. Sample a random $r \leftarrow \pm[2^\lambda]$.
3. Compute $x' := x^r$ and $y' := y^r$.
4. Compute a PoE proof π_{PoE} for the statement $y' \stackrel{?}{=} (x')^{2^T}$ using α.
5. Compute a zero-knowledge proof of knowledge of r, denoted by π_{PoK}.
6. Publish $x, x', y, y', \pi_{\text{PoE}}$ and π_{PoK}.

New Assumptions. As discussed above, we use two assumptions in this chapter that are natural modifications of well-known assumptions but, to the best of our knowledge, have not already been defined in previous work.

- The *decisional discrete log assumption with small exponents* (defined in Section 5.3) states that given group elements a, b it's hard to decide if there exists a discrete logarithm $w, b = a^w$ even if the discrete logarithm w is guaranteed to be bounded by 2^λ for a security parameter λ (rather than uniform as in the standard discrete log assumption).

  We require this assumption to hold in the groups of unknown order over which the corresponding VDFs are defined. In groups of known order a stronger assumption is sometimes made in Diffie-Hellman key exchange where, for efficiency reasons, the exponents are chosen to be random numbers of only, say 275 bits (https://www.rfc-editor.org/rfc/rfc7919#section-5.2). In [Can97] Canetti makes an even stronger assumption (DHI Assumption II) in groups of order $2q+1$ for a large prime q, which implies that discrete log is hard even when the set from which the exponent is chosen is "well spread", which basically means it can be an arbitrary set of only slightly superpolynomial size.

- The *generalized iterated squaring assumption* (defined in Section 5.6.2) states that given x, computing a tuple (r, y) such that $y = (x^r)^{2^T}$ requires T steps. This generalizes the iterated squaring assumption where one requires $r = 1$. Rotem and Segev [RS20] analyze the delay property of generic ring functions. They show that, based on the hardness of factoring, any generic ring function is a delay function with time parameter determined by the sequentiality depth of the function. While the generalized iterated squaring assumption does not consider a *function* (the correct output is not unique), we believe that the techniques of Rotem and Segev can be applied in a straightforward manner to show that, in the generic ring model, breaking the generalized iterated squaring assumption is equivalent to factoring.

5.1.1 More Related Work

Time-Based Deniability. Baldimtsi et al. [BKZZ16] build so called *proofs of work or knowledge*, with which a prover can prove that they either know the witness of a statement or it has solved a proof of work puzzle. However, in their work both cases are indistinguishable from the beginning, whereas in short-lived proofs the cases are indistinguishable only after time T has passed. Specter, Park and Green [SPG21] build protocols that prove that either a prover knows the witness of a statement or it has seen a value released at time T. Ferrari, Géraud and Sirkin [FGN15] construct *fading* signatures that also lose validity after a certain amount of time based on the RSW time-lock puzzle. However, they need to rely on a trusted authority that knows a trapdoor and they need that the verifier is more powerful than the prover.

Colburn [Col18] constructs short-lived proofs and signatures from proofs of work in his master thesis. The proofs of work in his thesis consist of finding preimages of hash functions and are thus parallelizable.

Wesolowski [Wes20] was the first one to use VDFs as a building block for time-based deniability. He presents an identification protocol based on a trapdoor VDF that loses its validity after time T. Finally, Arun, Bonneau and Clark [ABC22] were the first ones to build general short-lived proofs and signatures from VDFs.

5.2 Three Zero-Knowledge Proofs of Knowledge

We begin by presenting three zero-knowledge proofs of knowledge. We need the first one to construct a zero-knowledge proof of sequential work in Section 5.6, the second one to construct a watermarkable VDF in Section 5.4 and the third one to construct a zero-knowledge VDF in Section 5.5. The proofs of knowledge are not *tight*: while the size of the witness of an honest prover is bounded by 2^λ, the extractor might extract a witness of size up to $2^{3\lambda+2}$. Jumping ahead, this will affect the strength of the low order assumption needed for our VDF constructions: We will need to assume that it is hard to find elements of order up to $23\lambda + 2$.

5.2.1 Proof of Knowledge of Discrete Log

In Figure 5.1 we present Schnorr's protocol [Sch91] in hidden order groups. We use it as a building block to construct the zero knowledge PoSW in Section 5.6. Schnorr originally defined and analyzed the protocol in prime order groups. Later, Kiayias, Tsiounis and Yung [KTY04] proved that it is also secure in hidden order groups, where knowledge soundness is based on the strong RSA assumption. We note that the soundness property only guarantees knowledge of an exponent in $\pm[2^{3\lambda+2}]$ instead of $\pm[2^\lambda]$. This is sufficient for our application. The authors of [CPP17] claim that knowledge soundness of the protocol can be based on the RSA assumption instead of the Strong-RSA assumption but we are not aware of a formal proof.

Theorem 7 ([KTY04])**.** *Under the strong RSA assumption, the protocol in Figure 5.1 is an honest verifier zero-knowledge proof of knowledge with soundness error* $1/2^\lambda$*. The soundness property guarantees knowledge of an exponent in* $\pm[2^{3\lambda+2}]$*.*

Theorem 7 is a special case of [KTY04, Theorem 10]. For completeness we restate the proof for this case.

Instance: $(a_1, a_2, \mathbb{G})$, where $a_1, a_2 \in \mathbb{G}$

Parameters: statistical security parameter λ

Witness: Exponent $w \in \pm[2^\lambda]$ such that $a_1^w = a_2$ in $\mathbb{G}$

Protocol:

1. $\mathcal{P}$ samples $t \leftarrow \pm[2^{3\lambda}]$ uniformly at random, computes $b := a_1^t$ and sends it to $\mathcal{V}$.
2. $\mathcal{V}$ samples $c \leftarrow [2^\lambda]$ uniformly at random and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $s := t + cw$ and sends it to $\mathcal{V}$.
4. $\mathcal{V}$ checks if $s \in \pm[2^{3\lambda+1}]$ and $a_1^s = b a_2^c$ and outputs `accept` or `reject` accordingly.

Figure 5.1: Proof of Knowledge of Discrete Log [Sch91, KTY04].

Proof of Theorem 7. Completeness follows by inspection of the protocol. To prove knowledge soundness we construct an extractor $\mathcal{E}$ that outputs a witness w given two accepting transcripts (a_1, a_2, b_1, c, s) and $(a_1, a_2, b_1, c^*, s^*)$. Since both transcripts are accepting, it holds that $a_1^{s-s^*} = a_2^{c-c^*}$. Let $\gamma = \gcd(s-s^*, c-c^*)$ and α, β be such that $\gamma = \alpha(s-s^*)+\beta(c-c^*)$. With high probability it holds that γ is coprime to the order of $\mathbb{G}$ since otherwise we could factor the group order and in particular break the strong RSA assumption. We thus have

$$a_1^{\frac{s-s^*}{\gamma}} = a_2^{\frac{c-c^*}{\gamma}}$$

and hence

$$a_1 = a_1^{\alpha\frac{s-s^*}{\gamma}+\beta\frac{c-c^*}{\gamma}} = (a_2^\alpha a_1^\beta)^{\frac{c-c^*}{\gamma}}.$$

Now if $c - c^* > \gamma$, we can transform the prover into an algorithm that breaks the strong RSA assumption: $a_2^\alpha a_1^\beta$ is the $((c-c^*)/\gamma)$-root of a_1. We therefore have that $c - c^* = \gamma$ and hence $w = (s-s^*)/(c-c^*)$ is the discrete log of a_2 with base a_1. Since $s, s^* \in \pm[2^{3\lambda+1}]$ we have that $w \in \pm[2^{3\lambda+2}]$.

It remains to prove that the protocol is honest verifier zero knowledge. Consider the simulator $\mathcal{S}$ that takes as input a tuple (a_1, a_2, c^*), samples $s^* \leftarrow \pm[2^{3\lambda}]$ uniformly at random and computes $b^* = a_1^s a_2^{-c}$. To prove that this transcript is indistinguishable from a real transcript, we show that the statistical distance of the random variable $s^* \leftarrow \pm[2^{3\lambda}]$ to the random variable $s = t + cw$ for a fixed $w \in \pm[2^{2\lambda}]$ and uniformly random $t \leftarrow \pm[2^{3\lambda}]$ and $c \leftarrow [2^\lambda]$ is negligible. Since s^* is distributed uniformly over $\pm[2^{3\lambda}]$, it takes each value in this set with probability $1/2^{3\lambda+1}$. Now consider the distribution of s. Any value in the range $[-2^{3\lambda} + 2^{2\lambda}, 2^{3\lambda} - 2^{2\lambda}]$ is selected with probability $2^\lambda/(2^\lambda 2^{3\lambda+1}) = 1/2^{3\lambda+1}$ since for any choice of c we can find a t that yields the respective value. On the rest of the values the distributions might differ. If follows that the statistical distance of the two distributions is at most

$$1 - \frac{2^{3\lambda+1} - 2^{2\lambda+1}}{2^{3\lambda+1}} = \frac{1}{2^\lambda}.$$

□

Instance: $(a_1, a_2, a_3, a_4, \mathbb{G})$, where $a_1, a_2, a_3, a_4 \in \mathbb{G}$

Parameters: statistical security parameter λ

Witness: Exponent $w \in \pm[2^\lambda]$ such that $a_1^w = a_2$ and $a_3^w = a_4$ in $\mathbb{G}$

Protocol:

1. $\mathcal{P}$ samples $t \leftarrow [2^{3\lambda}]$ uniformly at random, computes $b_1 := a_1^t$ and $b_2 := a_3^t$ and sends (b_1, b_2) to $\mathcal{V}$.
2. $\mathcal{V}$ samples $c \leftarrow [2^\lambda]$ uniformly at random and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $s := t + cw$ and sends it to $\mathcal{V}$.
4. $\mathcal{V}$ checks if $s \in \pm[2^{3\lambda+1}]$, $a_1^s = b_1 a_2^c$ and $a_3^s = b_2 a_4^c$ holds and outputs `accept` or `reject` accordingly.

Figure 5.2: Proof of Knowledge of same discrete log [KTY04]

5.2.2 Proof of Knowledge of Same Discrete Log

The main tool in our construction of a watermarkable signature scheme is a proof of knowledge of the same discrete log for two different bases. The protocol is a special case of the general proof of knowledge for "discrete-log relations sets" introduced by Kiayias, Tsiounis and Yung in [KTY04]. It was first constructed in prime order groups by Chaum and Pederson [CP93]. We present it in Figure 5.2.

Theorem 8 ([KTY04, Theorem 10])**.** *Under the Strong-RSA assumption, the protocol in Figure 5.2 is an honest verifier zero-knowledge proof of knowledge with soundness error* $1/2^\lambda$*. The soundness property guarantees knowledge of an exponent in* $\pm[2^{3\lambda+2}]$*.*

Theorem 8 is a special case of [KTY04, Theorem 10]. The proof is very similar to the proof of Theorem 7 so we omit it. To make the protocol non-interactive, we apply the Fiat-Shamir heuristic, i.e., we replace the challenge sent by the verifier by a hash of the instance, the first message and an identifier `ID` of the prover. The protocol can be found in Figure 5.3. In our application to watermarkable VDFs, we need this protocol to be watermarked. We achieve this by including an ID of the prover in the input of the hash function that computes the Fiat-Shamir challenge.

5.2.3 Proof of Knowledge of Same Discrete Log with one Hidden Base

In our zero-knowledge VDF construction we need a proof of knowledge that's similar to the one in the last subsection but without revealing element a_3. The protocol is given in Figure 5.4. It is a combination of the protocol in Figure 5.2 and the well-known Guillou-Quisquater protocol [GQ90] for proving knowledge of a root.

Theorem 9. *Under the Strong-RSA assumption, the protocol in Figure 5.4 is an honest verifier zero-knowledge proof of knowledge with soundness error* $1/2^\lambda$*. The soundness property guarantees knowledge of an exponent in* $\pm[2^{3\lambda+2}]$*.*

Instance: $(a_1, a_2, a_3, a_4, \texttt{ID}, \mathbb{G})$, where $a_1, a_2, a_3, a_4 \in \mathbb{G}$ and $\texttt{ID}$ is a unique identifier of $\mathcal{P}$

Parameters: statistical security parameter λ, hash function H

Witness: Exponent $w \in [2^\lambda]$ such that $a_1^w = a_2$ and $a_3^w = a_4$ in $\mathbb{G}$

Protocol:

1. $\mathcal{P}$ samples $t \leftarrow [2^{3\lambda+1}]$ uniformly at random and computes $b_1 := a_1^t$ and $b_2 := a_3^t$.
2. $\mathcal{P}$ computes $c := H(a_1, a_2, a_3, a_4, b_1, b_2, \texttt{ID})$
3. $\mathcal{P}$ computes $s := t + cw$ and publishes (b_1, b_2, c, s) as the proof.
4. To check the proof (b_1, b_2, c, s), $\mathcal{V}$ checks if $H(a_1, a_2, a_3, a_4, b_1, b_2, \texttt{ID}) = c$, $s \in \pm[2^{3\lambda+1}]$ and if both $a_1^s = b_1 a_2^c$ and $a_3^s = b_2 a_4^c$ hold and outputs `accept` or `reject` accordingly.

Figure 5.3: `PoKsDL`: The watermarked non-interactive Proof of Knowledge of same discrete log

Instance: $(a_1, a_2, a_4, \mathbb{G})$, where $a_1, a_2, a_4 \in \mathbb{G}$

Parameters: statistical security parameter λ

Witness: Element a_3 and exponent $w \in \pm[2^\lambda]$ such that $a_1^w = a_2$ and $a_3^w = a_4$ in $\mathbb{G}$

Protocol: $\mathcal{P}$ and $\mathcal{V}$ repeat the following procedure λ times:

1. $\mathcal{P}$ samples $a_5 \leftarrow \mathbb{G}$ uniformly at random, computes $b_1 := a_5^w$ and sends b_1 to $\mathcal{V}$.
2. $\mathcal{V}$ samples a bit $b \leftarrow \{0, 1\}$ uniformly and random and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ samples $t \leftarrow \pm[2^{3\lambda}]$ uniformly at random, computes $a_6 := a_5 a_3^b$, $b_2 := a_1^t$ and $b_3 := a_6^t$ and sends (a_6, b_2, b_3) to $\mathcal{V}$.
4. $\mathcal{V}$ samples $c \leftarrow [2^\lambda]$ uniformly at random and sends it to $\mathcal{P}$.
5. $\mathcal{P}$ computes $s := t + cw$ and sends it to $\mathcal{V}$.
6. $\mathcal{V}$ checks if $s \in \pm[2^{3\lambda+1}]$, $a_1^s = b_2 a_2^c$ and $a_6^s = b_3 (b_1 a_4^b)^c$ hold and outputs `accept` or `reject` accordingly.

Figure 5.4: Proof of Knowledge of same discrete log with one hidden base

Proof. Completeness follows by inspection of the protocol. To prove knowledge soundness we consider one of the λ many executions. We construct an extractor $\mathcal{E}$ that outputs a witness (w, a_3) given four accepting transcripts $(b_1, 0, a_6, b_2, b_3, c, s)$, $(b_1, 0, a_6, b_2, b_3, c^*, s^*)$, $(b_1, 1, a_6^*, b_2^*, b_3^*, c^{**}, s^{**})$ and $(b_1, 1, a_6^*, b_2^*, b_3^*, c^{***}, s^{***})$. $\mathcal{E}$ first extracts w and then a_3.

1. Since the first and the second transcripts are accepting, it holds that $a_1^{s-s^*} = a_2^{c-c^*}$. Let $\gamma = \gcd(s - s^*, c - c^*)$ and α, β be such that $\gamma = \alpha(s - s^*) + \beta(c - c^*)$. Then we

have

$$a_1^{\frac{s-s^*}{\gamma}} = a_2^{\frac{c-c^*}{\gamma}}$$

and hence

$$a_1 = a_1^{\alpha\frac{s-s^*}{\gamma}+\beta\frac{c-c^*}{\gamma}} = (a_2^\alpha a_1^\beta)^{\frac{c-c^*}{\gamma}}.$$

Now if $c-c^* > \gamma$, we can transform the prover into an algorithm that breaks the strong RSA assumption: $a_2^\alpha a_1^\beta$ is the $(c-c^*/\gamma)$-root of a_1. We therefore have that $c-c^* = \gamma$ and hence $w = (s-s^*/c-c^*)$ is the discrete log of a_2 with base a_1 and the discrete log of b_1 with base a_6. By the same argument we get that $w = (s^{**}-s^{***}/c^{**}-c^{***})$ is the discrete log of $b_1 a_4$ with base a_6^*.

2. Now consider the second and third transcript. We have seen above that $a_6^w = b_1$ and $(a_6^*)^w = b_1 a_4$. This means that a_6^*/a_6 is a w-root of a_4.

