Practical Zero-Knowledge Protocols Based on the Discrete Logarithm Assumption

Stephanie Bayer

A report submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of the
University College London.

Department of Computer Science
University College London

December 13, 2013

I, Stephanie Gertrud Maria Bayer, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

London, December 13, 2013

Stephanie Bayer

Abstract

Zero-knowledge proofs were introduced by Goldwasser, Micali, and Rackoff. A zero-knowledge proof allows a prover to demonstrate knowledge of some information, for example that they know an element which is a member of a list or which is not a member of a list, without disclosing any further information about that element. Existing constructions of zero-knowledge proofs which can be applied to all languages in NP are impractical due to their communication and computational complexity. However, it has been known since Guillou and Quisquater's identification protocol from 1988 and Schnorr's identification protocol from 1991 that practical zero-knowledge protocols for specific problems exist. Because of this, a lot of work was undertaken over the recent decades to find practical zero-knowledge proofs for various other specific problems, and in recent years many protocols were published which have improved communication and computational complexity.

Nevertheless, to find more problems which have an efficient and practical zero-knowledge proof system and which can be used as building blocks for other protocols is an ongoing challenge of modern cryptography. This work addresses the challenge, and constructs zero-knowledge arguments with sublinear communication complexity, and achievable computational demands. The security of our protocols is only based on the discrete logarithm assumption.

Polynomial evaluation arguments are proposed for univariate polynomials, for multivariate polynomials, and for a batch of univariate polynomials. Furthermore, the polynomial evaluation argument is applied to construct practical membership and non-membership arguments. Finally, an efficient method for proving the correctness of a shuffle is proposed.

The proposed protocols have been tested against current state of the art versions in order to verify their practicality in terms of run-time and communication cost. We observe that the performance of our protocols is fast enough to be practical for medium range parameters. Furthermore, all our verifiers have a better asymptotic behavior than earlier verifiers independent of the parameter range, and in real life settings our provers perform better than provers of existing protocols. The analysis of the results shows that the communication cost of our protocols is very small; therefore, our new protocols compare very favorably to the current state of the art.

Acknowledgements

The last years leading to my thesis involved a lot of hard work, but were also very interesting and I learned a lot. I want to thank now all the people I worked with in this period.

First, I would like to thank my supervisor Dr. Jens Groth, who gave me helpful feedback and took his time to give me advice whenever needed. I also want to thank my former professors Prof. Otto Forster and Prof. Martin Huxley, who believed in me and encouraged me to go down this path. Without them I would never be in the position to write this work.

Next, I would like to thank people I worked with over the years. Prof. Yvo Desmedt was my second supervisor and helped me along the path as much as his always busy schedule allowed. Thanks go also to Dr. Nicolas Courtois and Dr. Sven Schäge who encouraged me and helped me on my way.

I would like to give special thanks to Dr. Douglas Wikström for his insightful discussion on shuffles and cooperation. Dr. Christopher Wolf was so kind to invite me over to Bochum. I had an enjoyable time there with many interesting discussions.

I also want to thanks Al and Felizitas, who where kind enough to proof-read this work.

I'm full of thanks for Dr. Marko Wolf and the Munich ESCRYPT team, for helping me along the last part of my way and showing me that there is a life after the thesis. Special thanks go here to Mirko and Thomas who saved my work when I thought everything is lost.

Last but not least I want to thank Wolfi who supported me most of time over the last years and encouraged me every time I doubted in myself. Thanks goes also to Anna, without her support the life in London would have been less enjoyable and much harder.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Computers and Internet play an important role in our modern life. For a lot of everyday tasks there are online solutions available. The one which probably first comes into one's mind is the use of e-mails instead of letters. Furthermore, a lot of people do their daily banking online, rather than going to the bank in person or sending a cheque. Wikipedia was developed to help us with the task of looking up and spreading knowledge. Other web based services for completely new tasks were created over the years such as Twitter and Facebook. Users dealing with any of these services are naturally concerned about the security of their data and their privacy.

To protect privacy, anonymising networks such as Tor [Tor13] were invented. They allow a user to hide their IP address and therefore make it difficult to trace the user's Internet activity. For this reason such networks are used by a number of groups including undercover police agents, abuse victims, and citizens living under dictatorships. During the Arab Spring in 2011, for instance, the Tor network experienced a spike in users from Libya and Egypt [Din11]. However, anonymous access to services can also lead to abuse. Wikipedia, for instance, allows anonymous postings, but blocks the IP address of misbehaving users. This crude solution means that if one user of Tor abuses Wikipedia, all users whose traffic comes out of the same Tor relay with this IP-address are blocked. Thus, one misbehaving user causes many innocent users to be punished.

Different solutions to deal with this problem were suggested. Johnson et al. [JKTS07] suggested the Nymble system. In this alternative solution IP-addresses are not blocked, but instead each user anonymously proves that they have not been blacklisted. Other solutions along that line were suggested, among others, by Li and Hopper [LH11], as well as Henry and Goldberg [HG11]. Non-membership proofs were also studied in their own right, the best asymptotic solutions so far were given by Brands et al. [BDD07] and Peng [Pen11].

The reversed case of membership proofs, where the user wants to prove anonymously that they belong to a certain set, are useful when operating a whitelist access control system or group signature schemes. Membership proofs are also useful in applications such as e-auctions where users want to prove that their bids belong to a set of approved values, another application of membership proofs is e-voting.

From a voter's perspective it is important that they are able to prove that their vote belongs to a

certain set and therefore their vote is valid and be counted correctly. Apart from that, potential users of e-voting schemes may fear that the secrecy of the vote does not hold true anymore and that somebody is able to link their votes back to them after the votes are counted. Because of this, it is important to ensure the secrecy of the vote. To guarantee anonymity, e-voting protocols can be based on mix-nets. A mix-net is a multi-party protocol which allows a group of senders to input a number of encrypted messages to the mix-net, which then outputs them in random order.

It is common to construct mix-nets from *shuffles* and let mix-servers take turns in shuffling the ciphertexts. Informally, a shuffle of ciphertexts $C_1, \ldots, C_n$ is a list of ciphertexts $C'_1, \ldots, C'_n$ which contain the same plaintexts in permuted order.

As long as the shuffle is permutation hiding, which means that nobody is able to link input and output of a shuffle operation, the anonymity inside a mix-net can be ensured. Permutation hiding can be accomplished by re-encrypting the permuted ciphertexts, but then these ciphertexts could be substituted without detection. Therefore, it is also important that the correctness of a shuffle is guaranteed, and it is assured that the output ciphertexts contain the same plaintexts as the input ciphertexts. Different solutions to prove correctness were proposed, among these, solutions based on permutation matrices [FS01, Fur05, GL07, Wik09], or solutions based on the invariance of polynomials under permutation of roots [Nef01, GI08, Gro10].

It can be seen that in various settings it is desirable to construct solutions which guarantee the anonymity of the user without compromising performance or usability of the protocol. We also see that many different solutions for various problems have been published relying on different approaches. Some of these are either broken [Cha81, JJ01, JJR02] or do not guarantee full anonymity [JKTS07, HHG10, AST02]. In contrast, it seems that solutions based on *zero-knowledge* do not suffer from these drawbacks.

Zero-knowledge proofs are interactive proofs which allow a user to prove a statement without revealing more than the correctness of the claim. This concept was introduced by Goldwasser, Micali, and Rackoff [GMR85]. An interactive proof consists of communication between a computationally unbounded prover and a probabilistic polynomial time verifier. The communication consists of challenges from the verifier and answers to these challenges from the prover. The goal of the prover is to convince the verifier of the statement in such a way that the verifier always accepts, if the assertion is true. On the other hand, the prover should only be able to cheat with negligible probability, which means that there should be a very low probability of the verifier accepting a false statement. Brassard, Chaum, and Crèpeau [BCC88] pointed out that in most applications it is secure enough to restrict the cheating prover to have polynomial time. This means an unbounded prover may be able to cheat, but a polynomial time prover is only able to cheat with negligible probability. To distinguish the different settings such zero-knowledge protocols are called *arguments* [BCY91].

Zero-knowledge protocols, i.e. proofs and arguments, are very powerful tools. They allow to prove statements from every language in NP, that means languages L where $x \in L$ can be proven in polynomial time given a witness w.

After several researchers [GMW91, BCC88] had discovered that every language in NP has a zero-

knowledge proof system, the question of whether or not these protocols can be efficient and practical arose. Various aspects in a zero-knowledge proof system can be considered to be optimized, such as round complexity [FS89, BCY91], computational complexity [Sch91], and communication complexity.

A lot of theoretical research was undertaken to reduce the communication complexity for all problems in NP. Boyar et al. [BLP93, BBP91] reduced the cost of zero-knowledge proofs to subquadratic. Killian's [Kil92] construction for proofs has also subquadratic communication cost, and Cramer and Damgård [CD97] showed that it is possible to get zero-knowledge proofs with linear communication cost. Recently, Ishai et al. [IKOS07] improved this result further; their approach depends only linearly on the statement size and they gave also a version depending quasi-linearly on the witness size n, that means the size of the argument is $O(n \log(n))$. Crescenzo and Fedyukovych [CF12] have taken a different approach, and also showed that it is possible to prove all statements in NP with communication size which depends linearly on the statement size. Their protocol performs better for small circuits sizes than the method proposed by Ishai et al. [IKOS07], which still has the best asymptotic cost for big circuits.

If we consider arguments, the cost can be even lower, Cramer and Damgård's [CD97] argument for circuit satisfiability is linear in the circuit size. Killian's [Kil92, Kil95] argument has only a polylogarithmic communication cost. The recent works by Ishai et al. [IKOS07] and Goldwasser et al. [GKR08] achieved arguments whose size depends only quasilinearly on the witness size.

The work done so far shows that it is possible to construct zero-knowledge protocols with a low communication cost. However, these techniques tend to have increased computational cost and therefore they are mostly of theoretical interest. It is considered to be one of the important challenges [Joh00] in modern cryptology to construct zero-knowledge proof systems which have *both* efficient communication *and* computational complexity. Therefore, in addition to the effort to lower the amount of communication for general zero-knowledge protocols, work was undertaken to reduce the communication cost for specific problems, for example as described above, without paying the price of high computational cost.

Among others, [FO97, Bra97, CS97] gave polynomial evaluation arguments with linear cost and [FS01, Nef01, Fur05, TW10] achieved linear communication cost for shuffles. However, none of these protocols are really practical, they suffer from a big constant factor and the size of all these arguments is greater than the size of the statement itself.