It remains to prove honest verifier zero-knowledge. Given (a_1, a_2, a_4, b, c), the simulator $\mathcal{S}$ constructs an accepting transcript $(b_1, b, a_6, b_2, b_3, c, s)$ as follows: It first samples $s \leftarrow \pm[2^{3\lambda}]$ and $a_6 \leftarrow \mathbb{G}$ uniformly at random. If $b = 0$, $\mathcal{S}$ samples $e \leftarrow \pm[2^\lambda]$ uniformly at random and sets $b_1 = a_6^e$. If $b = 1$, it samples $b_1 \leftarrow \mathbb{G}$ uniformly at random. Finally, $\mathcal{S}$ computes $b_2 = a_1^s a_2^{-c}$ and $b_3 = a_6^s b_1^{-c} a_4^{-bc}$. Indistinguishability follows since the distribution of the simulated s has statistical distance $1/2^\lambda$ from the distribution of an honestly computed s as we have seen in the proof of Theorem 7. □

5.3 Modified Discrete-Log Assumptions

In our constructions we need to rely on the assumption that it is hard to recognize whether there exists a small discrete log between two given elements or not. Note that this is easy in one case: Given two elements $a, b \in \mathbb{G}$, where a is a square and b is a non-square, there exists no discrete log of b to base a since a raised to any power yields a square. We assume that it is hard in all other cases.

Definition 14 (discrete log assumption with small exponents)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *discrete log assumption with small exponents* holds for $\texttt{GGen}$ if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$, and two elements $a, b \in \mathbb{G}$, where a is uniformly random and $b = a^w$ for some $w \leftarrow \pm[2^{\lambda-1}]$.
2. $\mathcal{A}$ outputs an integer $w' \in \pm[2^{\lambda-1}]$.
3. $\mathcal{A}$ wins if and only if $b = a^{w'}$.

Definition 15 (decisional discrete log assumption with small exponents)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *decisional discrete log assumption with small exponents* holds for $\texttt{GGen}$ if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$, and two elements $a, b \in \mathbb{G}$, where a is uniformly random and for b there are two possibilities of probability $1/2$ each: Either $b = a^w$ for some $w \leftarrow \pm[2^{\lambda-1}]$ or $b = z^{2^b}$ for a uniformly random group element $z \in \mathbb{G}$, where $b = 1$ if a is a square and $b = 0$ if not.
2. $\mathcal{A}$ outputs 0 or 1 indicating whether or not $b = a^w$ for some $w \in \pm[2^{\lambda-1}]$.
3. $\mathcal{A}$ wins if and only if it outputs the correct bit with probability greater than $1/2$.

Remark 4 (the special case of QR_N^+)**.** Note that the decisional discrete log assumption with small exponents holds information theoretically in the group of signed quadratic residues QR_N^+, where N is a safe prime modulus, whenever the group order of QR_N^+ is at least 2^λ. This is because in this group almost all elements are generators and all elements are squares. Hence, if you pick two random group elements, the discrete log of one element to the base the other element exists with high probability so the two cases in the assumption are statistically indistinguishable.

Further, in this case, we have a straightforward reduction from the strong RSA assumption to the discrete log assumption with small exponents: Given a random group element g, one can solve the strong RSA challenge by sampling a random group element h and sending (h, g) to the adversary $\mathcal{A}$ that breaks the discrete log assumption with small exponents. When $\mathcal{A}$ outputs w, the reduction sends (w, h) to the strong RSA challenger.

5.4 Watermarkable VDFs

In this section we show how to transform any PoE into a watermarkable VDF. We begin by recalling the definition of watermarkable VDFs.

5.4.1 Definition

Watermarkable verifiable delay functions were informally introduced by Wesolowski [Wes20]. The first formal definition was given by Arun, Bonneau and Clark in [ABC22].

Definition 16. A watermarkable VDF is a set of algorithms ($\texttt{Setup}$, $\texttt{Eval}$, $\texttt{WatermarkProve}$, $\texttt{Verify}$), where

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ on input statistical security parameter 1^λ and time parameter T outputs public parameters $\mathbf{pp}$.

$\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ on input $(\mathbf{pp}, x, T)$ outputs (y, α), where α is an advice string.

$\texttt{WatermarkProve}(\mathbf{pp}, x, \mu, y, \alpha) \to (y, \pi_\mu)$ outputs a proof for y with embedded watermark μ.

$\texttt{Verify}(\mathbf{pp}, x, \tilde{\mu}, y, \pi_\mu) \to \texttt{accept/reject}$ checks that $y = \texttt{Eval}(\mathbf{pp}, x)$ and that the watermark $\tilde{\mu}$ is embedded in π_μ.

The algorithm $\texttt{Eval}$ is deterministic and can compute the output y in T sequential steps. A watermarkable VDF must additionally satisfy four properties: The security properties of a basic VDF and watermark unforgeability. We state them informally below. The formal definitions can be found in [BBBF18] and [ABC22].

Completeness: For all tuples $(\mathbf{pp}, x, \mu, y, \pi_\mu)$, where $y = \texttt{Eval}(\mathbf{pp}, x)$ and $\pi_\mu = \texttt{WatermarkProve}(\mathbf{pp}, x, \mu, y, \alpha)$, algorithm $\texttt{Verify}(\mathbf{pp}, x, \mu, y, \pi_\mu)$ outputs `accept`.

Sequentiality: Any parallel algorithm that uses at most poly(λ) processors and outputs $y = \texttt{Eval}(\mathbf{pp}, x)$ with noticeable probability runs in time at least T.

Soundness: If $\texttt{Verify}(\mathbf{pp}, x, \tilde{\mu}, y, \pi_\mu)$ outputs `accept`, then the probability that $y \neq \texttt{Eval}(\mathbf{pp}, x)$ is negligible.

Watermark Unforgeability: For any pair of algorithms $(\mathcal{A}_0, \mathcal{A}_1)$, where $\mathcal{A}_0$ runs in time $O(\text{poly}(T, \lambda))$ and $\mathcal{A}_1$ runs in time less than T, the probability that $(\mathcal{A}_0, \mathcal{A}_1)$ wins the following game is negligible:

1. The challenger C runs $\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ and sends $\mathbf{pp}$ to $(\mathcal{A}_0, \mathcal{A}_1)$.
2. Precomputation algorithm $\mathcal{A}_0(\mathbf{pp})$ outputs advice string $\tilde{\alpha}$.
3. Challenger C samples a random input x, runs $\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ and sends $(x, y, \tilde{\alpha})$ to $\mathcal{A}_1$.
4. Online algorithm $\mathcal{A}_1$ sends q many watermark queries μ_i to C and obtains $\texttt{WatermarkProve}(\mathbf{pp}, x, \mu_i, y, \alpha) \to \pi_{\mu_i}$.
5. Algorithm $\mathcal{A}_1$ outputs a forgery pair (μ_*, π_{μ_*}) and wins if $\mu_* \neq \mu_i$ for all $i \in [q]$ and $\texttt{Verify}(\mathbf{pp}, x, \tilde{\mu}, y, \pi_\mu)$ outputs `accept`.

5.4.2 Construction

In Figure 5.5 we present our watermarkable VDF. The main idea is to randomize the instance (x, y) to $(x', y') := (x^r, y^r)$ with a secret exponent r and then provide a PoE for the statement $y' \stackrel{?}{=} (x')^{2^T}$ and a watermarked proof of knowledge for r using the protocol in Figure 5.2. We present the protocol as non-interactive since only non-interactive proofs need to be watermarked.

Theorem 10. *Let* `PoE` *be a complete and sound proof of exponentiation. The algorithms in Figure 5.5 define a sound and complete VDF, relative to the iterated squaring assumption, the strong RSA assumption and the low order assumption.*

Proof. Sequentiality of the VDF follows immediately from the iterated squaring assumption. Completeness follows by inspection of the protocol from the completeness property of `PoE`. Soundness follows from the low order assumption, the strong RSA assumption and soundness of `PoE`. To see this, we show how to transform an adversary $\mathcal{A}$ that outputs an accepting proof

$$\pi_\mu = (x', y', \texttt{PoKsDL}(\mathbf{pp}, x, x', y, y', \texttt{ID}), \texttt{PoE}(\mathbf{pp}, x', y', T))$$

with $x^{2^T} \neq y$ with probability δ into an adversary $\mathcal{B}$ that breaks either the low order assumption, the strong RSA assumption or soundness of `PoE` with probability δ. The adversary $\mathcal{B}$ does the following:

1. Try to extract the secret exponent r from `PoKsDL`. If this is not possible, use $\mathcal{A}$ to break the strong RSA assumption similar to the proof of Theorem 8.

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp} = (\mathbb{G}, H)$ outputs a finite abelian group $\mathbb{G}$ of unknown order and an efficiently computable hash function H.

$\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ on input $(\mathbf{pp}, x, T)$ outputs (y, α), where $y = x^{2^T}$ and α is an advice string for $\texttt{PoE}$.

$\texttt{WatermarkProve}(\mathbf{pp}, x, \texttt{ID}, y, \alpha) \to (y, \pi_\mu)$ outputs y and

$$\pi_\mu = (x', y', \texttt{PoKsDL}(\mathbf{pp}, x, x', y, y', \texttt{ID}), \texttt{PoE}(\mathbf{pp}, x', y', T)),$$

where $x' := x^r$ and $y' := y^r$ for some uniformly random $r \leftarrow \pm[2^\lambda]$.

$\texttt{Verify}(\mathbf{pp}, x, \tilde{\mu}, y, \pi_\mu) \to \texttt{accept/reject}$ checks if both $\texttt{PoKsDL}(\mathbf{pp}, x, x', y, y', \texttt{ID})$ and $\texttt{PoE}(\mathbf{pp}, x', y', T))$ verify.

Figure 5.5: A Watermarkable VDF from any proof of exponentiation using the proof of knowledge $\texttt{PoKsDL}$ presented in Figure 5.3. By $\texttt{PoE}(\mathbf{pp}, x, y, T)$ we denote the chosen proof of exponentiation with group parameters $\mathbf{pp}$ and statement $x^{2^T} = y$.

2. If r is extractable, compute $\tilde{y} := x^{2^T}$ and $\alpha := \tilde{y}y^{-r}$. If $\alpha \neq 1$, check if $\alpha^r = 1$. If so, then α is an element of low order and r is a multiple of its order. $\mathcal{B}$ outputs (α, r) and breaks the low order assumption.

3. If $\alpha = 1$ or $\alpha^r \neq 1$, then $(x')^{2^T} \neq y'$, so $\texttt{PoE}(\mathbf{pp}, x', y', T)$ is a proof for a false statement, which is a contradiction to the assumption that $\texttt{PoE}$ is sound.

Whenever the proof output by $\mathcal{A}$ is accepting but $x^{2^T} \neq y$, algorithm $\mathcal{B}$ terminates in one of the steps, which concludes the proof. □

Remark 5 (On the running time of algorithm $\mathcal{B}$)**.** Note that in the above proof the running time of algorithm $\mathcal{B}$ might be linear in the time parameter T because it needs to solve an iterated squaring instance in the second step. This means that, to break the low order assumption or the soundness of the PoE, it needs at least T steps. Giving an adversary time linear in T to break the soundness of the PoE is necessary for a meaningful soundness definition since an honest prover also needs T steps to compute the result of an instance and the corresponding proof. Giving an adversary against the low order assumption time linear in T to break it, is in line with its usage in the literature (see [Wes20, Pie19], where it is needed for soundness of PoEs). If $\mathcal{B}$ breaks the strong RSA assumption it is much faster since it never gets to step 2. In particular, we have that its running time is independent of T in this case.

Theorem 11. *Let $\texttt{PoE}$ be a complete and sound proof of exponentiation. The VDF defined by the algorithms in Figure 5.5 is watermark unforgeable in the random oracle model, relative to the strong RSA assumption, the decisional discrete log assumption with small exponents, the low order assumption and the decisional iterated squaring assumption.*

Proof. We show how to transform an adversary $\mathcal{A}$ that wins the watermark unforgeability game with probability δ into an adversary $\mathcal{B}$ that breaks either the soundness of $\texttt{PoKsDL}$ (and hence the strong RSA assumption), the discrete log assumption or the decisional iterated squaring assumption with probability $\delta/2$.

1. Let q be the number of queries that $\mathcal{A}$ is allowed to make. Upon receiving as input a group $\mathbb{G}$ and a time parameter T, $\mathcal{B}$ precomputes $q' \geq q$ tuples

$$\{(x_i, y_i, \texttt{PoE}(\mathbf{pp}, x_i, y_i, T))\}_{i \in [q']},$$

where for all $i \in [q']$, $x_i \leftarrow \mathbb{G}$ is a uniformly random group element, $y_i = x_i^{2^T}$ and $\texttt{PoE}(\mathbf{pp}, x_i, y_i, T)$ is an honestly computed proof of exponentiation. Call L the list of those tuples. $\mathcal{B}$ computes those tuples until L contains q entries in which x_i is a square.

2. $\mathcal{B}$ gets as input a discrete log challenge (g, g^a), where g is a random group element in $\mathbb{G}$ and a is a random number in $\pm[2^{\lambda-1}]$. Note that by the decisional discrete log assumption with small exponents, $\mathcal{B}$ should not be able to find a.

3. $\mathcal{B}$ sends $\mathbb{G}$ to $\mathcal{A}_0$ and obtains advice string $\tilde{\alpha}$.

4. $\mathcal{B}$ gets as input a decisional iterated squaring challenge consisting of two group elements x_d, y_d that are either uniformly random elements in $\mathbb{G}$ or x_d is uniformly random in $\mathbb{G}$ and $y_d = x_d^{2^T}$

5. To simulate the watermark unforgeability game for the statement $y \stackrel{?}{=} x^{2^T}$, it chooses one of the following two strategies at random, each with probability $1/2$. Note that $\mathcal{B}$ can always forge a proof of knowledge of same discrete log since $\texttt{PoKsDL}$ is honest verifier zero-knowledge and the random oracle is programmable.

   Strategy 1: Compute $y := g^{2^T}$ and $y' := (g^a)^{2^T}$ and send $(\mathbb{G}, H, g, y, \tilde{\alpha})$ to $\mathcal{A}_1$. When $\mathcal{A}_1$ makes a watermark query $\texttt{ID}_i$, sample a random $r \leftarrow \pm[2^{\lambda-1}]$, forge $\texttt{PoKsDL}(\mathbf{pp}, g, g^{ar}, y, (y')^r, \texttt{ID}_i)$ and compute $\texttt{PoE}(\mathbf{pp}, g^{ar}, (y')^r, T)$. Send

$$\pi_i := (g^{a+r}, (y')^r, \texttt{PoKsDL}(\mathbf{pp}, g, g^{ar}, y, (y')^r, \texttt{ID}_i), \texttt{PoE}(\mathbf{pp}, g^{ar}, (y')^r, T))$$

   to $\mathcal{A}_1$. If $\mathcal{A}_1$ wins the game, it outputs

$$(\texttt{ID}_*, \pi_* = (x_*, y_*, \texttt{PoKsDL}(\mathbf{pp}, g, x_*, y, y_*, \texttt{ID}_*)), \texttt{PoE}(\mathbf{pp}, x_*, y_*, T)).$$

   If $(x_*, y_*) \neq (x_i, y_i)$ for all $i \in [q]$, abort. Else, let $\ell \in [2^\lambda]$ be such that $(x_*, y_*) = (g^{a\ell}, (y')^\ell)$. $\mathcal{B}$ tries to extract an exponent ω from $\texttt{PoKsDL}$. If it is successful, it computes $a' := \omega/\ell$ over $\mathbb{Z}$ and checks if $g^{a'} = g^a$. If so, it can output a' and break the discrete log assumption. If it does not hold then $a' \neq a$ but $g^{a'\ell} = g^{a\ell}$ and hence $g^{a'}/g^a$ is an element of low order ℓ. If it is not able to extract, it can use $\mathcal{A}$ to break the strong RSA assumption similiar to the proof of Theorem 8.

   Strategy 2: Send $(\mathbf{pp}, x_d, y_d, \tilde{\alpha})$ to $\mathcal{A}_1$. If x_d is a square, then remove all tuples $(x_i, y_i, \texttt{PoE}(\mathbf{pp}, x_i, y_i, T))$, where x_i is not a square, from the list L. When $\mathcal{A}_1$ makes a watermark query $\texttt{ID}_i$, pick an unused tuple $(x_i, y_i, \texttt{PoE}(\mathbf{pp}, x_i, y_i, T))$ from L, forge $\texttt{PoKsDL}(\mathbf{pp}, x_d, x_i, y_d, y_i, \texttt{ID}_i)$ and send

$$\pi_i := (x_i, y_i, \texttt{PoKsDL}(\mathbf{pp}, x_d, x_i, y_d, y_i, \texttt{ID}_i), \texttt{PoE}(\mathbf{pp}, x_i, y_i, T))$$

   to $\mathcal{A}_1$. By the decisional discrete log assumption with small exponents and the zero-knowledge property of $\texttt{PoKsDL}$, π_i is indistinguishable from an honestly computed watermarked proof. If $\mathcal{A}_1$ outputs

$$(\texttt{ID}_*, \pi_* = (x_*, y_*, \texttt{PoKsDL}(\mathbf{pp}, x_d, x_*, y_d, y_*, \texttt{ID}_*), \texttt{PoE}(\mathbf{pp}, x_*, y_*, T)),$$

check if $(x_*, y_*) = (x_i, y_i)$ for some $i \in [q]$ and abort if it holds. Otherwise, try to extract the secret r from $\texttt{PoKsDL}$. If this is not possible, use $\mathcal{A}$ to break the strong RSA assumption as above. If it is possible, we have that the statement $y_d \stackrel{?}{=} x_d^{2^T}$ holds since the PoE is sound. In this case $\mathcal{B}$ sends 1 to the decisional iterated squaring challenger. If $\mathcal{A}_1$ does not output a tuple of the form above, $\mathcal{B}$ sends 0 or 1 to the decisional iterated squaring challenger each with probability $1/2$.

If adversary $\mathcal{B}$ does not abort in Strategy 1, it breaks either the strong RSA assumption or the decisional discrete log assumption with small exponents with probability δ. If $\mathcal{B}$ does not abort in Strategy 2, it either breaks the strong RSA assumption with probability δ or it recognizes a true instance in the decisional iterated squaring game with probability $1/2 + \delta/2$. Since aborting in Strategy 1 and aborting in Strategy 2 are mutually exclusive, the claim follows. □

Remark 6 (On the running time of algorithm $\mathcal{B}$)**.** Note that in the first strategy $\mathcal{B}$ runs in time linear in T to break the strong RSA assumption or the discrete log assumption. We therefore need to assume that these assumptions are secure against adversaries that run in time linear in T, which is at most 2^{32} in practice.

The next corollary follows from the discussion in Remark 4.

Corollary 2. *Let $\texttt{PoE}$ be a complete and sound proof of exponentiation and let $\mathbb{G} = \mathrm{QR}_N^+$, where N is a safe prime modulus. The construction in Figure 5.5 is a watermarkable VDF in $\mathbb{G}$ relative to the decisional iterated squaring assumption and the strong RSA assumption.*

Efficiency Watermarking a PoE with the construction in Figure 5.5 increases the complexity of the underlying PoE scheme as follows:

- The proof size grows by 4 group elements and one integer of size at most $2^{3\lambda+1}$.
- The verifier needs to perform 4 additional small group exponentiations (with exponents of size at most $2^{3\lambda+1}$) and 2 group multiplications.
- The prover needs to perform 4 additional small exponentiations (with exponents of size at most $2^{3l\lambda}$).

5.5 Zero-Knowledge VDFs

5.5.1 Definition

Zero-knowledge verifiable delay functions were introduced by Arun, Bonneau and Clark in [ABC22].

Definition 17. A zero-knowledge VDF is a set of algorithms ($\texttt{Setup}$, $\texttt{Eval}$, $\texttt{Prove}$ $\texttt{Verify}$, $\texttt{Sim}$), where

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ on input statistical security parameter 1^λ and time parameter T outputs public parameters $\mathbf{pp}$.

$\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ on input $(\mathbf{pp}, x, T)$ outputs (y, α), where α is an advice string.

$\texttt{Prove}(\mathbf{pp}, x, y, \alpha) \to \pi$ outputs a proof π of knowledge of element y.

$\texttt{Verify}(\mathbf{pp}, x, \pi) \to \texttt{accept/reject}$ checks that π is a valid proof of knowledge.

$\texttt{Sim}(\mathbf{pp}, x, c^*) \to \pi^*$ outputs a simulated proof of knowledge π^* using randomness c^*.

The algorithm $\texttt{Eval}$ is deterministic and can compute the output y in T sequential steps. A zero-knowledge VDF must additionally satisfy four properties: Completeness, sequentiality, knowledge soundness and zero-knowledge.

Completeness: For all $(\mathbf{pp}, x, y, \pi)$, where $y = \texttt{Eval}(\mathbf{pp}, x)$ and $\pi = \texttt{Prove}(\mathbf{pp}, x, y, \alpha)$, algorithm $\texttt{Verify}(\mathbf{pp}, x, y, \pi)$ outputs $\texttt{accept}$.

Sequentiality: Any parallel algorithm that uses at most $\text{poly}(\lambda)$ processors and outputs $y = \texttt{Eval}(\mathbf{pp}, x)$ with noticeable probability runs in time at least T.

Knowledge Soundness: For any adversary $\mathcal{A}$ that outputs a proof π for instance x of bit-length n, such that $\texttt{Verify}(\mathbf{pp}, x, \pi)$ outputs $\texttt{accept}$ with probability δ, there exists an extractor $\mathcal{E}$ that with probability at least $(\delta - \varepsilon)/\text{poly}(n)$ outputs element $y = \texttt{Eval}(\mathbf{pp}, x)$ in time less than T, where poly is some positive polynomial and $\varepsilon \in [0, 1]$ is called the *soundness error*.

Zero Knowledge: There exists a simulator $\mathcal{S}$ that, given instance x and randomness c^*, outputs a proof π^* in time less than T such that $\texttt{Verify}(\mathbf{pp}, x, \pi^*)$ outputs $\texttt{accept}$ and π^* is indistinguishable from an honestly computed proof.

5.5.2 Construction

Our construction of a zero-knowledge VDF can be found in Figure 5.6. We note that this construction can be transformed into a watermarkable zero-knowledge VDF by including a unique identifier in the computation of the randomness in the proof of knowledge of same discrete log. Since this extension is a straightforward combination of our two constructions, we refrain from analyzing it formally.

Theorem 12. *Let $\texttt{PoE}$ be a complete and sound proof of exponentiation. The algorithms in Figure 5.6 define a zero-knowledge VDF, relative to the iterated squaring assumption, the strong RSA assumption, the low order assumption and the decisional discrete log assumption with small exponents.*

Proof. Sequentiality of the VDF follows immediately from the iterated squaring assumption. Completeness follows by inspection of the protocol and from the completeness property of $\texttt{PoE}$. Knowledge soundness follows from the low order assumption, the strong RSA assumption and soundness of $\texttt{PoE}$. To see this, we describe an extractor $\mathcal{E}$ that outputs $y = x^{2^T}$ in time less than T by interacting with an adversary $\mathcal{A}$ that outputs an accepting proof

$$\pi = (x', y', \texttt{PoKsDLh}(\mathbf{pp}, x, x', y'), \texttt{PoE}(\mathbf{pp}, x', y', T)).$$

The extractor $\mathcal{E}$ first tries to extract an exponent r and a base element $\tilde{y}$ from $\texttt{PoKsDLh}$ such that $x^r = x'$ and $\tilde{y}^r = y'$. If this is not possible, it can break the strong RSA assumption

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp} = (\mathbb{G}, x_1^*, y_1^*, \texttt{PoE}(\mathbf{pp}, x_1^*, y_1^*, T), x_2^*, y_2^*, \texttt{PoE}(\mathbf{pp}, x_2^*, y_2^*, T), H)$
outputs a finite abelian group $\mathbb{G}$ of unknown order, a random square x_1^*, a random non-square x_2^* and the corresponding PoEs and an efficiently computable hash function H.

$\texttt{Eval}(\mathbf{pp}, x, T) \to (y, \alpha)$ outputs (y, α), where $y = x^{2^T}$ and α is an advice string for $\texttt{PoE}$.

$\texttt{Prove}(\mathbf{pp}, x, \texttt{ID}, y, \alpha) \to \pi$ outputs

$$\pi = (x', y', \texttt{PoKsDLh}(\mathbf{pp}, x, x', y'), \texttt{PoE}(\mathbf{pp}, x', y', T)),$$

where $x' := x^r$ and $y' := y^r$ for a uniformly random $r \leftarrow \pm[2^\lambda]$.

$\texttt{Verify}(\mathbf{pp}, x, \pi) \to \texttt{accept/reject}$ checks if $\texttt{PoKsDLh}(\mathbf{pp}, x, x', y')$ and $\texttt{PoE}(\mathbf{pp}, x', y', T)$ verify.

$\texttt{Sim}(\mathbf{pp}, x, c^*) \to \pi^*$ on input $\mathbf{pp}, x, c^*$, simulates $\texttt{PoKsDLh}(\mathbf{pp}, x, x^*, y^*)$ with randomness c^* and outputs

$$\pi^* = (x^*, y^*, \texttt{PoKsDLh}(\mathbf{pp}, x, x^*, y^*), \texttt{PoE}(\mathbf{pp}, x^*, y^*, T)),$$

for $x^* := x_1^*$, if x is a square and $x^* \leftarrow \{x_1^*, x_2^*\}$ uniformly random if x is a non-square.

Figure 5.6: A zero-knowledge VDF from any proof of exponentiation. $\texttt{PoKsDLh}$ is the non-interactive version of the proof of knowledge presented in Figure 5.4. By $\texttt{PoE}(\mathbf{pp}, x, y, T)$ we denote the chosen proof of exponentiation with group parameters $\mathbf{pp}$ and statement $x^{2^T} = y$.

similar to the proof of Theorem 9. Assume that $\tilde{y} \neq y$. Then we would have that $(\tilde{y}/y)^r = 1$ and hence $\tilde{y}/y$ would be an element of low order. Hence, by the low order assumption $\tilde{y} = y$. Since the running time of the extractor is independent of T, knowledge soundness follows.

It remains to prove zero knowledge. Consider the simulator $\texttt{Sim}$. From the zero-knowledge property of $\texttt{PoKsDLh}$ and the decisional discrete log assumption with small exponents, we follow that the simulated proof π^* is computationally indistinguishable from an honest proof. □

The next corollary follows from the discussion in Remark 4.

Corollary 3. *Let $\texttt{PoE}$ be a complete and sound proof of exponentiation and let $\mathbb{G} = \mathrm{QR}_N^+$, where N is a safe prime modulus. The construction in Figure 5.6 is a zero-knowledge VDF in $\mathbb{G}$ relative to the iterated squaring assumption and the strong RSA assumption.*

Efficiency Transforming a PoE into a zero-knowledge VDF with the construction in Figure 5.6 increases the complexity of the underlying PoE scheme as follows:

- The proof size grows by $4\lambda + 2$ group elements and λ many integers of size at most $2^{3\lambda+1}$.

- The verifier needs to perform 4λ additional small group exponentiations (with exponents of size at most $2^{3\lambda+1}$) and 3λ group multiplications.
- The prover needs to perform $3\lambda + 2$ additional small exponentiations (with exponents of size at most $2^{3\lambda}$).

5.6 Zero-Knowledge Proofs of Sequential Work

Our construction of a general zero-knowledge VDF is not practical, with the proof of knowledge of y being the bottleneck. Without it our construction does not satisfy the definition of a zero-knowledge VDF: If the prover just needed to output x, x', y' and a proof of knowledge of r such that $x^r = x'$, then it could first raise x to a random r and then compute $y' = (x')^{2^T}$. In particular, it would produce the output without ever knowing y. The main observation in this section is that, while this protocol does not satisfy the definition of a zero-knowledge VDF, it is still sufficient for the application to short-lived proofs presented in [ABC22, Section 7] because the prover still needs at least T steps to compute the output. The protocol can be found in Figure 5.7 and the application to short-lived proofs in the next section.

5.6.1 Definition

Definition 18. A zero-knowledge proof of sequential work is a set of algorithms ($\texttt{Setup}$, $\texttt{Prove}$ $\texttt{Verify}$, $\texttt{Sim}$), where

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ on input statistical security parameter 1^λ and time parameter T outputs public parameters $\mathbf{pp}$.

$\texttt{Prove}(\mathbf{pp}, x, y, \alpha) \to \pi$ outputs a proof π of sequential work of T steps.

$\texttt{Verify}(\mathbf{pp}, x, \pi) \to \texttt{accept/reject}$ checks that π is a valid proof of sequential work.

$\texttt{Sim}(\mathbf{pp}, x, c^*) \to \pi^*$ outputs a simulated proof of sequential work π^* using randomness c^*.

The algorithm $\texttt{Eval}$ is deterministic and can compute the output y in T sequential steps. A zero-knowledge proof of sequential work must additionally satisfy four properties: Completeness, sequentiality, soundness and zero-knowledge.

Completeness: For all $(\mathbf{pp}, x, y, \pi)$, where $y = \texttt{Eval}(\mathbf{pp}, x)$ and $\pi = \texttt{Prove}(\mathbf{pp}, x, y, \alpha)$, algorithm $\texttt{Verify}(\mathbf{pp}, x, \pi)$ outputs $\texttt{accept}$.

Sequentiality: Any parallel algorithm that uses at most poly(λ) processors and outputs a proof π', such that $\texttt{Verify}(\mathbf{pp}, x, \pi')$ outputs $\texttt{accept}$ with noticeable probability runs in time at least T.

Zero Knowledge: There exists a simulator $\mathcal{S}$ that, given instance x and randomness c^*, outputs a proof π^* in time less than T such that $\texttt{Verify}(\mathbf{pp}, x, \pi^*)$ outputs $\texttt{accept}$ and π^* is indistinguishable from an honestly computed proof.

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp} = (\mathbb{G}, x_1^*, y_1^*, \texttt{PoE}(\mathbf{pp}, x_1^*, y_1^*, T), x_2^*, y_2^*, \texttt{PoE}(\mathbf{pp}, x_2^*, y_2^*, T), H)$
outputs a finite abelian group $\mathbb{G}$ of unknown order, a random square x_1^*, a random non-square x_2^* and the corresponding PoEs and an efficiently computable hash function H.

$\texttt{Prove}(\mathbf{pp}, x) \to \pi$ outputs

$$\pi := (x', y', \texttt{PoKDL}(\mathbf{pp}, x, x'), \texttt{PoE}(\mathbf{pp}, x', y', T)),$$

where $x' := x^r$ and for some uniformly random $r \leftarrow \pm[2^\lambda]$.

$\texttt{Verify}(\mathbf{pp}, x, \pi) \to \texttt{accept/reject}$ checks if both $\texttt{PoKDL}(\mathbf{pp}, x, x')$ and $\texttt{PoE}(\mathbf{pp}, x', y', T))$ verify.

$\texttt{Sim}(\mathbf{pp}, x, c^*) \to \pi^*$ on input $\mathbf{pp}, x, c^*$, simulates $\texttt{PoKDL}(\mathbf{pp}, x, x^*)$ with randomness c^* and outputs

$$\pi^* = (x^*, y^*, \texttt{PoKDL}(\mathbf{pp}, x, x^*), \texttt{PoE}(\mathbf{pp}, x^*, y^*, T)),$$

for $x^* := x_1^*$, if x is a square and $x^* \leftarrow \{x_1^*, x_2^*\}$ uniformly random if x is a non-square.

Figure 5.7: A Zero-Knowledge Proof of Sequential Work from any proof of exponentiation PoE. PoKDL is the non-interactive version of the proof of knowledge presented in Figure 5.1. By $\texttt{PoE}(\mathbf{pp}, x, y, T)$ we denote the chosen proof of exponentiation with group parameters $\mathbf{pp}$ and statement $x^{2^T} = y$.

5.6.2 The Generalized Iterated Squaring Assumption

For the security of our construction we need to make the following assumption.

Definition 19 (generalized iterated squaring assumption)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *generalized iterated squaring assumption* holds for GGen if, for any probabilistic parallel algorithm $\mathcal{A}$ that uses at most poly(λ) processors and runs in time less than T, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$, a random group element x and an integer T.
2. $\mathcal{A}$ outputs a pair $(r, y) \in \mathbb{Z} \times \mathbb{G}$.
3. $\mathcal{A}$ wins if and only if $r \neq 0$ and $y = (x^r)^{2^T}$.

5.6.3 Construction

Theorem 13. *Let PoE be a complete and sound proof of exponentiation. The algorithms in Figure 5.7 define a zero-knowledge PoSW, relative to the generalized iterated squaring assumption, the strong RSA assumption and the decisional discrete log assumption with small exponents.*

Proof. Completeness follows by inspection of the protocol and from the completeness property of $\texttt{PoE}$. Sequentiality of the PoSW follows from the generalized iterated squaring assumption, the RSA assumption and soundness of $\texttt{PoE}$: Assume that an adversary $\mathcal{A}$ can output π in time less than T. We construct an adversary $\mathcal{B}$ that breaks either the RSA assumption or the generalized iterated squaring assumption as follows:

1. $\mathcal{B}$ obtains as input a the description of a group $\mathbb{G}$ and a generalized iterated squaring challenge $x \in \mathbb{G}$.
2. $\mathcal{B}$ forwards $\mathbb{G}$ and x to adversary $\mathcal{A}$.
3. If $\mathcal{A}$ is successful, it outputs a valid proof
$$\pi = (x', y', \texttt{PoKDL}(\mathbf{pp}, x, x'), \texttt{PoE}(\mathbf{pp}, x', y', T)).$$
4. $\mathcal{B}$ first tries to extract the secret exponent r from $\texttt{PoKDL}$. If this is not possible, it can use $\mathcal{A}$ to break the RSA assumption similar to the proof of Theorem 7.
5. If it is possible, $\mathcal{B}$ outputs (r, y') to break the generalized iterated squaring assumption.