In recent years more sophisticated techniques helped to reduce the argument size to sublinear. For instance Brands et al. [BDD07] stated a non-membership argument with square root cost or [GI08] proposed a sublinear shuffle argument. Furthermore, Groth [Gro09] gave different zero-knowledge arguments with sublinear size for general statements involving linear algebra, and based on this result Seo [Seo11] stated an improved version with less interaction.

This work addresses the challenge posed by Johnson [Joh00] and we construct zero-knowledge arguments with both sublinear communication complexity and practical computation complexity. The security of our protocols rests only on the discrete logarithm assumption, which conjectures that the calculation of the discrete logarithm for a random group element H to base G in various finite cyclic groups $\mathbb{G}$ is hard. More precisely, a group $\mathbb{G}$ is cyclic with finite order n if $\mathbb{G} = \{G, G^2, \ldots, G^{n-1}, G^n = 1\}$

is generated by the base element G, and the discrete logarithm of H is the unique element $x \in \mathbb{Z}_n$, for which $G^x = H$ holds.

The discrete logarithm assumption is one of the most fundamental and well-studied cryptographic assumptions. There are several types of prime order groups where the discrete logarithm assumption is believed to hold, for instance an order q subgroup of $\mathbb{Z}_p^*$ where p is a large prime, or a group of points on an elliptic curve. This yields group elements that can be much smaller than when using RSA-moduli or pairing-based cryptography, giving us an advantage compared to such schemes.

1.2 This work

One of the important challenges on modern cryptology is to find efficient zero-knowledge proof systems which can be used to construct real life protocols [Joh00]. In this work we address this problem and answer the challenge positively for zero-knowledge arguments based on the discrete logarithm assumption. Our new zero-knowledge arguments are all sublinear in the statement size; in addition, our shuffle argument, see Chapter 8, is also sublinear in the witness size. Apart from the sublinear communicational cost, which is desirable in itself, our arguments are practical in terms of computational cost. Thus, we are able to give protocols, which answer the challenge posed by Johnson [Joh00] to find more efficient zero-knowledge protocols.

We will first explain efficient protocols for various specific problems, which can be used as building blocks for more complex zero-knowledge arguments, detailed in Chapter 5. Then, we show how these techniques can be used to construct zero-knowledge arguments for different real life problems. More precisely we propose an univariate polynomial evaluation argument, a polynomial evaluation argument for multivariate polynomials, and an argument to show correct evaluation of a set of polynomials at the same time, see Chapter 6. We will also give non-membership proofs and membership proofs, see Chapter 7, and an argument to show correctness of a shuffle, which will be explained in Chapter 8. We compare all these protocols with former work and see that they compare very favorably to them. This holds especially regarding the communication cost, and all our verifiers are much lighter than the state of the art. Furthermore, we implemented our protocols to verify the practicality of the argument size and also to test the real life behavior of our protocols.

There are various reasons why our zero-knowledge arguments perform better than earlier protocols. First we have chosen our commitment scheme carefully. A commitment scheme allows the prover to commit themselves to a value without disclosing this value. Later the prover can open the commitment to reveal the value. We have chosen the general Pedersen commitment, which will be introduced in detail in Section 3.3.1. The general Pedersen commitment is homomorphic and therefore allows to verify sophisticated combinations of commitments and leads to reduced computational cost. Also the Pedersen commitment is length reducing, since it is possible to conceal many elements in one element, which is important to get sublinear communication cost. Furthermore, we will make use of a special design of the verifier's challenges. Instead of picking n different challenges x_i, the verifier constructs Vandermonde challenges $x, x^2, \ldots, x^n$. The use of the Vandermonde challenges allows us to use sophisticated batch verification and therefore reduces the communication and computational complexity.

The previous two reasons apply to all our protocols, in addition we use special techniques for

different settings. To reduce the complexity of our polynomial arguments, we have written the X^i in binary, similar to multi-exponentiation techniques, which gives us a logarithmic cost. Another trick we used for different arguments, for example, batch-polynomial evaluation or shuffle argument, is to arrange the witness in an $m \times n$ matrix, together with the general Pedersen commitment this leads to sublinear communication cost. Lastly, we used Lagrange interpolation polynomials in our constructions, which led to sophisticated ways to verify our commitments.

All these techniques combined give us zero-knowledge arguments for various applications, which all have sublinear cost and as a result compare very favorably against earlier work for the same problems.

In this work we construct our protocols as *honest verifier zero-knowledge* protocols [BMO90]; this means an honest but curious verifier will gain no knowledge from the interaction with the prover, but a dishonest verifier may be able to gain information. More precisely, we will use the stronger definition of *special honest verifier zero-knowledge* arguments [CDS94], in this case there exists a simulator $\mathcal{S}$ which can simulate the interaction between a prover and a verifier given the messages of the verifier beforehand. It is possible to transform these arguments into full zero-knowledge protocols secure against arbitrary verifiers using an OR-proof [CDS94] or standard techniques [FS89].

We will present our protocols in the plain model, that means that the security of our protocols is *only* based on the discrete logarithm assumption. The description of the finite groups and public keys used inside a protocol are part of our statement. This allows us to concentrate on the description of the protocols themselves, without worrying where the groups come from. In real life application the group description and keys could be supplied by a trusted third party that provides the environment the protocols are used in. Alternatively, the group description and the public keys can be generated by multi-party protocols of all parties involved. Another possibility is that the verifier decides on the group description and the public keys, and the prover verifies the correctness of these.

All of our protocols are described as interactive protocols, but all can be transformed to non-interactive proofs using the Fiat-Shamir heuristic [FS86] if a security proof in the random oracle model is secure enough. Nevertheless, in this work we will concentrate on the interactive setting only and therefore compare our work with interactive protocols.

The contributions of this dissertation can be summarized as follows.

1. We demonstrate how Groth's zero-knowledge argument for linear algebra relation [Gro09] can be used to construct a product argument to show $\prod_{i=1}^{N} a_i = b$ for secret a_i and public known b. This argument was joint work with Jens Groth and was published at Eurocrypt 2012 [BG12].

2. We demonstrate how Groth's zero-knowledge argument for linear algebra relation [Gro09] can be used to construct a product argument to show $\prod_{i=1}^{N} a_i = b$ for secret a_i and secret b.

3. We give a multi Hadamard product argument to show for secret vectors $\boldsymbol{a}_i, \boldsymbol{b}$ that $\boldsymbol{a}_1 \circ \ldots \circ \boldsymbol{a}_n = \boldsymbol{b}$, with sublinear complexity. This argument was published together with Jens Groth at Eurocrypt 2012 [BG12].

4. We give an argument to show that for secret vectors $\boldsymbol{a}_i, \boldsymbol{b}_i$ that $\sum_{i=1}^{m} \boldsymbol{a}_i * \boldsymbol{b}_i = 0$, for a bilinear map

$* : \mathbb{Z}_q\mathbb{Z}_q \rightarrow \mathbb{Z}_q$, with sublinear complexity. This was also published at Eurocrypt 2012 [BG12] together with Jens Groth.

5. We give a zero-knowledge argument for committed vectors $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i \in \mathbb{Z}_q^n$ satisfying $\boldsymbol{a}_i \circ \boldsymbol{b}_i = \boldsymbol{c}_i$, where $\circ$ denotes the entry-wise product. This Hadamard product argument is joint work with Jens Groth.

6. We propose an efficient and practical honest-verifier zero-knowledge polynomial evaluation argument with logarithmic communication and computational complexity. This result was published at Eurocrypt 2013 [BG13] together with Jens Groth.

7. We show how the polynomial evaluation argument can be adjusted to work with multivariate polynomials. The new argument has only polylogarithmic communication and computational complexity.

8. We explain how one can prove evaluation of L polynomials efficiently at the same time. The work leading to the argument was done together with Jens Groth.

9. We propose a practical zero-knowledge non-membership argument to show that a secret value u is not contained in a list $\mathcal{L}$, which is a straightforward application of our polynomial evaluation argument. This work was published at Eurocrypt 2013 [BG13] together with Jens Groth.

10. We give a zero-knowledge non-membership argument to show that a secret value u is not contained in a secret list $\mathcal{L}$.

11. We propose a practical zero-knowledge membership argument to show that a secret value u is contained in a list $\mathcal{L}$. This result was also published at Eurocrypt 2013 [BG13] together with Jens Groth.

12. We explain how one can simultaneously prove non-membership or membership of multiple lists This is also joint work with Jens Groth.

13. Lastly, we propose an efficient honest verifier zero-knowledge argument for the correctness of a shuffle. The argument was published at Eurocrypt 2012 [BG12] together Jens Groth.

In the following we will describe our *main* contributions in more detail.

Polynomial Evaluation: We propose efficient honest verifier zero-knowledge arguments for univariate or multivariate polynomials in Chapter 6.

We give an efficient zero-knowledge argument for two committed values u, v satisfying $P(u) = v$ for a given polynomial $P(X) \in \mathbb{Z}_q[X]$, where q is a prime and $\mathbb{Z}_q[X]$ is the ring of polynomials with coefficients in $\mathbb{Z}_q$. We work over a modular group $\mathbb{G}$ of order q and use the Pedersen commitment scheme, i.e. a commitment to u is of the form $G^u H^r$ for some $r \in \mathbb{Z}_q$. Given the coefficients of the polynomial $P(X) = \sum_{i=0}^{D} a_i X^i$ and two Pedersen commitments, our zero-knowledge argument can demonstrate knowledge of openings of the commitments to values u and v such that $P(u) = v$.

Our zero-knowledge polynomial argument is highly efficient. The communication complexity is $O(\log D)$ group and field elements, which is very small compared to the statement size of D field elements. The prover computes $O(\log D)$ exponentiations in $\mathbb{G}$ and $O(D \log D)$ multiplications in $\mathbb{Z}_q$, and the verifier computes $O(\log D)$ exponentiations in $\mathbb{G}$ and $O(D)$ multiplications in $\mathbb{Z}_q$. The constants in the expressions are small.

We also show how this zero-knowledge argument can be efficiently adapted to cover the case of multivariate polynomials. The new protocol only communicates $O((\log D)^n)$ elements for an n-variate polynomial with degree D, which leads to a much smaller argument size compared to all former approaches. Moreover, both parties have to calculate only $O((\log D)^n)$ exponentiations, but an increased number of multiplications. Although the number of multiplications dominates the computational complexity, the total computational cost is smaller than the $O(D^n)$ exponentiations needed by former approaches.