The running time of $\mathcal{B}$ is independent of T. By soundness of $\texttt{PoE}$ we have that $\mathcal{B}$ breaks one of the two assumptions with the same probability as the winning probability of $\mathcal{A}$. It remains to prove zero knowledge. Consider the simulator $\texttt{Sim}$. From the zero-knowledge property of $\texttt{PoKDL}$ and the decisional discrete log assumption with small exponents, we follow that the simulated proof π^* is computationally indistinguishable from an honest proof. □

The next corollary follows from the discussion in Remark 4.

Corollary 4. *Let $\texttt{PoE}$ be a complete and sound proof of exponentiation and let $\mathbb{G} = \mathrm{QR}_N^+$, where N is a safe prime modulus. The construction in Figure 5.7 is a zero-knowledge PoSW in $\mathbb{G}$ relative to the generalized iterated squaring assumption and the strong RSA assumption.*

Efficiency Transforming a PoE into a zero-knowledge proof of sequential work with the construction in Figure 5.7 increases the complexity of the underlying PoE scheme as follows:

- The proof size grows by 3 group elements and one integer of size at most $2^{3\lambda+1}$.
- The verifier needs to perform 2 additional small group exponentiations (with exponents of size at most $2^{3\lambda+1}$) and 1 group multiplications.
- The prover needs to perform 3 additional small exponentiations (with exponents of size at most $2^{3\lambda}$).

5.7 Short-Lived Proofs from our Zero-Knowledge PoSW

In this section we discuss how one can use our zero knowledge PoSW in the short-lived proof construction of [ABC22]. The main idea in the construction of [ABC22] is to transform any

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp} = (\mathbf{pp}_{\texttt{zkPoSW}}, \mathbf{pp}_{\mathcal{R}}) = (\mathbb{G}, b^*, y^*, \pi^*_{\texttt{PoE}}, H, \mathbf{pp}_{\mathcal{R}})$ outputs a finite abelian group $\mathbb{G}$ of unknown order, a uniformly random $b^* \leftarrow \mathbb{G}$, $y^* = (b^*)^{2^T}$, $\pi^*_{\texttt{PoE}} = \texttt{PoE}(\mathbf{pp}_{\texttt{zkPoSW}}, b^*, y^*, T)$, an efficiently computable hash function H and the public parameters of the proof system for $\mathcal{R}$.

$\texttt{Prove}(\mathbf{pp}, T, x, b, w) \to (\pi_{\texttt{zkPoSW}}, \pi_{\mathcal{R}})$ outputs

- Forged
$$\pi_{\texttt{zkPoSW}} = (b^*, y^*, \texttt{PoKDL}(\mathbf{pp}_{\texttt{zkPoSW}}, b, b^*), \pi_{\texttt{PoE}}),$$
where $\texttt{PoKDL}(\mathbf{pp}_{\texttt{zkPoSW}}, b, b^*)$ is forged with challenge c_1.
- An honestly computed proof $\pi_{\mathcal{R}}$ with random challenge c_2 such that $c_1 + c_2 = c = H(x, b, a)$, where a is the first element of $\pi_{\mathcal{R}}$.

$\texttt{Forge}(\mathbf{pp}, T, x, b) \to (\tilde{\pi}_{\texttt{zkPoSW}}, \tilde{\pi}_{\mathcal{R}})$ outputs

- Honestly computed
$$\tilde{\pi}_{\texttt{zkPoSW}} = (b', y', \texttt{PoKDL}(\mathbf{pp}_{\texttt{zkPoSW}}, b, b'), \texttt{PoE}(\mathbf{pp}_{\texttt{zkPoSW}}, b', y', T))$$
with random challenge c_1.
- A forged proof $\tilde{\pi}_{\mathcal{R}}$ with challenge c_2 such that $c_1 + c_2 = c = H(x, b, a)$, where a is the first element of $\pi_{\mathcal{R}}$.

$\texttt{Verify}(\mathbf{pp}, x, \pi_{\texttt{zkPoSW}}, \pi_{\mathcal{R}}) \to \texttt{accept/reject}$ checks $\pi_{\texttt{zkPoSW}}$ and $\pi_{\mathcal{R}}$ and outputs `accept` if and only if both proofs verify.

Figure 5.8: A short-lived proof from our zero knowledge PoSW. PoKDL is the non-interactive version of the proof of knowledge presented in Figure 5.1. By $\texttt{PoE}(\mathbf{pp}, x, y, T)$ we denote the chosen proof of exponentiation with group parameters $\mathbf{pp}$ and statement $x^{2^T} = y$.

sigma protocol Σ for a relation $\mathcal{R}$ into a short lived proof for $\mathcal{R}$ by combining Σ with a zero-knowledge VDF (which is also a sigma protocol) via the standard OR combination of sigma protocols. Since anyone can construct a valid VDF proof in T steps, the combined proof loses its validity after time $T \cdot \text{poly}(\lambda)$. The zero-knowledge property of the VDF is needed since an honest prover needs to be able to forge a VDF proof in time less than T, which is indistinguishable from an honest proof also after time $T \cdot \text{poly}(\lambda)$ has passed. We first recall some facts about sigma protocols before presenting our construction of a short-lived proof.

5.7.1 Sigma Protocols

Definition 20 (sigma protocol)**.** A *sigma protocol* (Σ-protocol) is an interactive honest verifier zero-knowledge proof of knowledge consisting of three messages:

- a first message by $\mathcal{P}$ denoted by u,
- a second message by $\mathcal{V}$ denoted by c and
- a third message by $\mathcal{P}$ denoted by z.

Cramer, Damgård and Schoenmakers [CDS94] showed that the set of relations with Σ-protocols is closed under disjunction: Let $\Sigma_1 = (u_1, c_1, z_1)$ be a sigma protocol for relation $\mathcal{R}_1$ and $\Sigma_2 = (u_2, c_2, z_2)$ be a sigma protocol for relation $\mathcal{R}_2$ and let x_1 be an instance of $\mathcal{R}_1$ and x_2 an instance of $\mathcal{R}_2$. The following protocol is an honest verifier zero-knowledge proof of knowledge of either a witness w_1 for x_1 or a witness w_2 for x_2. Assume without loss of generality that $\mathcal{P}$ knows witness w_1.

1. $\mathcal{P}$ picks a random c_2 and simulates $\Sigma_2 = (u_2, c_2, z_2)$.
2. $\mathcal{P}$ computes the message u_1 and sends (u_1, u_2) to $\mathcal{V}$.
3. $\mathcal{V}$ sends a random message c to $\mathcal{P}$.
4. $\mathcal{P}$ computes $c_1 = c \oplus c_2$, computes the honest third message z_1 and sends (z_1, z_2) to $\mathcal{V}$.
5. $\mathcal{V}$ accepts if and only if $c_1 \oplus c_2 = c$ and the transcripts for both Σ_1 and Σ_2 are valid.

5.7.2 Our Construction

In this section we show how to transform any sigma protocol Σ for a relation $\mathcal{R}$ into a short-lived proof for relation $\mathcal{R}$. The protocol can be found in Figure 5.8. It differs from the construction of [ABC22] in three ways:

- We don't work with a zero-knowledge VDF but a zero-knowledge PoSW. This is possible because the protocol does not need the uniqueness property of the VDF.
- We need to include a precomputed PoE in the public parameters because the honest prover simulates the outputs of the zkPoSW and in our construction the simulator needs a precomputed PoE.
- Our zkPoSW is not a sigma protocol but the proof of knowledge `PoKDL` is. In our construction it is sufficient to combine Σ and `PoKDL` via the standard disjunction of sigma protocols.

Using Pietrzak's PoE [Pie19] one can not only re-randomize the precomputed PoEs but also the PoEs needed for the forged proofs. Hence, it achieves much faster forging times than the construction based on a zero-knowledge version of Wesolowksi's proof given in [ABC22].

5.8 Conclusion

In this chapter we have seen how to efficiently watermark any proof of exponentiation to obtain practical watermarkable VDFs. We also constructed practical zero-knowledge proofs of sequential work that can be used to build short-lived proofs for any NP statement with fast forging times. Our zero-knowledge VDF construction is asymptotically efficent but not practical: The proof size grows by a factor λ because the proof of knowledge that is being used as a building block needs λ repetitions to be sound. One interesting open problem that remains is to construct a *practical* zero-knowledge version of Pietrzak's VDF, by either removing the need for λ repetitions in our general construction or by working directly with Pietrzak's protocol.

CHAPTER 6

Primality Testing from Proofs of Exponentiations

6.1 Introduction

The search for giant primes has long focussed on primes of special forms due to the availability of faster, custom primality tests.Two of the most well-known examples are

Mersenne numbers of the form $M_n = 2^n - 1$, for some $n \in \mathbb{N}$, which can be tested using the Lucas-Lehmer or the Lucas-Lehmer-Reisel test [Luc78, Leh27, Rie69]; and

Proth numbers of the form $P_{k,n} = k2^n + 1$, for some $n \in \mathbb{N}$ and odd $k \in \mathbb{N}$, which can be tested using Proth's theorem [Pro78].

To harness the computational resources required for finding giant primes, there are massive distributed projects like GIMPS (Great Internet Mersenne Prime Search) and PrimeGrid dedicated to the search for giant primes of special forms, including the ones above. A volunteer in such a distributed project can download an open-source software that locally carries out primality tests on candidate numbers, at the end of which, a candidate is either rejected as a composite number or confirmed as a new prime. The largest-known prime as of now is a Mersenne prime ($2^{82,589,933} - 1$) with 24,862,048 decimal digits found by GIMPS [GIM18].

Testing primality of giant numbers. The search for large primes is a time-consuming process: the GIMPS website warns that a single primality test could take up to a month. The reason for this is that these tests – whenever the prime candidate has no small prime factors[1] – require the computation of a very long sequence modulo an extremely large number. For example, Proth's theorem [Pro78] states that $P_{k,n} = k2^n + 1$ is prime if and only if, for a quadratic non-residue x modulo $P_{k,n}$, it holds that

$$x^{k2^{n-1}} \equiv -1 \bmod P_{k,n}. \tag{6.1}$$

[1] GIMPS first tests by trial division whether a candidate number has any prime divisors of size up to a bound between 2^{66} and 2^{81}. Only when this is not the case is that they run a more expensive specialized primality test: details can be found on this page.

To date, the largest-known Proth prime is $10223 \cdot 2^{31,172,165} + 1$ [Pri16]. Since n is of the order of magnitude 10^7 and the square-and-multiply algorithm is the fastest way currently known to carry out exponentiation, the test roughly requires 10^7 squarings modulo a 10^7-digit modulus. Unfortunately, performing this test does not yield an immediate witness that certifies the correctness of the result – in particular, if $P_{k,n}$ is composite, the test does not find a divisor of $P_{k,n}$.[2] Until very recently, the standard way for another party to independently validate the test result was by recomputing the result of Equation (6.1). In 2020, Pavel Atnashev demonstrated that proofs of exponentiation might be applicable in the context of these specialized primality tests to avoid the costly second recomputation.[3]

PoEs and efficient verification of primality tests. Since the primality test using Proth's theorem amounts to iterated exponentiation, it seems immediate that one would attempt to exploit PoEs also towards efficient verifiability in the context of primality tests for giant numbers. The idea is for the volunteer to use the (non-interactive) PoE to compute – alongside the result of the test – a proof that helps any other party verify the result. For this approach to be feasible,

1. computing the proof should not require much more additional resource (relative to the iterated exponentiation induced by the specialized primality test), and
2. the cost of verifying a proof should be significantly lower than that of recomputing the exponentiation.

Recently, this approach has been deployed in both GIMPS [GIM20] and PrimeGrid [Pri20], where (non-interactive) Pietrzak's PoE [Pie19] is used to certify (both primality and non-primality) of Mersenne and Proth numbers when used along with Lucas-Lehmer-Riesel test and Proth's theorem, respectively. In fact, one of the recently-found Proth primes, $68633 \cdot 2^{2715609} + 1$, has been certified so.

However, PoEs were constructed for groups whose order is hard to compute like, e.g., RSA group [RSA78] or class group [BW88]. If one party knows the group order, they can not only speed up the exponentiation but also (in many groups) construct *false* Pietrzak PoEs that lead a verifier to accept proofs for false statements. In the context of primality testing the underlying group is $\mathbb{Z}_{P_{k,n}}$, so the group order is known whenever $P_{k,n}$ is prime. While this does not speed up the computation of the primality test (since the modulus is larger than the exponent), it removes the soundness guarantee of the protocol. As we discuss next, a malicious prover can falsely convince a verifier that *any* Proth prime is composite using Pietrzak's protocol in those groups.

6.1.1 Our Contribution

The statistical security guarantee of Pietrzak's PoE applies only to groups where the low order assumption holds. This presents an issue with its usage in the GIMPS and PrimeGrid projects since there are no guarantees on the structure of the group in these applications. In

[2] Note that some primality tests, like, e.g., Miller-Rabin [Mil76, Rab80], can be modified to (sometimes) yield factors in case the number being tested is not a prime.

[3] More details can be found in this thread of mersenneforum.org. An implementation due to Atnashev is available on GitHub. The idea of using PoEs for certifying giant primes has been discussed also by Mihai Preda in another thread in the same forum already in August 2019

fact, if $P_{k,n}$ is prime, the order of the group is $P_{k,n} - 1 = k2^n$ so low-order elements (e.g., of order 2) do exist and can be found without much effort. We show in Section 6.3 how a malicious volunteer can exploit the attack from [BBF24] to generate a proof that "certifies" an arbitrary Proth *prime* as composite with constant probability. Indeed, people at GIMPS and PrimeGrid were aware of this [Atn22] and Pietrzak's PoE is currently employed in these projects more-or-less as a checksum to catch benign errors (e.g., hardware errors). When a volunteer is malicious and deliberately tries to mislead the project, there are no guarantees. This could force the volunteer network to waste additional computation and possibly postpone the discovery of another giant prime by years.

Are Cryptographically-Sound Certificates Possible?

In our work, we explore whether any cryptographic guarantee for *practical* proofs is possible in the above scenarios. Whilst it is theoretically possible to use existing results to certify non-primality, these measures, as we discuss in Section 6.1.1, turn out to be too expensive. As a first step towards practical proofs, we show how to achieve soundness for proving *non-primality* of Proth numbers. That is, we construct an interactive protocol for the language

$$L := \left\{(k,n) \in \mathbb{N}^2 : k \text{ is odd and } P_{k,n} = k2^n + 1 \text{ is not a prime}\right\}. \tag{6.2}$$

While ideally, one would want to certify both primality and non-primality, the latter is much more important for projects like GIMPS and PrimeGrid: they worry about missing out on primes rather than false claims stating that a composite is a prime. Primes are very sparse[4], so double checking claims of primality is not a problem, but performing *each* primality check twice to catch benign errors or a malicious volunteer is almost twice as expensive as using a sound non-primality test as suggested in this work.

Our interactive protocol has statistical soundness: if the candidate number to be tested is indeed a Proth prime, then even a computationally unbounded malicious prover (a malicious volunteer) will not be able to convince the verifier (say a server run by the project) that it is composite.

Theorem 14 (Informal)**.** *There is a practical public-coin statistically-sound interactive proof for the non-primality of Proth numbers.*

We provide an overview of our interactive protocol in the next section. Since it is public-coin, our interactive protocol can be made non-interactive via the Fiat-Shamir transform [FS87]. In general, the Fiat-Shamir transform only works for constant round protocols, which is not the case for our protocol, so showing that Fiat-Shamir works in our case needs a proof.

Corollary 5 (Informal)**.** *In the random-oracle model, there is a practical statistically-sound non-interactive proof for the non-primality of Proth numbers.*

Concrete Efficiency

We defer exact details about the complexity of our protocol to Section 6.4.3. Here, we provide concrete (worst-case) numbers for our non-interactive proof using the largest Proth prime known to date as the candidate: $10223 \cdot 2^{31172165} + 1$ [Pri16]. For $k = 10223$, $n = 31172165$ and security parameter $\lambda = 80$:

[4]For $N \in \mathbb{N}$, let $\pi(N)$ denote the number of primes less than N. By the prime number theorem, asymptotically $\pi(N)$ approaches $N/\log(N)$. For the case of Proth numbers, however, even the question of whether there are infinitely many of them is open [BKT22].

- the prover (additionally) stores 5584 group elements (which is around 20GB) and performs 13188 multiplications;
- the verifier performs 10046 multiplications; and
- the proof size is 26 elements of size 31172179, i.e., around 102 MB.

Note that recomputing the result of the primality test would take $n = 31172165$ multiplications, so our protocol reduces the number of multiplications by a factor of $\lfloor 31172165/(13188+10046) \rfloor = 1341$. Note that this takes the order of hours rather than days. In Section 6.4.4, we show that the additional cost of our protocol compared to the one that is being used now (which is not cryptographically sound) is moderate: In the above example, the prover performs 2021 and the verifier 4046 multiplications more than in the current implementation.

Applicability of Existing Statistically-Sound PoEs.

The issue with low-order elements when using Pietrzak's PoE out-of-the-box can be resolved using alternative PoEs that are statistically sound in *arbitrary* groups.[5] Indeed, such PoEs were recently proposed [BHR+21, HHK+22]. [BHR+21] can be regarded as a parallel-repeated variant of Pietrzak's protocol but, to achieve statistical soundness, the number of repetitions is as large as the security parameter. This leads to significant overhead in terms of both proof-size and computation. For example, to compute the PoE of [BHR+21] for the Proth prime from Section 6.1.1, the prover needs to perform 893312 multiplications and it takes the verifier 318800 multiplications to verify the proof consisting of 2080 group elements (i.e., 8160 MB). This means that our protocol reduces the number of multiplications of [BHR+21] by a factor of 52 and the proof size by a factor of 80. The overall approach in [HHK+22] is similar to that in [BHR+21], but it improves on the complexity of [BHR+21] whenever it is possible to choose the exponent to be a large q of a special form. In the primality testing application, we do not have the freedom to choose the exponent and, for the case of $q = 2$, the complexity of [HHK+22] is comparable to that of [BHR+21].

6.1.2 Technical Overview

Our starting point is Pietrzak's PoE (PPoE, Figure 2.2). Applying PPoE out-of-the-box as a certificate of non-primality for a Proth number $P_{k,n}$ is not sound since the group $\mathbb{Z}^*_{P_{k,n}}$ might have easy-to-find elements of low order. We show in Section 6.3 that this is indeed the case and it is not hard to generate PPoE proofs that "certify" a prime $P_{k,n}$ as composite.

Working around low-order elements. The way low-order elements are dealt with in [BHR+21, HHK+22] is via parallel repetition and/or by working with exponents q of a particular form. As explained in Section 6.1.1, we cannot exploit either of these techniques because of efficiency reasons and the restriction on the exponent placed by the primality test. Nevertheless, our interactive protocol, described in Figures 6.1 and 6.3, builds on some of the ideas in [Pie19, HHK+22] to get around the issue of low-order elements for the specific exponentiation considered in Proth's test (Equation (6.1)). Below, we give an overview of how this is accomplished – we refer the readers to Section 6.4 for a more detailed overview.

[5] One could also use SNARGs [Kil92, Mic94] for this purpose but, being a general-purpose primitive, the resulting schemes would not be practically efficient.

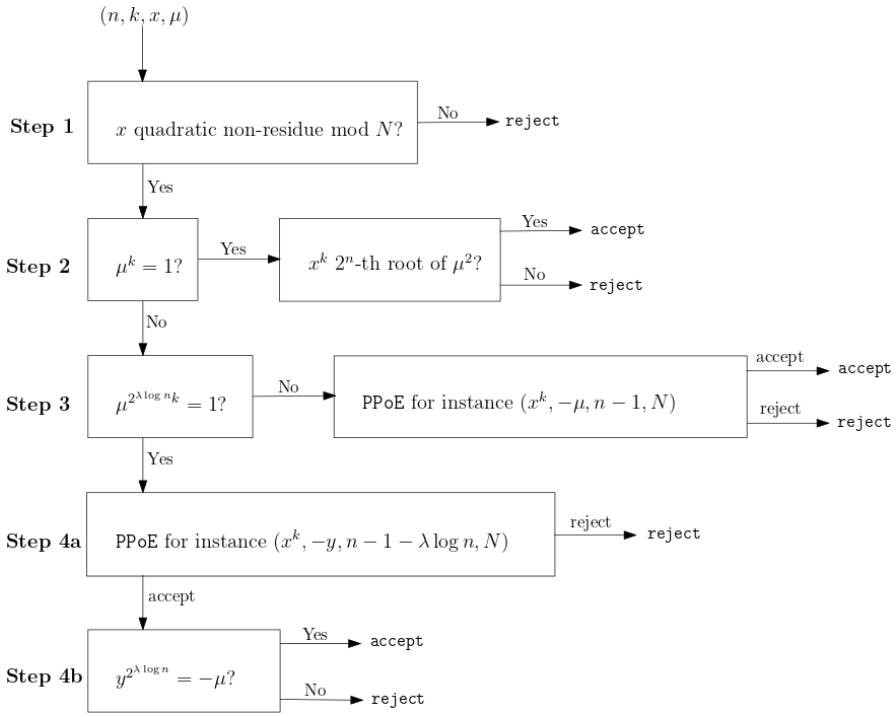


Figure 6.1: Overview of the protocol in Figure 6.3. All computations are done in the group $\mathbb{Z}_N^*$, where $N = k2^n + 1$.

For a prime $N := P_{k,n}$, suppose that $x \in \mathbb{Z}_N^*$ is a quadratic non-residue.[6] Suppose that a malicious prover $\mathcal{P}^*$ tries to convince the verifier $\mathcal{V}$ that

$$x^{k2^{n-1}} \equiv -\mu \bmod N, \quad 1 \neq \mu \in \mathbb{Z}_N^*. \tag{6.3}$$

Since N is a prime and the result must be -1 by Proth's theorem, the statement $(x, -\mu, k2^{n-1})$ corresponding to Equation (6.3) is μ-false. Our protocol exploits the fact that $\mathcal{V}$ does not care about the exact value of $x^{k2^{n-1}}$ and it rejects as long as the correct result is not equal to -1. This observation greatly simplifies the task for $\mathcal{V}$. As we show, it is sufficient to perform a few efficient checks on the order of μ, depending on which $\mathcal{V}$ can choose a sound method for verification.

- Our starting point is the case where $\mathrm{ord}(\mu)$ is "large", by which we mean $\mathrm{ord}(\mu) \nmid k2^{\lambda \log(n)}$ (Step 3). We show in this case that it is possible to use PPoE out of the box to prove the statement $(x^k, -\mu, n-1)$, which is equivalent to the statement in Equation (6.3). Key to proving this is the following observation on the fine-grained nature of soundness of PPoE: the "falseness" of a statement in each round of the sub-protocol cannot decrease by too much. More precisely, if the cheating prover starts with an α-false statement then the new statement is β-false for some β whose order cannot be much smaller than α's. Therefore, if the cheating prover starts off with a statement that is sufficiently false, which turns out to be when $\mathrm{ord}(\alpha) = k2^{\lambda \log(n)}$, then the statement in the final round remains false with overwhelming probability and is rejected by the PPoE verifier. We formalise this observation in Lemma 7 and point out

[6] In the actual protocol, we explain how the verifier can check if the Jacobi symbol of x is -1 (Step 1 in Figure 6.3).

that, while a similar lemma was proved in [HHK+22], there are some crucial differences: see Remark 7.

- Next is the case where $\mathrm{ord}(\mu)$ is "small and odd" (Step 2), i.e., the order is some divisor d of k. In this case, $\mathcal{V}$ can verify the statement in Equation (6.3) *without* any help from $\mathcal{P}^*$ as follows: find the inverse of 2^n modulo d and raise μ^2 to that element.[7] By Equation (6.3), this yields the same element as x^k if the prover is honest. If N is prime, this will yield a different element than x^d as we show in Section 6.4.2. This quick verification is only possible since d and 2^n are coprime, so it can only be used in this case.

- Finally, consider the case where the order is "small and even", by which we mean $\mathrm{ord}(\mu) \mid 2^{\lambda \log(n)}$ (Step 4). Here, we are in a situation where PPoE does not guarantee soundness (since the statement is not "false enough"). However, as in [HHK+22], it is possible to reduce the task of checking Equation (6.3) to that of verifying, using PPoE, the "smaller" statement obtained by taking the $2^{\lambda \log(n)}$-th root of Equation (6.3). To be precise, $\mathcal{P}$ and $\mathcal{V}$ verify the statement $(x^k, y, n-1-\lambda\log(n))$ using PPoE and, if convinced, $\mathcal{V}$ then checks whether $y^{2^{\lambda \log(n)}} = -\mu$, by itself, using a final exponentiation. This final exponentiation forces a malicious prover $\mathcal{P}^*$ to cheat with an element of high enough order during the PPoE. To see this, assume for example that $\mathcal{P}^*$ sends the honest result $y = x^{2^{n-1-\lambda\log(n)}}$. Then, $\mathcal{V}$'s final exponentiation leads to outright rejection since

$$y^{2^{\lambda \log(n)}} = (x^{k2^{n-1-\lambda\log(n)}})^{2^{\lambda\log(n)}} = x^{k2^{n-1}} \neq -\mu.$$

  On the other hand, $\mathcal{P}^*$ cheating with an element of such high order during the PPoE makes the verifier reject this PPoE with overwhelming probability (as in the first case).

We refer the reader to Sections 6.2 and 6.4 for the formal analysis.

6.1.3 More Related Work

General-purpose primality testing. Pratt showed that primality testing (of arbitrary integers) lies in the class **NP**, via the eponymous Pratt certificates [Pra75] (an alternative certificate of primality is the Atkin-Goldwasser-Kilian-Morain certificate [AM93, GK86]). Coupled with the fact that non-primality has succinct certificates in the form of factorization (which can be efficiently checked by integer multiplication) placed primality testing in **NP** ∩ **co-NP**. Probabilistic tests like Solovay-Strassen [SS77], Miller-Rabin [Mil76, Rab80] and Baillie–PSW [BW80, PSW80] soon followed, which placed primality testing in classes like **BPP**, **RP** or **ZPP**.[8] Finally, Agrawal, Kayal, and Saxena [AKS04] settled the question by showing that primality testing is in **P**. We refer the readers to [AKS04] for a more detailed exposition on (general-purpose) primality testing.

Giant prime numbers and custom primality tests. In addition to Mersenne numbers M_n and Proth numbers $P_{k,n}$, numbers of special form that have been targetted in the search for giant primes include Fermat numbers $F_n := 2^{2^n} + 1$ (which are a special case of Proth numbers), generalised Fermat numbers $F_{a,b,n} := a^{2^n} + b^{2^n}$ and Woodall numbers $W_n :=$

[7] Note that if μ has order d, then $-\mu$ has order $2d$ (which is not coprime to 2^{n-1}), which is the reason we have to square the statement in Equation (6.3) before computing the inverse of the exponent.

[8] In fact, Miller's test [Mil76] runs in strict polynomial time assuming the Generalised Riemann Hypothesis.

$n \cdot 2^n - 1$. We refer the readers to PrimePages for a more comprehensive list. These numbers of special forms are amenable to custom primality tests that run faster than general-purpose primality tests. For example, the Lucas-Lehmer (LL) test [Luc78, Leh27] is a determinstic primality test for M_n that runs in time $O(n \cdot \mu(n))$, where $\mu(n)$ denotes the complexity of multiplying two n-bit integers.[9] In comparison, for M_n, the complexity of deterministic AKS primality test is $\tilde{O}(n^6)$ and the complexity of probabilistic Miller-Rabin test is $O(\lambda n \cdot \mu(n))$ (for a statistical error of $2^{-\lambda}$). GIMPS relies on the Lucas-Lehmer-Riesel [Rie69] test, which is a generalization of the Lucas-Lehmer test for numbers of the form $k2^n - 1$. PrimeGrid performs a variety of primality tests including Proth's theorem [Pro78] for Proth numbers. They were first to realize that (Pietrzak's) PoE can be used to certify the results of Proth's primality test [Pri20]. They also noticed that low-order elements can affect the soundness of the protocol and, therefore, included some checks on the order of the result [Atn22]. For Proth number $P_{k,n}$, *given* a quadratic non-residue modulo $P_{k,n}$, the complexity of Proth's test [Pro78] is $O(\log(k) \cdot n\mu(n))$; otherwise it is a Las Vegas test (since we currently know how to generate a quadratic non-residue only in expected polynomial-time). An alternative is to use the deterministic Brillhart-Lehmer-Selfridge test [BLS75].

6.2 Pietrzak's PoE in Groups of Known Order

In this section, we recall Pietrzak's PoE (PPoE) [Pie19] and some of its properties. The protocol is presented in Figure 2.2. By inspection of the protocol we see that it has perfect completeness. Pietrzak proved the following complexity results in [Pie19]:

Proposition 2 ([Pie19, Section 6.2]). *On instance $(x, y, T, \mathbb{G})$ PPoE has the following efficiency properties:*

1. *$\mathcal{V}$ performs $3\lambda \log T$ multiplications in $\mathbb{G}$.*
2. *$\mathcal{P}$ performs $2\sqrt{T}$ multiplications in $\mathbb{G}$ and stores $\sqrt{T}$ group elements to compute the proof.*
3. *The size of the proof is $\log T$ elements of $\mathbb{G}$.*

Furthermore, Pietrzak proved that the PoE is statistically sound in groups without low-order elements, in particular safe prime RSA groups. Boneh et al. later proved computational soundness in groups where it is hard to find low-order elements (the low-order assumption) [BBF24]. Ideally, we would like to use PPoE in a group of known order, where an adversary can find low-order elements in polynomial time. However, Boneh et al. showed that this is not sound by presenting an attack with low-order elements in [BBF24]. In the following section, we analyze in what way these low-order elements affect the soundness of PPoE.

6.2.1 (Non-)Soundness

We analyze the soundness of PPoE in groups of known order. Assume that the correct result of an exponentiation is $x^{2^T} = y \mod N$ but $\tilde{\mathcal{P}}$ claims that for some $\alpha \neq 1 \mod N$ it is $x^{2^T} = y\alpha \mod N$. We sometimes call α the "bad" element and say that the second statement is *α-false*. Note that the prover's statement is of this form without loss of generality

[9] Since these numbers have a succinct representation, the complexity of these tests is, strictly-speaking, exponential in the size of the input (which is n for M_n).

because every element has an inverse in a group. This means that if the prover claims that the result is some group element β, we can always find a group element α such that $\beta = y\alpha$. Soundess of PPoE only depends on the order of this bad element α. If its order only has small prime divisors with small exponents, the probability that repeated exponentiation of this element with a random exponent decreases its order to one, and thus the verifier's check for $T = 1$ passes, is not negligible.

The following lemma bounds the probability that the order of the bad element "drops" by a factor p^ℓ in one round of PPoE. It will be the main tool in proving soundness of our non-primality certificate later on.

Lemma 7. *Let $(x, y\alpha, T, \mathbb{G})$ be an α-false statement for some $\alpha \in \mathbb{G}$, $\mu \in \mathbb{G}$ an arbitrary group element, p^e any prime power that divides the order of α and let $r \leftarrow \{0, 1, \ldots, 2^\lambda - 1\}$ be sampled uniformly at random. Assume that the statement $(x^r\mu, \mu^r y\alpha, T/2, \mathbb{G})$ is β-false for some $\beta \in \mathbb{G}$. For any $\ell \leq e$, the probability that $p^{e-\ell+1}$ does not divide the order of β is at most $1/p^\ell$.*

Proof. If the statement $(x^r\mu, \mu^r y\alpha, T/2)$ is β-false, we have $\mu = \gamma x^{2^{T/2}}$ such that $\beta = \alpha\gamma^{r-2^{T/2}}$. We want to bound the following probability

$$\begin{aligned}\Pr_r[\beta^{p^{e-\ell}s} = \alpha^{p^{e-\ell}s}\gamma^{(r-2^{T/2})p^{e-\ell}s} = 1] &= \Pr_r[\gamma^{(r-2^{T/2})p^{e-\ell}s} = \alpha^{-p^{e-\ell}s}] \\ &\leq \frac{1}{\operatorname{ord}(\gamma^{p^{e-\ell}s})} + \frac{1}{2^\lambda} \\ &= \frac{\gcd(d, p^{e-\ell}s)}{d} + \frac{1}{2^\lambda}, \end{aligned} \tag{6.4}$$

where d denotes the order of γ and s is any positive integer not divisible by p. The inequality follows from the fact that the size of the randomness space is 2^λ. Now assume that the above event holds. Then we have $\gamma^{p^{e-\ell}sm} = \alpha^{-p^{e-\ell}s}$ for some integer m, hence

$$\operatorname{ord}(\alpha^{-p^{e-\ell}s}) = \operatorname{ord}(\gamma^{p^{e-\ell}sm}) = \frac{d}{\gcd(d, p^{e-\ell}sm)}$$

and equivalently

$$d = \operatorname{ord}(\alpha^{-p^{e-\ell}s}) \gcd(d, p^{e-\ell}sm) \geq p^\ell \gcd(d, p^{e-\ell}sm).$$

Plugging into (6.4) we get

$$\Pr[\alpha^{p^{e-\ell}s}\gamma^{(r-2^{T/2})p^{e-\ell}s} = 1] \leq \frac{\gcd(d, p^{e-\ell}s)}{p^\ell \gcd(d, p^{e-\ell}sm)} + \frac{1}{2^\lambda} \leq \frac{1}{p^\ell} + \frac{1}{2^\lambda}.$$

□

Remark 7. A lemma of flavour similar to Lemma 7 was proven in [HHK+22, Lemma 1] for a parallel-repeated variant of PPoE. However, there are major differences between these: (i) the new statements in the protocol in [HHK+22] (and also [BHR+21]) are obtained in a slightly different way, using multiple random coins and (ii) [HHK+22, Lemma 1] was only proven for restricted choices of numbers p and e. Hence, Lemma 7 does not follow from [HHK+22, Lemma 1].