Next, we will explain how one can prove for L polynomials $P_1, \ldots, P_L$ at the same time that $P_i(u_i) = v_i$ for secret u_i, v_i without repeating our polynomial argument in parallel. This new approach leads to a communication cost of $O(\sqrt{L} \log D)$ elements, opposed to the formerly needed $O(LD)$ elements. Furthermore, the computational cost is quite efficient, the prover only needs to calculate $O(L \log D)$ exponentiations, whereas the verifier computes only $O(\sqrt{L} \log D)$ exponentiations.

Non-membership and membership proofs: In Chapter 7 we propose a practical non-membership proof and a membership proof for sets which are straightforward applications of our polynomial argument. As a consequence, the communication complexity of each proof is $O(\log D)$ group elements. This is a significant reduction compared to previous schemes with complexity $O(D)$ or $O(\sqrt{D})$. The prover's computation is a logarithmic number of exponentiations and a quasilinear number of multiplications in $\mathbb{Z}_q$. The verifier's computation is a logarithmic number of exponentiations and a linear number of multiplications in $\mathbb{Z}_q$.

Shuffle: We also propose a practically efficient honest verifier zero-knowledge argument for the correctness of a shuffle. Our argument is very efficient, in particular, we drastically decrease the communication complexity compared to previous shuffle arguments. We cover the case of shuffles of ElGamal ciphertexts, but it is possible to adapt our argument to other homomorphic encryption schemes as well. The detailed description can be found in Chapter 8.

Our shuffle argument has sublinear communication complexity. When shuffling $N = mn$ ciphertexts, arranged in an $m \times n$ matrix, our argument transmits $O(m + n)$ group elements giving a minimal communication complexity of $O(\sqrt{N})$ if we choose $m = n$. In comparison, Groth and Ishai's argument [GI08] communicates $\Theta(m^2 + n)$ group elements, and all other state of the art shuffle arguments communicate $\Theta(N)$ elements.

Practical Results: To verify the practicality of our new arguments we implemented some of our new protocols and some of the former protocols. The analysis of the results showed that our protocols perform better for real life parameters, and also that the performance is efficient enough to be practical.

The implementations of our protocols allowed us to analyze the influence of different parameters,

for example subgroup size or modulo size, on the running time and size of the argument. These results give us evidence for which parts of the protocols should be optimized further to get even more practical arguments.

When implementing the protocols we need to work over finite cyclic groups. There are many possible choices, for example subgroups of prime order modulo a large prime, or groups based on elliptic or hyper-elliptic curves. In our implementations we have focused on subgroups modulo a large prime, in order to keep our implementation as straightforward as possible. Adding the whole functionality for elliptic curves expected to increase the work disproportionately compared to the insight we expected from the results. Elements of elliptic curves are smaller; thus, applying elliptic curves to our protocols would lead to even smaller arguments. But this would be the case for all discrete logarithm based arguments. Similarly, implementations of protocols over elliptic curves tend to be faster than implementations over modular groups. Again, this is similar for all protocols based on the discrete logarithm. So, we would gain little extra knowledge by comparing our work with other discrete logarithm based arguments in this setting.

Our implementations are just straightforward implementations, i.e. our prover and verifier are simulated in the same program and communicate over simple data files; thus, we did not simulate a communication over SSL or other secure channels. The main focus of this work is to show that it is possible to construct practical zero-knowledge protocols and analyze the pure run-time of these. Establishing a secure connection between two parties adds some time on top of the execution time. This extra time would effect our results; therefore, we can justify our approach. We are also aware that our implementation can be open to general side channel attacks and that making the implementation secure against these might add extra run-time. Again, we only want to measure the pure run-time; thus, we ignored this issue for our implementation.

For the same reasons we did not set up key management. In real life applications key management is used for example to distribute cryptographic keys to parties in a system. In our work we just picked random numbers for the commitment key and the public encryption and passed these values over to the prover and verifier. We are aware that in this case an adversary might learn something relevant from the values, and as a consequence this should not be done in real life. But to change the key generation and storage does not affect the runtime of the underlying protocols, for example shuffle, which was the scope of our experiments. We can, therefore, justify the way we are dealing with the parameters.

1.3 Published Work

The univariate polynomial argument, Section 6.2, and the straightforward application of non-membership and membership arguments, Chapter 7, were published at Eurocrypt 2013 in *'Zero-knowledge Argument for Polynomial Evaluation with Application to Blacklists'* [BG13].

The shuffle argument, Chapter 8, and the underlying product argument, Section 5.3, were published at Eurocrypt 2012 in *'Efficient Zero-knowledge argument for Correctness of a Shuffle'* [BG12]. Recently, the developer of the Zeus e-voting scheme told us that they want to integrate our shuffle argument into Zeus [Zeu13]. At the moment we are working with them to find the best way to do it [Lou13]. Fur-

thermore, the Spanish e-voting company Scytl [Scy13] told us, that they are considering to include our argument in their e-voting platform.

The Hadamard argument, Section 5.4, and the batch polynomial argument, Section 6.4 are joint work with Jens Groth. So far they are unpublished. Furthermore, the multivariate polynomial argument is new and was not published before.

1.4 Thesis Structure

The thesis is organized as follows. First we will give all the background needed for our thesis. In more detail, in Chapter 2, we will briefly introduce the discrete logarithm assumption and give some overview of the history of zero-knowledge. In Chapter 3, we will state all definitions of the concepts needed during our thesis and introduce zero-knowledge formally. Next, in Chapter 4, we will introduce our approach to the implementation and discuss relevant optimizations.

In Chapter 5, we will state and explain zero-knowledge arguments for different problems, which are needed as building blocks for our zero-knowledge arguments. In Chapter 6 we describe our construction of the new polynomial evaluation arguments and also describe Brands et al.'s [BDD07] technique for the same problem. Furthermore, we discuss results from our implementation and compare our work in detail to Brands et al.'s approach. In the next chapter, Chapter 7, we explain how our polynomial argument can be applied to non-membership and membership proofs. Again we will recap Brands et al.'s [BDD07] technique and compare our result theoretically and practically with Brands et al.'s [BDD07] work. Lastly, in Chapter 8, we will state the construction of our efficient sublinear shuffle argument. Also, in this case we will discuss results obtained from our implementation. In Chapter 9, we will conclude and give ideas for possible future research.

Chapter 2

Background

This thesis is about efficient zero-knowledge arguments. We base their security on the discrete logarithm assumption. In this section we will introduce the discrete logarithm assumption and discuss the relevant aspects. Furthermore, we will introduce zero-knowledge protocols in more detail and give a short overview over the history.

2.1 Discrete Logarithm Assumption

The discrete logarithm assumption plays an important role in modern cryptography and various cryptographic tools are based on it, for example ElGamal encryption [ElG84] or the Pedersen commitment [Ped91].

Informally, the discrete logarithm assumption conjects that the calculation of the discrete logarithm is hard for a random group element H to base G in various finite cyclic groups $\mathbb{G}$. A group $\mathbb{G}$ is cyclic with order n, if there exists a generator $G \in \mathbb{G}$ such that $\mathbb{G} = \{G, G^2, \ldots, G^{n-1}, G^n = 1\}$ and $G^x \neq 1$ for $x < n$. The discrete logarithm of H is an element $x \in \mathbb{Z}_n$ for which $H = G^x$ holds.

More precisely, hard in this context means that there exists no polynomial time algorithm which can calculate the discrete logarithm of the element $H \in \mathbb{G}$. For a formal definition, see Section 3.2.

Diffie and Hellman [DH76] were the first to use the discrete logarithm assumption in the context of cryptography. They constructed a secure key exchange protocol based on the belief that the assumption holds for finite fields of prime order.

There are several types of groups where the discrete logarithm assumption is believed to hold. Among them certain prime order groups, for instance an order q subgroup of $\mathbb{Z}_p^*$ where q and p are large primes. It is also believed that the assumption holds for a group of points on an elliptic curve or on a hyper-elliptic curve.

To clarify this belief we have to investigate the different methods of calculating the discrete logarithm of an arbitrary element $H \in \mathbb{G}$. There are three classes of algorithms to solve the discrete logarithm problem. The first class consists of generic algorithms, that means the algorithms do not exploit special properties of the objects to which they are applied. Other algorithms work best for groups which have smooth group order, a group order is smooth if it is the product of small primes. The last class requires that smooth group elements exist. That means they can be represented as products of primes smaller

than a certain boundary α.

The first and obvious generic algorithm is to test the equation $G^x = H$ for all possible $x \in \mathbb{Z}_n$ until a x is found, where n is the order of the underlying group. This approach uses n group operations and therefore is not recommended for large n.

Baby Step-Giant Step [Sha71] improves the upper-bound of the run-time to find the discrete logarithm to $O(\sqrt{n})$. This method needs $O(\sqrt{n})$ space. A similar run-time is achieved by Pollard's ρ method [Pol78] without the space requirement. Pollard's ρ can be set up with constant size space.

Shoup [Sho97] showed that for any generic algorithm the lower bound of group operations to perform is at least $\Omega(\sqrt{p})$, where p is prime and divides the group order n and for all primes q, $q|n$ and it holds $q \leq p$. That means for groups with prime order p it is not possible to improve on the runtime of $\Omega(\sqrt{p})$ using generic algorithms.

For groups with smooth group order n the Pohlig-Hellman [PH78] algorithm works best. This algorithm calculates

$$O\left(\sum_{i=1}^{k} c_i(\log n + p_i)\right)$$

group operations for $n = \prod_{i=1}^{k} p_i^{c_i}$. This means the run-time to solve the discrete logarithm in groups for which the order has only small prime factors is quite small; therefore, such groups should be avoided in real life applications.

The last class of algorithms has subexponential run-time. The basic idea is to pick a factor base of small primes and use algebraic relations to solve the discrete logarithm. The algorithms can be applied to all groups for which the concept of smoothness makes sense and the group contains many smooth elements. As it is not known how to define the notion of smoothness on general elliptic curves or hyper-elliptic curves, it is not known how to find a factor base. Therefore, these algorithms cannot be applied to groups of points over elliptic curves or hyper-elliptic curves [Mil85, SS98]. However, for some special elliptic or hyper-elliptic curves it is possible to adapt the techniques [Gau00, Gau09, Die11].

To express the expected run-time in these cases we introduce the L-function by

$$L_n[\alpha] = \exp\left[\sqrt{\ln n \ln\ln n}^{\alpha}\right]$$

and the extended L-function by

$$L_n[\alpha, c] = \exp\left[(c + o(1))(\ln n)^{\alpha}(\ln\ln n)^{1-\alpha}\right],$$

where n is the order of the group we are looking at.

The first algorithms with subexponential run-time are the Index-Calculus methods. In the variant described by Pomerance [Pom87], the algorithm calculates $L_n[\sqrt{2}]$ group operations and requires $L_n[\sqrt{1/2}]$ space. Coppersmith, Odlyzko, and Schroepple's [COS86] version of Index-Calculus yields an even better run-time. Their variant only requires $L_n[1/2]$ time and space.

The last two algorithms are derived from the number field sieve factorization algorithm [LL93], namely the number field sieve (NFS) algorithm and the similar function field sieve (FFS) [Sch08]. Both algorithms work in finite fields $\mathbb{F}_{p^n}$, where p is prime; hence, in $\mathbb{F}_p = \mathbb{Z}_p$. These algorithms have a better expected run-time than the variants of Index-Calculus. The run-time of NFS is

$$L_n[1/3; (64/9)^{1/3}]$$

and FFS runs in

$$L_n[1/3; (32/9)^{1/3}].$$

We have to mention that recent work by Joux [Jou13] brought down the complexity to calculate the discrete logarithm on $\mathbb{F}_{p^n}$ to $L_n[1/4; c]$ for small p. However, this result does not affect our work as we will work over finite fields with large prime order.

We see that there are many ways to find the discrete logarithm in groups; however, none of these algorithms runs in expected polynomial time. We also have to mention the work by Shor [Sho97]. He showed that it is possible to calculate the discrete logarithm of a general group element in polynomial time providing that a quantum computer with sufficient number of bits exists. At the moment it seems that it will take a long time before such a computer can be built and we believe that the discrete logarithm assumption holds in near future.

All in all, we can justify to base our protocols on cryptographic schemes, which are secure as long as the discrete logarithm assumption holds, as we have seen that the assumption indeed holds in various groups.

2.2 Zero-Knowledge

The concept of zero-knowledge, introduced by Goldwasser, Micali, and Rackoff [GMR85] is very powerful and is used to construct various cryptographic protocols. Informally, a zero-knowledge proof is an interactive protocol, in which the prover wants to convince the verifier of the validity of their assertion without revealing more than the correctness of the claim.

An interactive proof is a protocol between two parties, called prover and verifier. The goal of the prover is to convince the verifier of the validity of an assertion. The protocol consists of multiple rounds and each round consists of a message either from the prover or the verifier. More precisely, such an interactive protocol should allow an honest prover to always convince the verifier of a true statement. That means the protocols should be *complete*. On the other hand, the protocol should guarantee that no prover can find a strategy to convince the verifier of a wrong statement except with small probability. This property is called the *soundness* of the interactive proof.

Lastly, the interactive protocol should be *zero-knowledge*. That means a malicious verifier should gain no new information from the interaction with the prover. Or in other words, zero-knowledge means that everything that the verifier can calculate after the interaction with the prover can be calculated by the verifier by the assertion itself.

Proof vs. Argument: A zero-knowledge proof is a protocol as described above, for which the requirements holds for a computationally unbounded prover and a probabilistic polynomial time verifier. Even an unbounded time prover should not be able to cheat.

Brassard et al. [BC86, BCC88] relaxed the soundness condition such that no polynomial time prover should be able to fool the verifier, but an unbounded time prover might be able to cheat. This weaker definition is good enough for most cryptographic protocols. Interactive proof systems which satisfy this definition are called arguments [BCY91].

Levels of zero-knowledge: The zero-knowledge property of an interactive proof or argument requires that the verifier learns nothing new from the communication with the prover other than what he can deduce from just seeing the statement itself. Depending how strong or weak an adversary is allowed to be we have different levels of zero-knowledge.

In a perfect world nobody, not even an unbounded time adversary, can learn anything from the transcript of the interaction. This is called *perfect* zero-knowledge. Protocols which fulfill the principle that an unrestricted adversary cannot deduce any useful information from the transcript, except with negligible probability, are *statistical* zero-knowledge. Lastly, if a probabilistic polynomial time adversary will not get any useful information from the transcript, but an unbounded adversary can deduce information, then this type is called *computational* zero-knowledge. Clearly, a perfect zero-knowledge protocol is also statistical zero-knowledge, and statistical zero-knowledge implies computational zero-knowledge.

Computational zero-knowledge is the most liberal notion of the three levels, but good enough for real life protocols. However, it is not known if in the future more powerful devices will be developed, for example a quantum computer, which allows an adversary to deduce information from the transcript. In this case the other two levels of zero knowledge guarantee a higher level of security, and therefore it is preferable to construct interactive protocols which are perfect or statistical zero-knowledge.

Proof of knowledge: Zero-knowledge proofs defined in [GMR85] are proofs of language membership that means the prover convinces the verifier that common input x is in some fixed language L. In more detail the prover wants to show that x has some property, for example is a quadratic residue, or is a 3-colorable graph. Though, sometimes the prover wants to show knowledge of some object, and in this case the definition by Goldwasser et al. does not fit. To get an adequate definition the notion of *proof of knowledge* was first suggested in [GMR89], but not formalized. Formal definitions were first given by Feige et al. [FFS88, FS89], and Tompa and Woll [TW87], but they only considered provers which can convince the verifier with non-negligible probability. This can lead to problems, as pointed out by Bellare and Goldreich [BG92]. In the same work they gave the nowadays standard definition of a proof of knowledge and also of arguments of knowledge.

Private coin vs. public coin: In the setting of interactive proof systems, defined by Goldwasser et al. [GMR85], the verifier can pick some randomness in private, and there are no restrictions on the use of the outcome. The verifier can perform any polynomial time computation on it and send the result to the prover. Therefore, this setting is called private coin. On the contrary in the setting of Babai [Bab85] it is required that the verifier shows the result of the coin toss to the prover. Thus, this is called a public

coin protocol.

In some of the early examples of interactive proofs, it was important that a verifier keeps their coin tosses secret. So, it seems that the general case of private coin is stronger and can prove more statements. However, Goldwasser and Sisper [GS86] showed that private coin interactive proofs can be transformed into public coin interactive proofs. This transformation increases the number of rounds only by an additive constant. However, the new prover needs to be super-polynomial time and the transformation does not preserve zero-knowledge. Furthermore, the transformation cannot be applied to arguments.

Okamoto [Oka96] was able to show that private coin statistical zero-knowledge proofs can be transformed into public coin zero-knowledge proofs which need a super-polynomial time prover. Similar results were found by Vadhan [Vad06], who showed that computational private coin zero-knowledge equals computational public coin zero-knowledge. Furthermore, Pass et al. [PV10] state a transformation from private coin zero-knowledge into public coin zero-knowledge which can also be applied to arguments.

Existence of zero-knowledge proofs systems: Goldwasser et al. [GMR85, GMR89] gave in their seminal work the first examples for zero-knowledge proofs. However, it was not clear how powerful the new notion is and how much can be proven using zero-knowledge protocols.

Goldreich, Micali, and Wigderson [GMW86, GMW91] showed that, assuming one-way functions exist, every language in NP has a computational zero-knowledge proof. Ben-Or et al. [BOGG$^+$88] generalized this result and showed that every language that has an interactive proof has a computational zero-knowledge proof, given a secure probabilistic encryption scheme exists. Goldreich and Kahan [GK96] explained how to construct constant-round computational zero-knowledge proofs for every language in NP.

Unfortunately, the same does not hold for statistical zero-knowledge proofs, Fortnow [For87] showed that if all languages in NP have a statistical zero-knowledge proof then the polynomial time hierarchy will collapse. However, for some problems in NP statistical zero-knowledge proofs exist.

For arguments the situation is better, Brassard et al. [BCC88] proved that every language in NP has a statistical zero-knowledge argument based on specific algebraic assumptions. Naor et al. [NOVY92] showed that the existence of one-way permutations is enough for every language in NP to have a perfect zero-knowledge argument. Furthermore, Nguyen et al. [NOV06] showed that every language in NP has a statistical zero-knowledge argument under the assumption that one-way functions exist. Feige and Shamir [FS89] gave constructions for perfect zero-knowledge arguments of knowledge for all NP.

Honest verifier zero-knowledge: The definition of zero-knowledge requires that no verifier, not even a cheating one, learns anything new from the conversation with the prover. A weaker definition of *Honest verifier zero-knowledge (HVZK)* was given by Bellare et al. [BMO90], they considered the amount of information which can be extracted by an honest verifier following the protocol.

This definition is not strong enough for cryptographic applications; however, Bellare et al. [BMO90] showed that there exists a statistical zero-knowledge proof for each problem which has a statistical HVZK proof, under the assumption that the discrete logarithm assumption or the factoring assumption

holds. Ostrovsky et al. [OVY93] have proven that the same result holds under the more general assumption that one-way permutation exists. Okamoto [Oka96] were able to show that if a language L has a statistical HVZK proof then L has a statistical zero-knowledge proof, given one-way functions exist.

Damgård [Dam93] could show that each constant round public coin statistical HVZK proof or argument can be transformed in a constant round public coin statistical zero-knowledge proof system. This transformation does not rely on any computational assumption. Damgård et al. [DGOW95] proposed other transformations from constant round public coin HVZK to constant round public coin zero-knowledge with less round complexity, but these transformations can only be applied to proofs. Goldreich et al. [GSV98] gave the first transformation of statistical HVZK into statistical general zero-knowledge under no condition, which holds for all interactive proof systems. Vadhan [Vad06] showed that the same is true for computational zero-knowledge.

All the transformations above are not practically viable, a more practical approach was given by Feige and Shamir [FS89] provided the discrete logarithm holds. Also, Groth [Gro04], Jarecki and Lysyanskaya [JL00], Damgård [Dam00], Garay et al. [GMY06] gave ways to transform HVZK proof systems into zero-knowledge proof systems which cost only a small number of elements, see Section 3.5.

Optimizations: All general constructions of zero-knowledge proofs and arguments are not practical and efficient. Naturally, the question arose whether or not interactive proof protocols can be more efficient. Various aspects in zero-knowledge proof systems can be considered to be optimized, that is the number of rounds, communication complexity, and computational complexity.