Instance: $(x, y, T, \mathbb{G})$, where $x, y \in \mathbb{G}$, $T \in \mathbb{N}$ is even and $x^{2^T} = y$ in $\mathbb{G}$

Input to $\mathcal{P}^*$: $\alpha \in \mathbb{G}$

Parameters: statistical security parameter λ

Statement: $x^{2^T} = y\alpha$ in $\mathbb{G}$

Protocol:

1. For $T = 1$:
   - If $x^2 = y$, $\mathcal{V}$ outputs `accept`.
   - Else, $\mathcal{V}$ outputs `reject`.
2. For $T > 1$:
   a) $\mathcal{P}^*$ sends $v = \alpha^{-1} x^{2^{T/2}}$ to $\mathcal{V}$.
   b) If $v \notin \mathbb{G}$, $\mathcal{V}$ outputs `reject`. Otherwise, $\mathcal{V}$ samples $r \leftarrow \{0, 1, \ldots, 2^\lambda - 1\}$ uniformly at random and sends it to $\mathcal{P}^*$.
   c) $\mathcal{P}^*$ and $\mathcal{V}$ compute $x' := x^r v$ and $y' := v^r y$ in $\mathbb{G}$.
   d) If $T/2$ is even, $\mathcal{P}^*$ and $\mathcal{V}$ run the protocol on instance $(x', y', T/2, \mathbb{G})$ with input $\alpha^{2^{T/2}-r-1}$ to $\mathcal{P}^*$. If $T/2$ is odd, $\mathcal{P}^*$ and $\mathcal{V}$ run the protocol on instance $(x', y'^2, (T+1)/2, \mathbb{G})$ with input $\alpha^{2(2^{T/2}-r-1)}$ to $\mathcal{P}^*$.

Figure 6.2: An attack with success probability at least $1 - (1 - 1/\operatorname{ord}(\alpha))^{\log T}$.

Corollary 6. *Let $(x, y\alpha, T, \mathbb{G})$ be an α-false statement for some $\alpha \in \mathbb{G}$ and 2^e any power of 2 that divides the order of α. The probability that $2^{e-\ell}$ does not divide the order of the bad element after one round of `PPoE` is at most $1/2^\ell$.*

Proof. By Lemma 7 we know that the probability that $2^{e-\ell+1}$ does not divide the order of the bad element of the instance $(x', y', T/2, \mathbb{G})$ is at most $1/2^\ell$. Now if T is odd, the new instance of the protocol is $(x', y'^2, (T+1)/2, \mathbb{G})$, so the bad element is squared once. This reduces its order by a factor of 2, which yields the claim. □

6.3 Attacking Pietrzak's Protocol in Proth Number Groups

In this section we show how a malicious prover $\mathcal{P}^*$ can falsely convince the verifier $\mathcal{V}$ that a Proth *prime* is composite when using Pietrzak's PoE. This attack was first described in [BBF24]. Let $N = k2^n + 1$ be prime and x be any quadratic non-residue modulo N. Since N is prime, it holds that $x^{k2^{n-1}} = -1 \mod N$. The easiest way for $\mathcal{P}^*$ to cheat is claiming that the result of this exponentiation is 1 instead of -1 and then multiplying the honest messages by -1 until the recombination step (Step 2d of `PPoE`) yields a correct instance. The probability that $\mathcal{V}$ accepts this false "proof" of non-primality is $1 - 1/2^{\log(n-1)} = 1 - (n-1)^{-1}$. To see this, consider the first round of the protocol. $\mathcal{P}^*$ multiplies the correct midpoint $v = (x^k)^{2^{(n-1)/2}}$ by -1 and sends the message $-v$ to $\mathcal{V}$. $\mathcal{V}$ samples a random coin r and they both compute

$x' = -x^{kr}v$ and $y' = (-v)^r$ to create the new statement $x'^{2^{(n-1)/2}} = y'$. Plugging in the values for x', y' and v, we see that the new statement is correct whenever r is an odd integer:

$$\begin{aligned} & x'^{2^{(n-1)/2}} = y' \\ \Leftrightarrow\ & (-x^{kr}v)^{2^{(n-1)/2}} = (-v)^r \\ \Leftrightarrow\ & v^{2^{(n-1)/2}} = (-1)^r \\ \Leftrightarrow\ & (x^k)^{2^{n-1}} = (-1)^r. \end{aligned}$$

If r is even, the statement remains false and $\mathcal{P}^*$ does the same in the next round. $\mathcal{V}$ only outputs `reject` if all of the random coins are even which happens with probability $1/2^{\log(n-1)} = (n-1)^{-1}$ since $\log(n-1)$ is the number of rounds. Otherwise $\mathcal{V}$ outputs `accept` on a false statement. A generalization of this attack is shown in Figure 6.2. Instead of multiplying the correct statement by -1, $\mathcal{P}^*$ multiplies the correct statement by an arbitrary group element α and adapts its messages accordingly. The success probability can be lower bounded by $1-(1-1/\operatorname{ord}(\alpha))^{\log(n-1)}$ which is the probability that in at least one round the bad element is raised to a multiple of the order of α. If $\operatorname{ord}(\alpha)$ is not a prime number, this bound is not tight since the order of the bad element can decrease during the execution of the rounds, making the success probability even higher. In the case where N is prime, the prover knows the group order $N-1 = k2^n$ and its factorization and can therefore construct elements of sufficiently low order.

6.4 Certifying Non-Primality of Proth Primes

In this section, we present the interactive protocol for verifying that a Proth number $N = k2^n + 1$ is *not* prime, i.e., that $x^{k2^{n-1}} = -\mu \mod N$ for some $\mu \neq 1$ and x a small prime number that is a quadratic non-residue modulo N. This means that from now on all group operations will be performed in the group $\mathbb{Z}_N^*$.

The protocol presented in Figure 6.3 consists of four steps in which $\mathcal{V}$ performs different checks on the order of the element μ and then chooses the best method for verification accordingly. An overview can be found in Figure 6.1.

In the first step, $\mathcal{V}$ checks if x has Jacobi symbol -1 modulo N since the primality test is only conclusive if x is a quadratic non-residue. To this end, $\mathcal{V}$ first computes $a := N \mod x$ and, if $a \neq 0$, checks if the Jacobi symbol $(\frac{x}{N}) = (\frac{a}{x})$ is -1. If $a = 0$ we know that x is a divisor of N and hence N is composite, so $\mathcal{V}$ can already accept in Step 1. If the Jacobi symbol is 1, it is unclear if x is a quadratic residue mod N so $\mathcal{V}$ rejects the proof. If the Jacobi symbol is -1, the protocol moves on to the next step.

In the second step, $\mathcal{V}$ checks if the element μ has small *odd* order dividing $k2^n$, i.e., order dividing k, by computing $\tilde{\mu} := \mu^k \mod N$. If $\tilde{\mu} \neq 1$, the order of μ does not divide k and $\mathcal{V}$ goes on to the next step. If $\tilde{\mu} = 1$, $\mathcal{V}$ can easily find the order d of μ by factoring the (small) integer k. Then $\mathcal{V}$ can verify the statement without any message from $\mathcal{P}$. In fact, $\mathcal{V}$ only verifies the statement $x^{k2^n} = \mu^2$ by computing the 2^n-th root of μ^2. Unfortunately we can not compute the 2^{n-1}-th root of $-\mu$ because $-\mu$ has order $2d$ and the inverse of 2^{n-1} modulo $2d$ does not exist. This additional squaring step eliminates potential "bad" elements of order 2, so this check only proves that $x^{k2^{n-1}} = -\mu \cdot \alpha \mod N$, for some element α of order 2 or $\alpha = 1$. Luckily, this is enough information for $\mathcal{V}$ since we only want to rule out the possibility that the result of the exponentiation is -1 and $\mu^{-1} \neq \alpha$ since μ has odd order.

If $\mathcal{V}$ gets to the third step, we know that the order of μ does not divide k. To make sure that it does not divide k times a small power of 2 either, $\mathcal{V}$ checks if $\mu^{k2^{2^{\lambda \log n}}} \neq 1 \mod N$. If this holds, we know that $\mathcal{V}$ can accept a PPoE for the statement $x^{k2^{2^{n-1}}} = -\mu \mod N$ because a malicious prover will only be successful in convincing $\mathcal{V}$ with negligible probability. If $\mu^{k2^{2^{\lambda \log n}}} = 1$, such a PoE is not sound, so $\mathcal{V}$ goes on to the next step.

If $\mathcal{V}$ gets to the last step, we know that the order of μ is too small to soundly accept a PPoE. However, we now know that the order of μ is even, so we can use the following trick: Instead of sending a PPoE for the statement $x^{k2^{2^{n-1}}} = -\mu \mod N$, $\mathcal{P}$ sends a PPoE for the statement $x^{k2^{2^{n-1-\lambda \log n}}} = y \mod N$ for some element $y \in \mathbb{Z}_N^*$. Then $\mathcal{V}$ checks if $y^{2^{2^{\lambda \log n}}} = -\mu$. If this holds and the PoE is correct, $\mathcal{V}$ outputs accept. Else, $\mathcal{V}$ outputs reject.

Remark 8. The complexity of Steps 3 and 4 of our protocol could be slightly improved with the following changes:

- Instead of $\mathcal{V}$ computing the exponentiation $\tilde{\mu}_1^{2^{2^{\lambda \log n}}}$ in Step 3, $\mathcal{P}$ could send a proof for the statement $\tilde{\mu}_1^{2^{2^{\lambda \log n}}} \neq 1$. This can be done in a sound manner since again $\mathcal{V}$ only wants to rule out *one* result, so $\mathcal{P}$ and $\mathcal{V}$ can execute Steps 2-4 recursively. This reduces the work for $\mathcal{V}$ but increases the work for $\mathcal{P}$. However, the PoEs can be batched together similarly to the batching protocol in [HHK+22] so the proof size only grows by one group element.
- If $\mathcal{V}$ and $\mathcal{P}$ find out in Step 3 that $\tilde{\mu}_1^{2^{2^{\lambda \log n}}} = 1$, they also know the smallest integer i such that $\tilde{\mu}_1^{2^{2^{\lambda \log n - i}}} \neq 1$. This means that, in Step 4, $\mathcal{P}$ can send a PoE for the statement
$$(x^k)^{2^{2^{n-1-\lambda \log n+i}}} = y \mod N,$$
and $\mathcal{V}$ only needs to check if $y^{2^{2^{\lambda \log n - i}}} = -\mu \mod N$. This reduces the work for $\mathcal{V}$ by i multiplications.

For simplicity of the analysis and because the improvements are minor, we omit these changes and analyze the protocol as it is stated in Figure 6.3.

6.4.1 Completeness

In this section, we show that $\mathcal{V}$ always outputs accept if $\mathcal{P}$ is honest.

Theorem 15. *The protocol in Figure 6.3 has perfect completeness.*

Proof. We show that if $\mathcal{P}$ is honest, $\mathcal{V}$ does not output reject in any step and outputs accept in one of the steps.

Step 1. Assume that x does not divide N since otherwise $\mathcal{V}$ accepts in the first step and completeness holds trivially. If $\mathcal{P}$ is honest, x has Jacobi symbol $(\frac{x}{N}) = -1$. Furthermore, since x is prime and does not divide N, we have
$$\left(\frac{x}{N}\right) = (-1)^{(x-1)k2^n/4} \left(\frac{N}{x}\right) = \left(\frac{N}{x}\right) = \left(\frac{N \mod x}{x}\right),$$
where the first equality follows from the law of quadratic reciprocity. Hence, $\mathcal{V}$ does not reject in this step and goes on to the next one.

Instance: (n, k, x, μ), where $n \in \mathbb{N}$, $0 < k < 2^{n-1}$ an odd integer, $\mu \in \mathbb{Z}_N^*$ with $\mu \neq 1$ and $x \in \mathbb{Z}_N^*$ a small prime number with Jacobi symbol -1 modulo $N := k2^n + 1$

Parameters: statistical security parameter λ

Statement: $x^{k2^{n-1}} = -\mu \mod N$

Protocol:

1. $\mathcal{V}$ computes $a := N \mod x$ and if $a \neq 0$ the Jacobi symbol $(\frac{a}{x})$.
   - If $a = 0$, output `accept`.
   - If $a \neq 0$ and $(\frac{a}{x}) = -1$, go to Step 2.
   - Else, output `reject`.
2. $\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{\mu}_1 := \mu^k \mod N$
   - If $\tilde{\mu}_1 \neq 1$, go to Step 3.
   - Else, $\mathcal{V}$ computes $d := \mathrm{ord}(\mu)$ and $a := 2^{-n} \mod d$. If $x^k = \mu^{2a} \mod N$ output `accept`. Else, output `reject`.
3. $\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{\mu}_2 := \tilde{\mu}_1^{2^{\lambda \log n}} \mod N$.
   - If $\tilde{\mu}_2 = 1$, go to Step 4.
   - Else, $\mathcal{P}$ sends $\texttt{PPoE}(x^k, -\mu, n-1, N)$. If the `PPoE` verifier accepts, output `accept`. Else, output `reject`.
4. a) $\mathcal{P}$ sends a group element y and a `PPoE` $(x^k, y, n - 1 - \lambda \log n, N)$ for some $y \in \mathbb{Z}_N^*$. If the `PPoE` verifier rejects, output `reject`. Else, go to Step 4b.
   b) $\mathcal{V}$ computes $\tilde{y} := y^{2^{\lambda \log n}} \mod N$. If $\tilde{y} = -\mu$ output `accept`. Else, output `reject`.

Figure 6.3: The non-primality certificate.

Step 2. If $\tilde{\mu}_1 \neq 1$, $\mathcal{V}$ does not output anything in this step and goes on to the next one. Assume $\tilde{\mu}_1 = 1$ and let $a := 2^{-n} \mod d$, where d is the order of μ. Then we have

$$\mu^{2a} = (\mu^2)^{2^{-n}} = ((x^k)^{2^n})^{2^{-n}} = x^k \mod N$$

so $\mathcal{V}$ accepts if $\mathcal{P}$ is honest.

Step 3. If $\tilde{\mu_2} = 1$, $\mathcal{V}$ does not output anything in this step and goes on to the next one. If $\tilde{\mu_2} \neq 1$, completeness follows immediately from the completeness property of `PPoE`.

Step 4. If $\mathcal{P}$ is honest, the verifier does not reject in Step 4a by the completeness property of `PPoE`. In Step 4b, the verifier checks if

$$y^{2^{\lambda \log n}} = (x^{k2^{n-1-\lambda \log n}})^{2^{\lambda \log n}} = -\mu \mod N,$$

which holds if $\mathcal{P}$ is honest.

□

6.4.2 Soundness

For our purposes, it is sufficient to consider a relaxed definition of soundness. We only want to rule out the event that a malicious prover $\tilde{\mathcal{P}}$ can convince $\mathcal{V}$ that a Proth number is not prime even though it is. This means we do not need to care about a cheating prover that convinces $\mathcal{V}$ of a wrong result of the exponentiation $x^{k2^{n-1}} \mod N$ as long as the correct result is not -1.

Definition 21. We call a non-primality certificate *sound* if the probability that $\mathcal{V}$ outputs `accept` on a statement (n, k, x, μ) for some $\mu \neq 1$ but $x^{k2^{n-1}} = -1 \mod N$ is negligible. We call that probability the *soundness error*.

Theorem 16. *The protocol in Figure 6.3 has soundness error at most* $2^{-\lambda+2} \log n$.

Proof. We bound the probability that $\mathcal{V}$ falsely accepts an incorrect statement in each step individually.

Step 1. If $\mathcal{V}$ accepts in Step 1, x is a divisor of N so N must be composite. Assume that this does not hold and x has Jacobi symbol 1 modulo N, i.e., $(\frac{x}{N}) = 1$. Then,

$$\left(\frac{a}{x}\right) = \left(\frac{N}{x}\right) = (-1)^{(x-1)k2^n/4} \left(\frac{x}{N}\right) = 1,$$

so $\mathcal{V}$ rejects in Step 1.