Brassard, Crépeau, and Yung [BCY91] constructed a 6-move perfect zero-knowledge argument for all languages in NP. Whereas, Feige and Shamir [FS89] showed that it is possible to construct computational zero-knowledge arguments of knowledge with 4-rounds given some algebraic assumptions or in 5-rounds assuming that one-way functions exist. Similar results were stated by Bellare et al. [BJY97] they showed the existence of 4-round zero-knowledge arguments of knowledge, given that one-way functions exist. As shown by Goldreich and Krawczyk [GK90] this result is optimal, unless the language is in BPP.

In the case of proofs, Goldreich and Kahan [GK96] explain how to construct constant round zero-knowledge proofs with 5-rounds, under widely believed number-theoretical assumptions. Results by Katz [Kat12] indicated that for computational zero-knowledge that 5-rounds are optimal and therefore the result by Goldreich and Kahan are optimal. Recently, Ong and Vadhan [OV08] showed that indeed all languages which have a statistical zero-knowledge proof have a constant round statistical zero-knowledge proof.

These results only hold for zero-knowledge proofs for language membership and it was not clear if constant round proofs of knowledge with constant rounds exist. The gap was closed by Lindell [Lin10] who showed the existence of 5-round computational zero-knowledge proofs of knowledge for all languages in NP.

Round complexity is one aspect which can be optimized, another one is the amount of data sent between both parties. In protocols with a unbounded number of rounds a basic protocol was repeated k

times to achieve a security of 2^{-k}, this approach has a cost of at least $\Omega(kn)$ bits, where n is the statement size. Special techniques allowed Boyar et al. [BLP93, BBP91] to reduce this cost to subquadratic for any language L and Kilian [Kil92] reduced this cost slightly for proofs using PCPs.

Cramer and Damgård [CD97] were able to construct a zero-knowledge proof for all NP with linear communication cost and constant number of rounds. Recently, Ishai et al. [IKOS07] improved this result further, their approach depends only linearly on the statement size and they gave also a version depending quasilinearly on the witness size. Crescenzo and Fedyukovych [CF12] have taken a different approach and also showed that it is possible to prove all statements in NP with communication cost which depends linearly on the statement size. Their proof performs better than [IKOS07] for small circuits; however, for big circuits the technique by [IKOS07] gives better performance.

Cramer and Damgård's [CD97] technique for arguments turns out to have the same complexity as the technique by the authors for proofs. But for arguments the communication cost can be even lower. Kilian's [Kil92, Kil95] technique leads to arguments for each language in NP with constant number of rounds and a communication with polylogarithmic cost in the statement size. The recent work by Ishai et al. [IKOS07], and Goldwasser et al. [GKR08] achieved arguments with communication depending only quasilinearly on the witness size.

The last important aspect which can be optimized is the computational complexity. Since Schnorr's [Sch91] construction to prove identification in zero-knowledge, it has been known that it is possible to have zero-knowledge proofs and arguments with low computational complexity. However, all general constructions to prove statements in NP requires heavy reductions to special problems; thus, they are not efficient in this respect. Moreover, it is hard to give good lower bounds on the computational complexity, as all protocols have to read at least the statement. However, work was undertaken to reduce the cost of zero-knowledge protocols for specific problems. For instance, [BDD07, Pen11] for non-membership arguments or [CD98, Gro09, Gro11] for circuit satisfiability.

Chapter 3

Preliminaries

In this chapter we will give definitions of the key concepts needed to construct our zero-knowledge arguments. When arguing that a protocol is correct we will use homomorphic commitment schemes extensively, different commitment schemes can be used. We will describe our work using the Pedersen commitment scheme [Ped91] as it is based on the discrete logarithm. Our shuffle protocol works for different types of homomorphic encryption schemes, for example ElGamal encryption [ElG84] or Paillier encryption [Pai99]. We will focus on ElGamal encryption since it is based on the discrete logarithm assumption and also for notational reasons. Finally, we give precise definitions of what we mean by honest verifier zero knowledge arguments.

3.1 Notation

Definition 1 (Negligible, Overwhelming)**.** We say a function $f : \mathbb{N} \rightarrow [0, 1]$ is negligible if

$$f(x) = O(x^{-c})$$

for every constant $c > 0$. We say $1 - f$ is overwhelming if f is negligible.

To define security in our protocols, we will use a security parameter λ. The security parameter λ is written in unary 1^λ and is given as input to all parties in our protocols. Intuitively, the higher the security parameter the more secure the protocol, see also Section 4.1.1. Formally, we define security in the following by saying a protocol has security level λ if an adversarial algorithm requires at least 2^λ steps to succeed.

We write $y = A(x; r)$ when the algorithm A, on input x and randomness r, outputs y. We write $y \leftarrow A(x)$ for the process of picking randomness r uniformly at random and setting $y = A(x; r)$. We also write $y \leftarrow S$ for sampling y uniformly at random from a set S.

We will denote field elements from $\mathbb{Z}_q$ with lower case characters. For randomizers we will stick throughout the thesis to $r, s, t, \rho, \sigma, \tau \in \mathbb{Z}_q$ and to $x, y, z \in \mathbb{Z}_q^*$ for challenges, which are sent by the verifier. The encryption scheme and the commitment scheme may use different underlying groups, but we require that both groups have the same prime order q. We will write $\mathbb{G}$ for the group used by the commitment scheme and write $\mathbb{H}$ for the ciphertext space.

We will use bold letters for vectors, for example $\boldsymbol{a} = (a_1, \ldots, a_n)$ or $\boldsymbol{M} = (M_1, \ldots, M_m)$, and use the standard notation of upper case letters for matrices.

For vectors of group elements, we write $\boldsymbol{X} \circ \boldsymbol{Y} = (X_1 Y_1, \ldots, X_n Y_n)$ for the entry-wise product and correspondingly $\boldsymbol{X}^z = (X_1^z, \ldots, X_n^z)$. We write $\boldsymbol{x}_\pi$ if the entries of vector $\boldsymbol{x}$ are permuted by the permutation π, i.e. $\boldsymbol{x}_\pi = (x_{\pi(1)}, \ldots, x_{\pi(n)})$. For vectors of field elements, we use the standard inner product $\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i$.

3.2 Discrete Logarithm Assumption

All our protocols can be used with different homomorphic commitment schemes and homomorphic encryption schemes. We will focus throughout the work on schemes based on the discrete logarithm assumption which goes back to the work of Diffie and Hellman [DH76]. This assumption is one of the most fundamental and well-studied cryptographic assumptions, see also Section 2.1.

Definition 2 (Discrete Logarithm Problem)**.** Let $\mathbb{G}$ be a multiplicative cyclic group with generator G and order n. Given an $H \in \mathbb{G}$ the discrete logarithm problem is to find an $x \in \mathbb{Z}_n$ such that $H = G^x$, i.e. to find the discrete logarithm x of H to base G.

Definition 3 (Discrete Logarithm Assumption)**.** Let $\mathbb{G}$ be a multiplicative cyclic group with generator G and order n. The discrete logarithm assumption for $\mathbb{G}$ holds if for all non-uniform probabilistic polynomial time algorithms $\mathcal{A}$

$$\Pr[H \leftarrow \mathbb{G}; x \leftarrow \mathcal{A}(\mathbb{G}, n, G, H) : G^x = H]$$

is negligible.

3.3 Homomorphic Commitment

A commitment scheme is a way to commit to a value or a vector without revealing these values. Given an opening it is later possible to reveal the original message.
A commitment scheme consists of

- a probabilistic key generation algorithm $\mathcal{G}$
- a commitment algorithm com_{ck}
- an opening algorithm op_{ck}.

On input of the security parameter 1^λ the key generation algorithm produces

- a public commitment key ck
- a message space $\mathcal{M}_{ck}$
- a randomizer space $\mathcal{R}_{ck}$
- a commitment space $\mathcal{C}_{ck}$
- an opener space $\mathcal{O}_{ck}$.

The commitment space $\mathcal{C}_{ck}$ is determined by the choices of the commitment key, the message and randomizer space, and the commitment algorithm. The commitment algorithm takes an $m \in \mathcal{M}_{ck}$ and an $r \in \mathcal{R}_{ck}$ as input and computes $c_m = \mathrm{com}_{ck}(m; r)$, the commitment of m. Given the opening $d \in \mathcal{O}_{ck}$ and commitment c_m the opening algorithm can reveal the original message $m = \mathrm{op}_{ck}(c_m, d)$. In many commitment schemes the opening d contains the message m and the randomness r.

We require the commitment scheme to be binding and hiding. Informally, binding means it is not possible to find two messages $m_1, m_2 \in \mathcal{M}_{ck}, m_1 \neq m_2$ and $r_1, r_2 \in \mathcal{R}_{ck}$ such that $c = \mathrm{com}_{ck}(m_1, r_1)$ is also a commitment to m_2, r_2. Hiding requires that the commitment reveals no information about the message.

Definition 4 (Binding)**.** A commitment scheme is (computationally) binding if for any probabilistic polynomial time adversary $\mathcal{A}$

$$\Pr[ck \leftarrow \mathcal{G}(1^\lambda); \quad (m_1, r_1, m_2, r_2) \leftarrow \mathcal{A}(ck) : \\ m_1, m_2 \in \mathcal{M}_{ck}, r_1, r_2 \in \mathcal{R}_{ck}, m_1 \neq m_2 \ \wedge \mathrm{com}_{ck}(m_1; r_1) = \mathrm{com}_{ck}(m_2; r_2)]$$

is negligible in λ.

If this holds for unbounded adversary $\mathcal{A}$ the commitment scheme is unconditionally binding or perfectly binding.

Definition 5 (Hiding)**.** A commitment scheme is (computationally) hiding if for any probabilistic polynomial time adversary $\mathcal{A}$

$$\big| \Pr[ck \leftarrow \mathcal{G}(1^\lambda); (m_1, r_1, m_2, r_2) \leftarrow \mathcal{A}(ck); \\ m_1, m_2 \in \mathcal{M}_{ck}, r_1, r_2 \in \mathcal{R}_{ck}; c \leftarrow \mathrm{com}_{ck}(m_1, r_1) : \mathcal{A}(c) = 1] \\ - \Pr[ck \leftarrow \mathcal{G}(1^\lambda); ((m_1, r_1, m_2, r_2) \leftarrow \mathcal{A}(ck); \\ m_1, m_2 \in \mathcal{M}_{ck}, r_1, r_2 \in \mathcal{R}_{ck}; c \leftarrow \mathrm{com}_{ck}(m_2, r_2) : \mathcal{A}(c) = 1] \big|$$

is negligible in λ.