Step 2. Recall that we consider a relaxed definition of soundness (see Definition 21). This means that we only need $\mathcal{V}$ to reject, when the correct result of the exponentiation is -1. We show that if this is the case and $\tilde{\mu}_1 = 1$, $\mathcal{V}$ always rejects in Step 2(b). Assume that N is prime. If $\mathcal{V}$ gets to Step 2(b), we know that d is a divisor of k and hence odd. This means that μ^2 has order d. $\mathcal{V}$ computes $a := 2^{-n} \mod d$ and checks if $x^k = (-\mu)^{2a} = \mu^{2a} \mod d$.

1. Since N is prime it holds that $(x^k)^{2^{n-1}} = -1 \mod N$ so the order of x is $2^n k$. This means that the order of x^k is 2^n.
2. On the other hand, we know that the order of μ^2 is d so the order of μ^{2a} is a divisor of d and hence odd.

1. and 2. together yield that $x^k \neq \mu^{2a} \mod N$ so the verifier rejects.

Step 3. If $\mathcal{V}$ gets to Step 3 and $\tilde{\mu}_2 \neq 1$, we know that the order of the bad element μ is divisible by $2^{\lambda \log T}$. This means that a malicious prover convinces $\mathcal{V}$ to falsely accept if the execution of the PoE reduces the order of the bad element on average by 2^λ per round. In particular, there must be at least one round where the order drops by at least 2^λ. By Corollary 6, this happens with probability at most $2^{-\lambda+2}$ for a fixed round. Applying a union bound, we conclude that, in this case, $\mathcal{V}$ accepts with probability at most $2^{-\lambda+2} \log n$.[10]

[10] PrimeGrid has already implemented a check $\mu^{k \cdot 2^{64}} = 1$? [Atn22]. Our analysis shows that an exponent of 64 is not sufficient for cryptographic soundness as this only gives $64/\log(n)$ bits of security "per round"; once we apply the Fiat-Shamir methodology to make the proof non-interactive, each round can be attacked individually.

Step 4. If $\mathcal{V}$ gets to Step 4, a malicious prover needs to cheat in Step 4 (a) since otherwise the check in Step 4 (b) will not go through. This means that the prover needs to multiply the claim in Step 4 (a) by a bad element α. What can we say about the order of α? We know that it needs to pass the following check:

$$(y\alpha)^{2^{\lambda \log T}} = -\mu,$$

where $y^{2^{\lambda \log T}} = -1$. This means that α is of the following form:

$$\alpha^{2^{\lambda \log T}} = \mu.$$

It is well known that

$$\operatorname{ord}(\alpha^{2^i}) = \frac{\operatorname{ord}(\alpha)}{\gcd(2^i, \operatorname{ord}(\alpha))} = \operatorname{ord}(\mu).$$

(A proof can be found in any standard textbook on group theory, e.g., [DF03, Proposition 5]). Now the order of μ is even so we know that $\operatorname{ord}(\alpha) = 2^i \operatorname{ord}(\mu)$ for $i = \lambda \log T$. In particular, we have that $2^{\lambda \log T}$ is a divisor of the order of α. We can apply Corollary 6 and a union bound by the same argument as above and conclude that $\mathcal{V}$ accepts with probability at most $2^{-\lambda+2} \log n$ in this case.

All the cases together show that $\mathcal{V}$ outputs `accept` with probability at most $2^{-\lambda+2} \log n$ whenever N is prime. □

Corollary 7. *The Fiat-Shamir transform of the protocol in Figure 6.3 yields a statistically sound non-interactive protocol in the random oracle model: The probability that $\mathcal{P}$ finds a non-primality certificate for a prime number with up to Q random oracle queries is at most $Q2^{-\lambda+2}$.*

Proof. As we have seen in the proof of Theorem 16, a cheating prover $\tilde{\mathcal{P}}$ can convince $\mathcal{V}$ to accept a proof of non-primality of a prime number only if $\tilde{\mathcal{P}}$ manages to decrease the order of the bad element by at least 2^λ in one of the rounds of a PPoE. By Corollary 6, this happens with probability at most $2^{-\lambda+2}$, where the probability depends only on the random coins. Assume that $\tilde{\mathcal{P}}$ makes up to Q queries to the random oracle. By the union bound, the probability that $\tilde{\mathcal{P}}$ finds a query that triggers the above event is at most $Q2^{-\lambda+2}$. □

6.4.3 Efficiency

In this section, we analyze the complexity of the Fiat-Shamir transform of the protocol presented in Figure 6.3. Note that this complexity depends on the step in which the protocol returns the output. We summarize the results of this section in Table 6.1.

Prover's complexity.

We compute the number of multiplications the prover has to perform additionally to finding a quadratic non-residue modulo N and computing the initial exponentiation.

Step 1. If the protocol returns the output in Step 1, $\mathcal{P}$ does not perform any additional computations.

Output in	Prover's complexity	Verifier's complexity	Proof size
Step 1	0	$\log n$	0
Step 2	$1.5 \log k$	$2.5 \log k + 2 \log n$	0
Step 3	$1.5 \log k + \lambda \log n + 2\sqrt{n}$	$1.5 \log k + (4\lambda + 1) \log n$	$\log n$
Step 4	$1.5 \log k + \lambda \log n + 2\sqrt{n}$	$1.5 \log k + (5\lambda + 1) \log n$	$\log n + 1$

Table 6.1: Complexity of the protocol in Figure 6.3 depending on the step in which it outputs the result. Prover's and Verifier's complexity are measured in the number of multiplications and proof-size in the number of group elements. We denote by λ the statistical security parameter.

Step 2. $\mathcal{P}$ checks if $\mu^k = 1$ via "square and multiply", which is approximately $1.5 \log k$ multiplications. If this holds, $\mathcal{P}$ does not perform any other computations.

Step 3. If the protocol runs until Step 3, $\mathcal{P}$ has checked if $\mu^k = 1$, which did not hold and now checks if $(\mu^k)^{2^{\lambda \log n}}$, which is $\lambda \log n$ additional multiplications. If this holds, $\mathcal{P}$ computes the proof of $\texttt{PPoE}(x^k, -\mu, n-1, N)$ which, by Proposition 2, can be done with $2\sqrt{n}$ multiplications and storage of $\sqrt{n}$ group elements.

Step 4. If the protocol runs until Step 4, $\mathcal{P}$ has checked if $\mu^k = 1$ and $(\mu^k)^{2^{\lambda \log n}}$, which did not hold. Now $\mathcal{P}$ computes the proof of $\texttt{PPoE}\ (x^k, y, n - 1 - \lambda \log n, N)$, which, by Proposition 2, can be done with $2\sqrt{n - \lambda \log n}$ multiplications and storage of $\sqrt{n - \lambda \log n}$ group elements.

Verifier's complexity.

Step 1. Computing $a := k2^n + 1 \mod x$ takes approximately $\log n$ multiplications. Computing the Jacobi symbol $(\frac{a}{x})$ takes approximately $\log^2 x$ multiplications. Since x is a very small prime number in practice, we will ignore the $\log^2 x$ multiplications from now on.

Step 2. $\mathcal{V}$ checks if $\mu^k = 1$ via "square and multiply", which is approximately $1.5 \log k$ multiplications. If this holds, $\mathcal{V}$ computes $2^{-n} \mod d$, where d is the order of μ. This is another $\log n + \log k$ multiplications.

Step 3. If the protocol runs until Step 3, $\mathcal{V}$ has checked if $\mu^k = 1$, which did not hold and now checks if $(\mu^k)^{2^{\lambda \log n}}$, which is $\lambda \log n$ additional multiplications. If this holds, $\mathcal{V}$ verifies the proof of $\texttt{PPoE}(x^k, -\mu, n-1, N)$ which is $3\lambda \log n$ multiplications by Proposition 2.

Step 4. If the protocol runs until Step 4, $\mathcal{V}$ has checked if $\mu^k = 1$ and $(\mu^k)^{2^{\lambda \log n}}$, which did not hold. Now $\mathcal{V}$ verifies the proof of $\texttt{PPoE}\ (x^k, y, n - 1 - \lambda \log n, N)$, which is $3\lambda \log(n - \lambda \log n)$ multiplications (by Proposition 2) and then performs an exponentiation with exponent $2^{\lambda \log n}$, which is another $\lambda \log n$ multiplications.

Proof size.

Step 1. If $\mathcal{V}$ already accepts or rejects in Step 1, there is no proof needed.

Step 2. If $\mu^k = 1$, $\mathcal{V}$ can check the result themselves so there is no proof needed.

Step 3. If $\mathcal{P}$ sends a proof in this step, the proof size is equal to the size of the proof of $\texttt{PPoE}(x^k, -\mu, n-1, N)$, which is $\log(n-1)$ by Proposition 2.

Step 4. If $\mathcal{P}$ sends a proof in this step, it consists of a group element y and the proof of $\texttt{PPoE}\ (x^k, y, n-1-\lambda\log n, N)$, which is $\log(n-1-\lambda\log n)$ by Proposition 2.

Example.

We give a numerical example of the complexity of the protocol when it outputs the result in Step 4 (the most expensive case) using the largest Proth prime known to date: $10223 \cdot 2^{31172165} + 1$ [Pri16]. For $k = 10223$ and $n = 31172165$ we have $\lceil \log k \rceil = 14$ and $\lceil \log n \rceil = 25$. If we choose the security parameter as $\lambda = 80$, we get that the prover stores $\lceil \sqrt{31172165} \rceil = 5584$ group elements, performs 13188 multiplications, the verifier performs 10046 multiplications and the proof size is 26 elements of size 31172179, i.e., around 102 MB. Note that recomputing the result of the primality test would take $n = 31172165$ multiplications in the same group, so our protocol reduces the number of multiplications by a multiplicative factor of $\lfloor 31172165/(13188 + 10046) \rfloor = 1341$.

Our protocol also achieves significant savings compared to [BHR+21]: To compute the PoE of [BHR+21], the prover needs to perform $2\lambda\sqrt{n} = 893312$ multiplications and the verifier does $2\lambda^2 \log n + 2\lambda = 318800$ multiplications to verify the proof consisting of $\lambda \log n = 2080$ group elements (i.e. 8160 MB). This means that our protocol reduces the number of multiplications of [BHR+21] by a factor of 52 and the proof size by a factor of 80.

6.4.4 Comparison with Pietrzak's PoE

We saw in Section 6.4.3 that the complexity of the protocol is the highest, when it outputs the result in Step 4. Even in this case the additional cost compared to the naive implementation of $\texttt{PPoE}$ is moderate: Instead of performing $2\sqrt{n}$ multiplications, $\mathcal{P}$ needs $1.5\log k + \lambda \log n + 2\sqrt{n}$ multiplications to compute the proof. Using the numbers from the example in Section 6.4.3 this is 2021 extra multiplications on top of the 11168 multiplications that are performed in the naive implementation. Instead of performing $3\lambda \log n$ multiplications, $\mathcal{V}$ needs $1.5\log k + (5\lambda + 1)\log n$ multiplications to verify the result. This is 4046 additional multiplications to the 6000 multiplications of the naive implementation in our example. The proof size grows by one group element from $\log n$ to $\log n + 1$. In our example, this corresponds to a proof size of 102 MB instead of 98 MB. If the protocol outputs the result in Step 1 or Step 2 it is even more efficient than $\texttt{PPoE}$. Recall that the implementation of $\texttt{PPoE}$ in groups of known order does not have any soundness guarantees and for the groups that we are using, there are known attacks that break soundness. In contrast, we showed in Section 6.4.2 that our protocol is statistically sound in these groups. We conclude that our protocol yields a major soundness improvement at moderate additional costs.

6.5 Conclusion

In this chapter, we presented an efficient protocol that gives a certificate of non-primality for Proth numbers. While we believe that a certificate of non-primality is more useful than a certificate of primality in the context of search for giant primes, constructing the latter is certainly an intriguing problem. Though, our techniques are not directly applicable to prove *primality* of Proth numbers because our protocol only rules out that the correct result is

one specific number (namely -1). Conversely, when proving primality one has to rule out *all results except for one* (again -1). Constructing a cryptographic certificate of primality therefore remains an open problem.

Another open problem is to demonstrate the applicability of PoEs towards certifying (non-) primality of other types of numbers such as, for example, Mersenne numbers, which are of the form $N = 2^n - 1$. The primality of Mersenne numbers is tested via the Lucas Lehmer test amounting to computation of long modular recursive sequences. Equivalently, the test can be performed via exponentiation in an extension ring: It can be shown that N is prime if and only if $(2 + \sqrt{3})^{(N+1)/2} = -1$ in $\mathbb{Z}_N[\sqrt{3}]$. Thus, one could hope to employ PoEs also in the context of Mersenne number. However, there are some major differences to the case of Proth numbers. In particular, the order of the corresponding multiplicative group is not necessarily efficiently computable even when the candidate is a prime, which is one of the issues preventing the use of our protocol.

CHAPTER 7

Conclusion

In this thesis we studied verifiable delay functions based on iterated squaring and their foundational building blocks proofs of exponentiation. We considered both theoretical and practical aspects of VDFs and PoEs.

Chapter 3 considered statistically sound PoEs that are secure even when the group order is known, enabling the use of transparent setups without sacrificing security. Our construction reduces the proof size of statistically sound PoEs for structured exponents, making statistically sound PoEs more practical for deployment in blockchain systems and cryptographic proofs such as SNARKs.

Next, in Chapter 4, we enhanced the scalability of PoEs through efficient batching protocols. These allow the aggregation of multiple proofs into a single one, which substantially reduces communication and verification costs. Our constructions demonstrate significant performance gains over existing batch PoEs and introduce the first batch PoE that significantly reduces both proof size and verifier complexity.

When exploring advanced applications in Chapter 5, we constructed zero-knowledge and watermarkable VDFs, which are needed for the construction of short-lived proofs and signatures. These primitives offer authentication that expires after a predefined amount of time.

Finally, Chapter 6 demonstrates the utility of PoEs in primality testing, particularly for Proth numbers. By adapting Pietrzak's PoE protocol, we offer cryptographically sound certificates of non-primality, which significantly reduce the workload in the ongoing search for giant prime numbers.

The results of this thesis advance the state of the art of verifiable delay functions, offering provably secure, efficient, and versatile cryptographic constructions. Our contributions open pathways for further research into post-quantum secure VDFs, optimized implementations, and broader integrations in cryptographic protocols and blockchain ecosystems.