A commitment scheme is unconditionally or perfectly hiding if this condition also holds for unbounded $\mathcal{A}$.

In addition, a commitment scheme may fulfill the trapdoor property. In this case the generator $\mathcal{G}$ also outputs a trapdoor t, which allows us to open a commitment $c_m = \mathrm{com}_{ck}(m; r)$ to any message m'. In other words, without knowledge of the trapdoor t the commitment scheme is binding, but given the trapdoor it is possible to cheat arbitrarily.

The trapdoor property seems to contradict the task of a commitment scheme, which is to allow a prover to bind themselves to a value and later reveal exactly this value. However, as long as the prover does not know the trapdoor, the commitment scheme is binding. Therefore, trapdoor commitments can be used as normal commitment schemes. Furthermore, the trapdoor property is important for some pro-

tocols. For instance, it can be used to convert honest verifier zero-knowledge arguments into arguments which are zero-knowledge with respect to any verifier, see also Section 3.5.

We will require for our work that the commitment scheme is homomorphic.

Definition 6 (Homomorphic)**.** A commitment scheme is homomorphic if for $ck \leftarrow \mathcal{G}(1^\lambda)$ the message space $\mathcal{M}_{ck}$, the randomizer space $\mathcal{R}_{ck}$, and the commitment space $\mathcal{C}_{ck}$ are additive abelian groups and

$$\mathrm{com}_{ck}(a+b; r+s) = \mathrm{com}_{ck}(a; r)\mathrm{com}_{ck}(b; s), \quad \forall a, b \in \mathcal{M}_{ck}, r, s \in \mathcal{R}_{ck}.$$

In addition we also require for some of our arguments that it is possible to calculate a commitment to n elements in $\mathbb{Z}_q$, where q is a large prime, at the same time. Many homomorphic commitments schemes with this property can be used, but for convenience we just focus on a generalization of the Pedersen commitment scheme [Ped91].

3.3.1 Generalized Pedersen Commitment

The general Pedersen commitment scheme allows commitment to n elements at the same time and works for cyclic groups. More precisely, on input of 1^λ and 1^n the key generation algorithm $\mathcal{G}$ outputs

- a cyclic group $\mathbb{G}$ with prime order q and security level λ, and $\mathcal{M}_{ck} = \mathbb{Z}_q^n, \mathcal{R}_{ck} = \mathbb{Z}_q$ and $\mathcal{C}_{ck} = \mathbb{G}$
- a commitment key $ck = (\mathbb{G}, G_1, \ldots, G_n, H)$ where $G_1, \ldots, G_n, H$ are random generators of the group $\mathbb{G}$
- the opening space $\mathcal{O}_{ck} = \mathcal{M}_{ck} \times \mathcal{R}_{ck}$

To conceal n elements $(a_1, \ldots, a_n) \in \mathbb{Z}_q^n$ we pick randomness $r \in \mathbb{Z}_q$ and compute

$$\mathrm{com}_{ck}(a_1, \ldots, a_n; r) = H^r \prod_{i=1}^{n} G_i^{a_i}.$$

Calculating a commitment to less than n elements is possible, this is done by setting the remaining entries a_i to 0 and the special case $n = 1$ corresponds to the standard Pedersen commitment. For instance, if the commitment key is $ck = (\mathbb{G}, G_1, \ldots, G_n, H)$ and the commitment should be calculated to $m < n$ elements $(a_1, \ldots, a_m)$, we set $a_j = 0$ for $mj \leq n$.

The commitment is computationally binding under the discrete logarithm assumption, i.e a probabilistic polynomial time adversary has negligible probability of finding two different openings of the same commitment c. To find one opening, the adversary can choose a randomness r at random and then try to find suitable message m. This yields to the equation $G^m = cH^{-r} = H_1$. Since the discrete logarithm assumption holds in the underlying group, the probability of finding m is negligible. Furthermore, since the randomness r is picked uniformly from the randomizer space $\mathcal{R}_{ck}$, the commitment is uniformly distributed in $\mathbb{G}$ no matter what the messages are. Therefore, the commitment scheme is considered to be perfectly hiding.

The general Pedersen commitment has a trapdoor, this trapdoor consists of $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_q^n$ such that

$$G_i = H^{x_i}, \text{ for } i = 1, \ldots, n.$$

Given a commitment $c = \mathrm{com}_{ck}(\boldsymbol{m}; r)$ and the trapdoor $\boldsymbol{x}$ we can open c to $\boldsymbol{m'} \in \mathbb{Z}_q^n, r' \in \mathbb{Z}_q$ by setting

$$m_i = \overline{m}_i + m_i' \quad \text{for} \quad \overline{m}_i \in \mathbb{Z}_q, \quad i = i, \ldots, n$$

and

$$r = r' - \sum_{i=1}^{n} \overline{m}_i x_i.$$

In this case

$$\begin{aligned} c &= G_1^{m_1} G_2^{m_2} G_3^{m_3} \cdot \ldots \cdot G_n^{m_n} H^r \\ &= G_1^{\overline{m}_1 + m_1'} G_2^{\overline{m}_2 + m_2'} \cdot \ldots \cdot G_n^{\overline{m}_n + m_n'} H^{r' - \sum_{i=1}^{n} \overline{m}_i x_i} \\ &= G_1^{\overline{m}_1 + m_1' - \overline{m}_1} G_2^{\overline{m}_2 + m_2' - \overline{m}_2} \cdot \ldots \cdot G_n^{\overline{m}_n + m_n' - \overline{m}_n} H^{r'} \\ &= G_1^{m_1'} G_2^{m_2'} \cdot \ldots \cdot G_n^{m_n'} H^{r'}. \end{aligned}$$

The generalized Pedersen commitment scheme is homomorphic, for all $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_q^n$ and $r, s \in \mathbb{Z}_q$ we have

$$\mathrm{com}_{ck}(\boldsymbol{a}; r)\mathrm{com}_{ck}(\boldsymbol{b}; s) = H^r \prod_{i=1}^{n} G_i^{a_i} \cdot H^s \prod_{i=1}^{n} G_i^{b_i} = H^{r+s} \prod_{i=1}^{n} G_i^{a_i + b_i} = \mathrm{com}_{ck}(\boldsymbol{a} + \boldsymbol{b}; r + s).$$

We stress that a commitment consists of a single group element no matter how big n is, as the product of n group elements is a group element. This means the commitment scheme is length reducing and we can commit to n elements with a single small commitment. This property is crucial to get sublinear communication cost.

We use the generalized Pedersen commitment scheme in the work because of its elegance and its security resting on the discrete logarithm assumption. However, our protocols could also work with other homomorphic commitment schemes which allows to calculate a commitment to n elements at the same time. We will describe our protocols in a way such that it would be easy to plug in another homomorphic commitment scheme.

Notation: For a commitment to a value a we will write $c_a = \mathrm{com}_{ck}(a; r)$. Moreover, for a matrix $A \in \mathbb{Z}_q^{n \times m}$ with columns $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ we shorten notation by defining $\mathrm{com}_{ck}(A; \boldsymbol{r}) = (\mathrm{com}_{ck}(\boldsymbol{a}_1; r_1), \ldots, \mathrm{com}_{ck}(\boldsymbol{a}_m; r_m))$. We will also abuse this notation slightly and define the com-

mitment to $\boldsymbol{a} \in \mathbb{Z}_q^N$ where $N = mn$ as

$$\mathrm{com}_{ck}(\boldsymbol{a}; \boldsymbol{r}) = \big(\mathrm{com}_{ck}(a_1, \ldots, a_n; r_1),\ \ldots,\ \mathrm{com}_{ck}(a_{((m-1)n+1)}, \ldots, a_N; r_m)\big).$$

We define a bilinear map

$$\mathbb{G}^n \times \mathbb{Z}_q^n \to \mathbb{G} \quad \text{by} \quad \boldsymbol{c}^{\,\boldsymbol{b}} = (c_1, \ldots, c_m)^{(b_1, \ldots, b_m)^T} = \prod_{j=1}^{m} c_j^{b_j}$$

and for a matrix B with columns $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m$ we define $\boldsymbol{c}^{\,B} = \big(\boldsymbol{c}^{\,\boldsymbol{b}_1}, \ldots, \boldsymbol{c}^{\,\boldsymbol{b}_m}\big)$. It is useful to observe that the underlying linear algebra behaves nicely, i.e. $\mathrm{com}_{ck}(A; \boldsymbol{r})^{\boldsymbol{b}} = \mathrm{com}_{ck}(A\boldsymbol{b}; \boldsymbol{r} \cdot \boldsymbol{b})$ and $\mathrm{com}_{ck}(A; \boldsymbol{r})^B = \mathrm{com}_{ck}(AB; \boldsymbol{r}B)$.

3.4 Homomorphic Encryption

A public key encryption scheme consists of a set of probabilistic polynomial time algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. The key generation algorithm $\mathcal{G}$, generates

- a public key pk
- a secret key sk
- a message space $\mathcal{M}_{pk}$
- a randomizer space $\mathcal{R}_{pk}$
- a cipher space $\mathcal{C}_{pk}$.

The encryption algorithm $\mathcal{E}$ takes $M \in \mathcal{M}_{pk}$, $r \in \mathcal{R}_{pk}$, and the public key pk as input and computes $C = \mathcal{E}(M, r, pk) \in \mathcal{C}_{pk}$. We will write in the following $\mathcal{E}(M, r, pk) = \mathcal{E}_{pk}(M, r)$. Whereas the decryption algorithm $\mathcal{D}$ takes $C \in \mathcal{C}_{pk}$ and the secret key sk as input and outputs $M = \mathcal{D}(C, sk) \in \mathcal{M}_{pk}$ or $\perp$ for failure. We will write $\mathcal{D}(C, sk) = \mathcal{D}_{sk}(C)$. $\mathcal{E}_{pk}$ and $\mathcal{D}_{sk}$ are chosen such that they fulfill

$$\Pr[(pk, sk) \leftarrow \mathcal{G}(1^\lambda); (M, r) \leftarrow \mathcal{M}_{pk} \times \mathcal{R}_{pk} : \mathcal{D}_{sk}(\mathcal{E}_{pk}(M; r)) = M]$$

is overwhelming.

For this work we require that the encryption scheme is at least IND-CPA secure, that means an adversary seeing a ciphertext learns nothing about the message inside.

Definition 7 (IND-CPA)**.** An encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is indistinguishable under chosen plaintext (IND-CPA) attack if for any polynomial time adversary $\mathcal{A}$

$$\begin{aligned} &|\Pr[(pk, sk) \leftarrow \mathcal{G}; (M_1, M_2) \leftarrow \mathcal{A}, |M_1| = |M_2|; c \leftarrow \mathcal{E}_{pk}(M_1) : \mathcal{A}(c) = 1] \\ &- \Pr[(pk, sk) \leftarrow \mathcal{G}; (M_1, M_2) \leftarrow \mathcal{A}, |M_1| = |M_2|; c \leftarrow \mathcal{E}_{pk}(M_2) : \mathcal{A}(c) = 1]| \end{aligned}$$

is negligible in the security parameter λ.

Another requirement for our encryption scheme is that it has to be homomorphic.

Definition 8 (Homomorphic)**.** An encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is homomorphic if for any pair $(pk, sk) \leftarrow \mathcal{G}(1^\lambda)$ and $\mathcal{M}_{pk}, \mathcal{C}_{pk} \leftarrow \mathcal{G}(1^\lambda)$ the message space $\mathcal{M}_{pk}$ and the cipher space $\mathcal{C}_{pk}$ are abelian groups, and for all $M_1, M_2 \in \mathcal{M}_{pk}$, $r_1, r_2 \in \mathcal{R}_{pk}$ and the encryption function satisfies

$$\mathcal{E}_{pk}(M_1 M_2; r_1 + r_2) = \mathcal{E}_{pk}(M_1; r_1)\mathcal{E}_{pk}(M_2; r_2).$$

Definition 9 (Re-encryption)**.** Let $M \in \mathcal{M}_{pk}, r \in \mathcal{R}_{pk}$ and $C = \mathcal{E}_{pk}(M, r) \in \mathcal{C}_{pk}$, $C' \in \mathcal{C}_{pk}$ is a re-encryption of C if there exist $s \in \mathcal{R}_{pk}$ such that $C' = \mathcal{E}_{pk}(1; s) \cdot C$ and $\mathcal{D}_{sk}(C') = \mathcal{D}_{sk}(C)$.

Our shuffle argument works with many different homomorphic encryption schemes where the message space has large prime order q. We will focus on the ElGamal encryption [ElG84] as its security is based on the discrete logarithm assumption and is IND-CPA secure, but also for notational convenience.

3.4.1 ElGamal Encryption

The ElGamal encryption scheme [ElG84] in this group works as follows. On input 1^λ the key generation algorithm $\mathcal{G}$

- outputs a cyclic group $\mathbb{G}$ with large prime order q
- picks the secret key $sk = x \in \mathbb{Z}_q^*$ at random
- sets the public key $pk = (\mathbb{G}, Y)$, where $Y = G^x \in \mathbb{G}$.

To encrypt a message $M \in \mathbb{G}$ we choose a random $\rho \in \mathbb{Z}_q$ and compute the ciphertext

$$\mathcal{E}_{pk}(M; \rho) := (G^\rho, Y^\rho M) = (U, V)$$

belonging to the ciphertext space $\mathbb{H} = \mathbb{G} \times \mathbb{G}$.

To decrypt a ciphertexts $(U, V) \in \mathcal{C}_{pk}$ we compute $M = VU^{-x}$.

The ElGamal encryption scheme is homomorphic with entry-wise multiplication. For all pairs $(M_1, \rho_1), (M_2, \rho_2) \in \mathbb{G} \times \mathbb{Z}_q$ it holds that

$$\begin{aligned}\mathcal{E}_{pk}(M_1 M_2; \rho_1 + \rho_2) &= (G^{\rho_1+\rho_2}; Y^{\rho_1+\rho_2} M_1 M_2) = (G^{\rho_1}, Y^{\rho_1} M_1)(G^{\rho_2}, Y^{\rho_2} M_2) \\ &= \mathcal{E}_{pk}(M_1; \rho_1)\mathcal{E}_{pk}(M_2; \rho_2).\end{aligned}$$

We will by default assume that the ciphertexts used in the shuffle are valid, i.e. for each ciphertext C we have $C \in \mathbb{H} = \mathbb{G} \times \mathbb{G}$. For the most common choices of the group $\mathbb{G}$ used in practice this is something that can be tested quite easily by any interested party.

Notation: In the whole thesis we will use the upper-case letter C to denote a ciphertext. For $\boldsymbol{M} = (M_1, \ldots, M_n)$ and $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$ we define

$$\mathcal{E}_{pk}(\boldsymbol{M}; \boldsymbol{\rho}) = \big(\mathcal{E}_{pk}(M_1; \rho_1),\ \ldots,\ \mathcal{E}_{pk}(M_m; \rho_m)\big).$$

We also define a bilinear map

$$\mathbb{H}^n \times \mathbb{Z}_q^n \to \mathbb{H} \quad \text{by} \quad \boldsymbol{C}^{\boldsymbol{a}} = (C_1, \ldots, C_n)^{(a_1,\ldots,a_n)^T} = \prod_{i=1}^{n} C_i^{a_i}.$$

For a matrix $A \in \mathbb{Z}_q^{n\times m}$ with column vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ we define $\boldsymbol{C}^A = (\boldsymbol{C}^{\boldsymbol{a}_1}, \ldots, \boldsymbol{C}^{\boldsymbol{a}_m})$. It is useful to observe that $(\boldsymbol{C}^A)^B = \boldsymbol{C}^{AB}$.

3.5 Generalized $\Sigma-$Protocols

For all our arguments we consider a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ which are both probabilistic polynomial time interactive algorithms.

Let R be a polynomial time decidable binary relation, we call w a witness for a statement a if $(a, w) \in R$. We define the language

$$L_R := \{a \mid \exists w : (a, w) \in R\}$$

as the set of statements a that have a witness w for the relation R.

The public transcript produced by $\mathcal{P}$ and $\mathcal{V}$ is denoted by $tr \leftarrow \langle \mathcal{P}(\mathfrak{s}), \mathcal{V}(\mathfrak{t})\rangle$ when both parties interact on inputs $\mathfrak{s}$ and $\mathfrak{t}$. The transcript consists of the initial message from the prover, the challenges from the verifier, the answers from the prover and the decision to accept or reject from the verifier. We write $\langle \mathcal{P}(\mathfrak{s}), \mathcal{V}(\mathfrak{t})\rangle = b$ depending on whether the verifier rejects $b = 0$, or accepts $b = 1$.

Definition 10 (Argument)**.** $(\mathcal{P}, \mathcal{V})$ is called an *argument* for a relation R with perfect completeness if for all non-uniform polynomial time interactive adversaries $\mathcal{A}$ we have:
Perfect completeness:

$$\Pr[(a, w) \leftarrow \mathcal{A}(\text{hist}) : (a, w) \notin R \text{ or } \langle \mathcal{P}(a, w), \mathcal{V}(a, hist)\rangle = 1] = 1$$

Computational soundness:

$$\Pr[a \leftarrow \mathcal{A}(\text{hist}) : a \notin L_R \text{ and } \langle \mathcal{A}, \mathcal{V}(a, hist)\rangle = 1]$$

is negligible, where $hist$ contains all information an adversary can obtain before they output a statement and a witness.

Definition 11 (Public coin)**.** An argument $(\mathcal{P}, \mathcal{V})$ is called *public coin* if the verifier chooses their messages uniformly at random and independently of the messages sent by the prover, i.e. the challenges correspond to the verifier's randomness ρ.

An argument is zero-knowledge if it does not leak information about the witness beyond what can be inferred from the truth of the statement. We will present arguments that have special honest verifier zero-knowledge in the sense that if the verifier's challenge is known in advance, then it is possible to simulate the entire argument without knowing the witness.

Definition 12 (SHVZK)**.** A public coin argument $(\mathcal{P}, \mathcal{V})$ is called a *perfect special honest verifier zero-knowledge (SHVZK)* argument for R if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned}&\Pr[(a, w, \rho) \leftarrow \mathcal{A}(\text{hist}); tr \leftarrow \langle \mathcal{P}(a, w), \mathcal{V}(a; \rho)\rangle : (a, w) \in R \text{ and } \mathcal{A}(tr) = 1]\\ =&\Pr[(a, w, \rho) \leftarrow \mathcal{A}(\text{hist}); tr \leftarrow \mathcal{S}(a, \rho) : (a, w) \in R \text{ and } \mathcal{A}(tr) = 1]\end{aligned}$$

where ρ is the public coin randomness used by the verifier as the challenge.

Most SHVZK arguments in literature follow a special 3-move structure, that means the transcript consists of the initial message $\mathfrak{a}$ of the prover, one random challenge x from the verifier, and the final answer $\mathfrak{b}$ of the prover. Most of these arguments also fulfill the special soundness property.

Definition 13 (Special Soundness)**.** An argument $(\mathcal{P}, \mathcal{V})$ has *perfect special soundness* if there exists a polynomial time extractor E, such that for all adversaries $\mathcal{A}$ we have:

$$\begin{aligned}&\Pr[(\mathfrak{a}, tr_1, tr_2) \leftarrow \mathcal{A}(hist), tr_1 = (\mathfrak{a}, x_1, \mathfrak{b}_1), tr_2 = (\mathfrak{a}, x_2, \mathfrak{b}_2), x_1 \neq x_2;\\ &w \leftarrow E(\mathfrak{a}, tr_1, tr_2) : \mathcal{V}(tr_1) \wedge \mathcal{V}(tr_2) = 0 \text{ or } (x, w) \in R] = 1.\end{aligned}$$

If the extractor E can extract the witness only with overwhelming probability, then the argument has *special soundness*.

That means given two accepting transcripts with different challenges x_1, x_2, it is possible to efficiently compute a witness w such that $(a, w) \in R$.

Definition 14 (Σ-Protocol)**.** A 3-move SHVZK argument $(\mathcal{P}, \mathcal{V})$ with general special soundness is called a $\Sigma-$*Protocol* and a 3-move SHVZK argument $(\mathcal{P}, \mathcal{V})$ with perfect general special soundness is called a perfect $\Sigma-$*Protocol*.