Bibliography

[ABC22] Arasu Arun, Joseph Bonneau, and Jeremy Clark. Short-lived zero-knowledge proofs and signatures. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 487–516, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland. 7, 8, 65, 66, 68, 73, 77, 80, 82, 84

[ACK21] Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 549–579, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. 58, 59, 61

[AHPP23] Benedikt Auerbach, Charlotte Hoffmann, and Guillermo Pascual-Perez. Generic-group lower bounds via reductions between geometric-search problems: With and without preprocessing. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023: 21st Theory of Cryptography Conference, Part III*, volume 14371 of *Lecture Notes in Computer Science*, pages 301–330, Taipei, Taiwan, November 29 – December 2, 2023. Springer, Cham, Switzerland. x

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. 90

[AM93] A. O. L. Atkin and F. Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61(203):29–68, 1993. 90

[Atn22] Pavel Atnashev. Personal communication, 2022. February 2022. 87, 91, 97

[AVD20] Vidal Attias, Luigi Vigneri, and Vassil S. Dimitrov. Implementation study of two verifiable delay functions. In Emmanuelle Anceaume, Christophe Bisière, Matthieu Bouvard, Quentin Bramas, and Catherine Casamatta, editors, *2nd International Conference on Blockchain Economics, Security and Protocols, Tokenomics 2020, October 26-27, 2020, Toulouse, France*, volume 82 of *OASIcs*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 49

[AVD22] Vidal Attias, Luigi Vigneri, and Vassil S. Dimitrov. Efficient verification of the wesolowski verifiable delay function for distributed environments. *IACR Cryptol. ePrint Arch.*, page 520, 2022. 44, 49

[AVD23] Vidal Attias, Luigi Vigneri, and Vassil S. Dimitrov. Rethinking modular multi-exponentiation in real-world applications. *J. Cryptogr. Eng.*, 13(1):57–70, 2023. 47

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. 2, 9, 12, 73

[BBF24] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions using proof of exponentiation. *IACR Communications in Cryptology (CiC)*, 1(1):7, 2024. 4, 14, 15, 21, 25, 44, 87, 91, 93

[BCH+22] Nir Bitansky, Arka Rai Choudhuri, Justin Holmgren, Chethan Kamath, Alex Lombardi, Omer Paneth, and Ron D. Rothblum. PPAD is as hard as LWE and iterated squaring. TCC 2022, to appear, 2022. 9

[BDH+25] Juraj Belohorec, Pavel Dvořák, Charlotte Hoffmann, Pavel Hubáček, Kristýna Mašková, and Martin Pastyřík. On extractability of the KZG family of polynomial commitment schemes. Cryptology ePrint Archive, Report 2025/514, 2025. xi, 58

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland. 37, 39, 40

[BGR98] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250, Espoo, Finland, May 31 – June 4, 1998. Springer Berlin Heidelberg, Germany. xv, 7, 10, 15, 45, 49, 52, 53

[BHR+21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 123–152, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. xiv, xv, 3, 4, 5, 6, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 43, 46, 49, 88, 92, 100

[BKSW20] Karim Belabas, Thorsten Kleinjung, Antonio Sanso, and Benjamin Wesolowski. A note on the low order assumption in class group of an imaginary quadratic number fields. Cryptology ePrint Archive, Report 2020/1310, 2020. 5, 10

[BKT22] B. Borsos, A. Kovács, and N. Tihanyi. Tight upper and lower bounds for the reciprocal sum of proth primes. *Ramanujan J*, 59:181–198, 2022. 87

[BKZZ16] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Indistinguishable proofs of work or knowledge. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 902–933, Hanoi, Vietnam, December 4–8, 2016. Springer Berlin Heidelberg, Germany. 68

[BLS75] John Brillhart, D. H. Lehmer, and J. L. Selfridge. New primality criteria and factorizations of $2^m \pm 1$. *Mathematics of Computation*, 29(130):620–647, 1975. 91

[BN00] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Santa Barbara, CA, USA, August 20–24, 2000. Springer Berlin Heidelberg, Germany. 5

[BNHR22] Andrej Bogdanov, Miguel Cueto Noval, Charlotte Hoffmann, and Alon Rosen. Public-key encryption from homogeneous CLWE. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 565–592, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland. x

[BP00] Colin Boyd and Chris Pavlovski. Attacking and repairing batch verification schemes. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 58–71, Kyoto, Japan, December 3–7, 2000. Springer Berlin Heidelberg, Germany. 25

[BS07] D.J. Bernstein and J.P. Sorenson. Modular exponentiation via the explicit chinese remainder theorem. *Mathematics of Computation*, 76:443–454, 2007. 1, 14

[BW80] Robert Baillie and Samuel S. Wagstaff. Lucas pseudoprimes. *Mathematics of Computation*, 35(152):1391–1417, 1980. 90

[BW88] Johannes Buchmann and Hugh C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, June 1988. 3, 86

[BY93] Michael J. Beller and Yacov Yacobi. Batch Diffie-Hellman key agreement systems and their application to portable communications. In Rainer A. Rueppel, editor, *Advances in Cryptology – EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 208–220, Balatonfüred, Hungary, May 24–28, 1993. Springer Berlin Heidelberg, Germany. 10

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski, Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469, Santa Barbara, CA, USA, August 17–21, 1997. Springer Berlin Heidelberg, Germany. 67

[CCD+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. Multiparty generation of an RSA modulus. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 64–93, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. 5

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor,

Advances in Cryptology – CRYPTO'94, volume 839 of *Lecture Notes in Computer Science*, pages 174–187, Santa Barbara, CA, USA, August 21–25, 1994. Springer Berlin Heidelberg, Germany. 84

[CE12] Jeremy Clark and Aleksander Essex. CommitCoin: Carbon dating commitments with Bitcoin - (short paper). In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 390–398, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer Berlin Heidelberg, Germany. 2

[CHK+19] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1103–1114, Phoenix, AZ, USA, June 23–26, 2019. ACM Press. 9

[CHP12] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. *J. Cryptol.*, 25(4):723–747, 2012. 49

[CKKS17] Giovanni Di Crescenzo, Matluba Khodjaeva, Delaram Kahrobaei, and Vladimir Shpilrain. Computing multiple exponentiations in discrete log and RSA groups: From batch verification to batch delegation. In *2017 IEEE Conference on Communications and Network Security, CNS 2017, Las Vegas, NV, USA, October 9-11, 2017*, pages 531–539. IEEE, 2017. 49

[CLM23] Valerio Cini, Russell W. F. Lai, and Giulio Malavolta. Lattice-based succinct arguments from vanishing polynomials - (extended abstract). In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 72–105, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. 9

[Col18] Michael Colburn. Short-lived signatures. Master's thesis, Concordia University, 2018. 68

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Santa Barbara, CA, USA, August 16–20, 1993. Springer Berlin Heidelberg, Germany. 70

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 451–467, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland. 2

[CP19] Bram Cohen and Krzysztof Pietrzak. The Chia network blockchain. Technical report, 2019. https://www.chia.net/assets/ChiaGreenPaper.pdf, Accessed: 2022-07-29. 2, 4

[CPP17] 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, *Advances in Cryptology – EUROCRYPT 2017,*

Part II, volume 10211 of *Lecture Notes in Computer Science*, pages 321–350, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland. 68

[CSHT21] Jorge Chavez-Saab, Francisco Rodríguez Henríquez, and Mehdi Tibouchi. Verifiable isogeny walks: Towards an isogeny-based postquantum VDF. Cryptology ePrint Archive, Report 2021/1289, 2021. 9

[CSRHT22] Jorge Chavez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Verifiable isogeny walks: Towards an isogeny-based postquantum vdf. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography*, pages 441–460, Cham, 2022. Springer International Publishing. 9

[DF03] David S. Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley and Sons, 3rd edition, 2003. 27, 98

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 585–605, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany. 4

[DGMV20] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20: 12th International Conference on Security in Communication Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 65–84, Amalfi, Italy, September 14–16, 2020. Springer, Cham, Switzerland. 9

[DMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 248–277, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland. 9

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147, Santa Barbara, CA, USA, August 16–20, 1993. Springer Berlin Heidelberg, Germany. 1

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 125–154, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland. 9

[Erd32] Paul Erdős. Beweis eines satzes von Tschebyschef (on a proof of a theorem of Chebyshev, in german). *Acta Litt. Sci. Szeged*, 5:194–198, 01 1932. 30

[FGN15] Houda Ferradi, Rémi Géraud, and David Naccache. Slow motion zero knowledge identifying with colliding commitments. In *Revised Selected Papers of the 11th International Conference on Information Security and Cryptology - Volume 9589*, Inscrypt 2015, page 381–396, Berlin, Heidelberg, 2015. Springer-Verlag. 68

[Fia97] Amos Fiat. Batch RSA. *Journal of Cryptology*, 10(2):75–88, March 1997. 10, 49

[FKPS21] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 447–479, Raleigh, NC, USA, November 8–11, 2021. Springer, Cham, Switzerland. 5

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 331–361, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. 5

[FPS22] Cody Freitag, Rafael Pass, and Naomi Sirkin. Parallelizable delegation from LWE. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 623–652, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland. 9

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer Berlin Heidelberg, Germany. 4, 12, 15, 87

[FS00] Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, March 2000. 13

[GIM18] Great Internet Mersenne Prime Search GIMPS. GIMPS discovers largest known prime number: $2^{82,589,933} - 1$. https://www.mersenne.org/primes/press/M82589933.html, 2018. Accessed: 2022-05-18. 85

[GIM20] Great Internet Mersenne Prime Search GIMPS. Prime95 v30.3. https://www.mersenneforum.org/showthread.php?t=25823, 2020. Accessed: 2022-05-19. 86

[GK86] Shafi Goldwasser and Joe Kilian. Almost all primes can be quickly certified. In *18th Annual ACM Symposium on Theory of Computing*, pages 316–329, Berkeley, CA, USA, May 28–30, 1986. ACM Press. 90

[GQ90] Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231, Santa Barbara, CA, USA, August 21–25, 1990. Springer, New York, USA. 70

[GRY24] Ziyi Guan, Artur Riazanov, and Weiqiang Yuan. Breaking verifiable delay functions in the random oracle model. Cryptology ePrint Archive, Report 2024/766, 2024. 9

[Hen10] Ryan Henry. Pippenger's multiproduct and multiexponentiation algorithms. Technical report, University of Waterloo, 2010. 47

[HHI25] Charlotte Hoffmann, Pavel Hubáček, and Svetlana Ivanova. Practical batch proofs of exponentiation. *IACR Communications in Cryptology*, 2(3), 2025. x, 6, 7

[HHK+22] Charlotte Hoffmann, Pavel Hubácek, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Practical statistically-sound proofs of exponentiation in any group. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 370–399, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. x, 4, 5, 88, 90, 92, 95

[HHKK23] Charlotte Hoffmann, Pavel Hubácek, Chethan Kamath, and Tomás Krnák. (Verifiable) delay functions from lucas sequences. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023: 21st Theory of Cryptography Conference, Part IV*, volume 14372 of *Lecture Notes in Computer Science*, pages 336–362, Taipei, Taiwan, November 29 – December 2, 2023. Springer, Cham, Switzerland. x, 9

[HHKP23] Charlotte Hoffmann, Pavel Hubácek, Chethan Kamath, and Krzysztof Pietrzak. Certifying giant nonprimes. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 530–553, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland. x, 8, 9

[HK09] Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 637–653, Santa Barbara, CA, USA, August 16–20, 2009. Springer Berlin Heidelberg, Germany. 13

[Hof24] Charlotte Hoffmann. Traceable secret sharing based on the chinese remainder theorem. Cryptology ePrint Archive, Report 2024/811, 2024. xi

[HP25] Charlotte Hoffmann and Krzysztof Pietrzak. Watermarkable and zero-knowledge verifiable delay functions from any proof of exponentiation. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025: 28th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 15674 of *Lecture Notes in Computer Science*, pages 36–66, Røros, Norway, May 12–15, 2025. Springer, Cham, Switzerland. x, 7, 8, 48, 59

[HS23] Charlotte Hoffmann and Mark Simkin. Stronger lower bounds for leakage-resilient secret sharing. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *Progress in Cryptology - LATINCRYPT 2023: 8th International Conference on Cryptology and Information Security in Latin America*, volume 14168 of *Lecture Notes in Computer Science*, pages 215–228, Quito, Ecuador, October 3–6, 2023. Springer, Cham, Switzerland. x

[Inc23] Chia Network Inc. Chia network. http://chia.net, 2023. Accessed: 2023-12-19. 6

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 723–732, Victoria, BC, Canada, May 4–6, 1992. ACM Press. 88

[KLX20] Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part III*, volume 12552 of *Lecture Notes in Computer Science*, pages 390–413, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland. 5

[KMT22] Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. MinRoot: Candidate sequential function for ethereum VDF. Cryptology ePrint Archive, Report 2022/1626, 2022. 9

[KTY04] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 571–589, Interlaken, Switzerland, May 2–6, 2004. Springer Berlin Heidelberg, Germany. xv, 68, 69, 70

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194, Singapore, December 5–9, 2010. Springer Berlin Heidelberg, Germany. 37, 49

[Leh27] D. H. Lehmer. Tests for primality by the converse of Fermat's theorem. *Bulletin of the American Mathematical Society*, 33(3):327 – 340, 1927. 85, 91

[LM23] Russell W. F. Lai and Giulio Malavolta. Lattice-based timed cryptography. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 782–804, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. 9

[LMPR23] Gaëtan Leurent, Bart Mennink, Krzysztof Pietrzak, and Vincent Rijmen. Analysis of MinRoot: Public report (requested by ethereum foundation). Technical report, Ethereum Foundation, 2023. 9

[LSS20] Esteban Landerreche, Marc Stevens, and Christian Schaffner. Non-interactive cryptographic timestamping based on verifiable delay functions. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020: 24th International Conference on Financial Cryptography and Data Security*, volume 12059 of *Lecture Notes in Computer Science*, pages 541–558, Kota Kinabalu, Malaysia, February 10–14, 2020. Springer, Cham, Switzerland. 2

[Luc78] Edouard Lucas. Théorie des fonctions numériques simplement périodiques. *American Journal of Mathematics*, 1(4):289–321, 1878. 85, 91

[LV20] Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume

12172 of *Lecture Notes in Computer Science*, pages 632–651, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. 9

[LW15] Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Paper 2015/366, 2015. 2

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005. 48

[MBS82] John Mason, Leonard Burton, and Kaye Stacey. *Thinking Mathematically*. Pearson Education, Harlow, England, 1st edition, 1982. 48

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press. 88

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976. 8, 86, 90

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany. 9

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013: 4th Innovations in Theoretical Computer Science*, pages 373–388, Berkeley, CA, USA, January 9–12, 2013. Association for Computing Machinery. 2

[MN96] David M'Raïhi and David Naccache. Batch exponentiation: A fast DLP-based signature generation strategy. In Li Gong and Jacques Stearn, editors, *CCS '96, Proceedings of the 3rd ACM Conference on Computer and Communications Security, New Delhi, India, March 14-16, 1996*, pages 58–61. ACM, 1996. 10

[Möl01] Bodo Möller. Algorithms for multi-exponentiation. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers*, volume 2259 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2001. 47

[MSW20] Mohammad Mahmoody, Caleb Smith, and David J. Wu. Can verifiable delay functions be based on random oracles? In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020: 47th International Colloquium on Automata, Languages and Programming*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 83:1–83:17, Saarbrücken, Germany, July 8–11, 2020. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. 9

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of

Lecture Notes in Computer Science, pages 620–649, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. 14

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf. 2

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 60:1–60:15, San Diego, CA, USA, January 10–12, 2019. Leibniz International Proceedings in Informatics (LIPIcs). xiv, xv, xvi, 3, 4, 5, 16, 20, 22, 23, 24, 25, 31, 32, 33, 34, 44, 75, 84, 86, 88, 91

[Pip80] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM J. Comput.*, 9(2):230–250, 1980. 47

[Pra75] Vaughan R. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4(3):214–220, 1975. 90

[Pri16] PrimeGrid. World record Colbert number discovered! http://www.primegrid.com/forum_thread.php?id=7116, 2016. Accessed: 2022-05-18. 8, 86, 87, 100

[Pri20] PrimeGrid. Proposal: A new sierpiński problem. https://www.primegrid.com/forum_thread.php?id=9107, 2020. Accessed: 2022-05-19. 86, 91

[Pro78] François Proth. Theoremes sur les nombres premiers. *Comptes rendus de l'Académie des Sciences de Paris*, 87(926), 1878. 8, 85, 91

[PSW80] Carl Pomerance, J. L. Selfridge, and Samuel S. Wagstaff. The pseudoprimes to $25 \cdot 10^9$. *Mathematics of Computation*, 35(151):1003–1026, 1980. 90

[Rab80] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980. 8, 86, 90

[Rab83] Michael O. Rabin. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267, 1983. 2, 5

[Rie69] Hans Riesel. Lucasian criteria for the primality of $N = h \cdot 2^n - 1$. *Mathematics of Computation*, 23(108):869–875, 1969. 85, 91

[Rot21] Lior Rotem. Simple and efficient batch verification techniques for verifiable delay functions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 382–414, Raleigh, NC, USA, November 8–11, 2021. Springer, Cham, Switzerland. xv, 6, 10, 15, 17, 34, 43, 45, 46, 50, 55, 56

[RS20] Lior Rotem and Gil Segev. Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 481–509, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. 14, 67

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. 3, 86

[RSS20] Lior Rotem, Gil Segev, and Ido Shahaf. Generic-group delay functions require hidden-order groups. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 155–180, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland. 9

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, 1996. 1, 14

[SB19] Kineret Segal and Tom Brand. Presenting: VeeDo a STARK-based VDF service. Technical report, StarkWare, 2019. 9

[SB20] István András Seres and Péter Burcsi. A note on low order assumptions in RSA groups. *IACR Cryptol. ePrint Arch.*, page 402, 2020. 10

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, jan 1991. xv, 68, 69

[Sha19] Barak Shani. A note on isogeny-based hybrid verifiable delay functions. Cryptology ePrint Archive, Paper 2019/205, 2019. https://eprint.iacr.org/2019/205. 9

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer Berlin Heidelberg, Germany. 48

[SJH+21] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar R. Weippl. RandRunner: Distributed randomness from trapdoor VDFs with strong uniqueness. In *ISOC Network and Distributed System Security Symposium – NDSS 2021*, Virtual, February 21–25, 2021. The Internet Society. 2, 5

[SPG21] Michael A. Specter, Sunoo Park, and Matthew Green. KeyForge: Non-attributable email from forward-forgeable signatures. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 1755–1773. USENIX Association, August 11–13, 2021. 68

[SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977. 90

[Str64] Ernst G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70(806-808):16, 1964. 47

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, March 19–21, 2008. Springer Berlin Heidelberg, Germany. 2

[Wes20] Benjamin Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33(4):2113–2147, October 2020. xiv, 3, 5, 8, 15, 17, 31, 32, 43, 44, 68, 73, 75

[Xav22] Charles F. Xavier. Pipemsm: Hardware acceleration for multi-scalar multiplication. *IACR Cryptol. ePrint Arch.*, page 999, 2022. 49