In our work the special use of Vandermonde challenges $x, x^2, \ldots, x^n$ makes it impossible to extract witnesses given only two accepting arguments. However, given $n + 1$ accepting witnesses with different challenges x it is possible to extract witnesses. Furthermore, some of the protocols require more than one challenge, and some of the protocols consist of more than three rounds. In all these cases the standard definition of a $\Sigma-$Protocol does not fit and we have to generalize the definition.

Definition 15 (Generalized Special Soundness)**.** An argument $(\mathcal{P}, \mathcal{V})$ has *perfect general special soundness* if there exists a polynomial time extractor E, such that for all adversaries $\mathcal{A}$ we have:

$$\begin{aligned}&\Pr[(\mathfrak{a}, Tr) \leftarrow \mathcal{A}(hist), Tr = \{tr_1, \ldots, tr_n\}, tr_i = (\mathfrak{a}, \boldsymbol{x}_i, \mathfrak{b}_i);\\ &w \leftarrow E(b, Tr) : \left(\bigwedge_{i=1}^{n} \mathcal{V}(tr_i)\right) = 0 \text{ or } (a, w) \in R] = 1\end{aligned}$$

where $\boldsymbol{x}_i$ contains all challenges from the verifier and $\boldsymbol{x}_{ik} \neq \boldsymbol{x}_{jk}$ for all $1 \leq i, j \leq n$, $1 \leq k \leq |\boldsymbol{x}_i|$, and $\mathfrak{b}_i$ contains all answers to $\boldsymbol{x}_i$ of the prover.

If the extractor E can extract the witness only with overwhelming probability, than the argument has *general special soundness*.

The definition of general special soundness implies that given a long enough list of accepting transcripts with different challenges $\boldsymbol{x}$ it is possible to extract a witness w for a statement a independent of the required structure of the randomness and the number of moves.

Definition 16 (Generalized Σ-Protocol)**.** A SHVZK argument $(\mathcal{P}, \mathcal{V})$ with general special soundness is called a *generalized* $\Sigma-$*Protocol*.

A perfect SHVZK argument $(\mathcal{P}, \mathcal{V})$ with perfect general special soundness and perfect completeness is called a *perfect generalized* $\Sigma-$*Protocol*.

For a 3-move argument $(\mathcal{P}, \mathcal{V})$ with $n = 2$ and $\boldsymbol{x} \in \mathbb{Z}_q^* \times \mathbb{Z}_q^*$, generalized special soundness is consistent with special soundness. If such an argument is also SHVZK it holds that generalized $\Sigma-$protocols are $\Sigma-$protocols in the classical sense.

It is not hard to see that general special soundness implies computational witness extended emulation [Lin03, GI08]. Informally, their definition says that given an adversary that produces an acceptable argument with some probability, there exists an emulator that produces an accepting argument with the same probability and at the same time provides a witness w. Note, that an argument which has witness extended emulation does not need to be zero-knowledge. The definition does not require that the witness stays secrete during the protocol, only that an emulator can produces an accepting argument and extract a witness.

Informally, an argument $(\mathcal{P}, \mathcal{V})$ for relation R is called an *argument of knowledge* if it has witness-extended emulation. As a result a generalized $\Sigma-$protocol is a SHVZK argument of knowledge and this implies that proving that an argument is a $\Sigma-$protocol gives us automatically that the protocol is an argument of knowledge.

Plain model. We will describe all our protocols in the plain model. We will consider the group description and the commitment key as part of the statement, in the case of shuffling the public encryption key will also be part of the statement. In real life zero-knowledge protocols are subprotocols of other zero-knowledge protocols or part of other cryptographic protocols. That means it is reasonable to assume that in such a setting the group description and the public keys are inherited from the outer protocols or can be supplied by the trusted third party that provides the environment.

One important factor that such a set up works is that the group description and the public keys are easily verifiable. We work over abelian modular groups $\mathbb{G} \subset \mathbb{Z}_p^*$ with prime order q, if for q it holds that $q|p-1$, then we can easily check the group description and the commitment key. First we test p, q for primality and then $G \neq 1, \quad G^q \equiv 1 \mod p$, in the case that all checks are accepting we accept the group description. To test that the commitment key ck is valid, we test

$$G_i^q \equiv 1 \mod p \quad \text{for} \quad 1 \leq i \leq q$$

and $G_i \neq 1$, for $i = 1, \ldots, n$ and $H \neq 1$. For shuffling the statement also contains the public encryption

key $pk = \{\mathbb{G}, y\}$. To verify pk we can test if $y \in \mathbb{G}, y^q \equiv 1 \mod p$, and $y \neq 1$.

If the trust in the third party is not reasonable, the generation of the keys can be achieved by adding an extra round of interaction. In this round the verifier generates the groups and public keys, and sends them to the prover. Assuming the prover can verify the validity of the keys our arguments will still be perfect SHVZK.

It is not desirable that the public encryption key is supplied by the verifier, who would automatically know the decryption key. In this case the verifier could decrypt the statement and learn the permutation. However, our argument to prove correctness of a shuffle is still zero-knowledge in this case, since the verifier learns nothing new from the interaction with the prover, as they are able to decrypt the ciphertexts before the protocols.

Another possibility to generate the group description and keys is that they are generated by multi-party protocols of all parties involved. This also adds some extra interaction on top of the protocol. The setup algorithm can also return some side-information that may be used by an adversary; however, we require that even with this side-information the commitment scheme should remain computationally binding. The side-information models that the keys may be set up using some multi-party computation protocol that leaks some information, the adversary may see some decryptions or even learn the decryption key, etc. Our protocols are secure in the presence of such leaks as long as the commitment scheme is computationally binding.

Full Zero-Knowledge. On the one hand it is much easier to design arguments of knowledge with respect to the honest verifier than zero-knowledge arguments with respect to any malicious verifier. On the other hand in real life applications honest verifier zero-knowledge may not suffice since a malicious verifier may give non-random challenges. However, it is easy to convert a SHVZK [CDS94] argument into a full zero-knowledge argument secure against *arbitrary* verifiers. The conversion can be very efficient and only costs a small additional overhead, so we will in this work without loss of generality just focus on building efficient SHVZK arguments.

The OR-proof [CDS94] can be used to transform HVZK arguments of knowledge into real zero-knowledge arguments. The statement can be set up with an additional group element D, and the prover will now use an OR-proof to show that they know a witness for the statement being true or they know the discrete logarithm of D. Since the prover does not know the discrete logarithm of D this is a convincing argument of knowledge. On the other hand the simulator can also simulate this extra argument and therefore, the whole protocol.

If the verifier supplies the group description and the keys, the OR-proof can again be used to convert a HVZK argument following the lines of [FS89]. The verifier calculates $D_1 = G^x \in \mathbb{G}$, picks $D_2 \leftarrow \mathbb{G}$ and sends D_1, D_2 to the prover. The verifier then proves that they know either the discrete logarithm of D_1 or D_2. The prover shows now either knowledge of discrete logarithm of D_1 or D_2, or knowledge of a witness of their statement.

If the whole protocol is set up in the common reference string (CRS) model, standard techniques can be used. They include the technique by Groth [Gro04], which forces the challenges to be uni-

formly random by applying a hash function. Another standard technique on the CRS model is given by [JL00, Dam00]. In their conversion form HVZK into ZK the prover sends in each round an additional commitment, containing the message, to the verifier. If the commitment scheme has a trapdoor, this trapdoor can be used to simulate the protocol for all verifiers.

The OR-proof [CDS94] is also widely used in the CRS model, the common reference string is either set up with an additional group element D or verification key vk of an existentially unforgeable adaptive chosen message attack secure signature scheme [GMY06]. In the first case the prover behaves like the additional group element is part of the statement, and proves knowledge of a witness for the statement or of the discrete logarithm.

In the second case the prover generates a key pair (vk', sk') of a string one-time signature scheme, and reveals vk' to the verifier. Now, the prover shows that either their statement is true or they know a signature for vk' under verification key vk. As the prover cannot know a signature on vk', this convinces the verifier that the prover knows a witness for the statement. Just like above, we can set up the simulator such that it knows the signature and it is easy to simulate the proof.

All these conversions yield arguments of knowledge with perfect zero-knowledge at the price of a couple of extra group elements and are therefore efficient.

3.6 The Schwartz-Zippel Lemma

For completeness we will state a variation of the Schwartz-Zippel lemma that we will use several times.

Lemma 1 (Schwartz-Zippel)**.** *Let P be a non-zero multi-variate polynomial of degree D over $\mathbb{Z}_q$, then the probability of $P(x_1, \ldots, x_n) = 0$ for randomly chosen $x_1, \ldots, x_n \leftarrow \mathbb{Z}_q^*$ is at most $\frac{D}{q-1}$.*

Given two multi-variate polynomials P_1 and P_2 we can test whether

$$P_1(x_1, \ldots, x_n) - P_2(x_1, \ldots, x_n) = 0$$

for random $x_1, \ldots, x_n \leftarrow \mathbb{Z}_q^*$ or not. This equation will always hold if $P_1 = P_2$, whereas if $P_1 \neq P_2$ the probability that the test pass is only $\frac{\max(D_1, D_2)}{q-1}$.

Chapter 4

Implementation Details

To validate the practicality of our main zero knowledge arguments and to check their real life merit, we decided to implement our protocols.

All code is written in C++ because this is a fast, portable, and widely-used programming language. Furthermore, the NTL library provided by Shoup [Sho09] for C++ allows high-performance operation on big integers, which is needed to simulate real life performance. To achieve best possible performance we configured the NTL library with GMP [GMP11] as the primary long integer package.

The NTL library gives the facility to do efficient modular arithmetic on large integers, and also provides a functionality to generate cryptographically strong pseudo-random numbers which are needed by our cryptographic primitives. Building on these functionalities we implemented the generalized Pedersen commitment scheme and ElGamal encryption for subgroups modulo a prime p.

All our code is single threaded. To obtain results, e.g. run time of our protocols and argument size, we run the code on a MacBook Pro with a 2.54 GHz Intel Core 2 Duo CPU, 3 MB 8 way L2 Cache, and 4 GB RAM running Mac OS X 10.7.

4.1 Modular Groups

We decided to test the real life behavior of our protocol on modular subgroups. One reason for this choice is that the discrete logarithm assumption is believed to hold in such groups, see Section 3.2. Another reason for this choice is that modular groups are quite easy to generate, see Section 4.1.2, and operations on them are easy to implement.

4.1.1 Security

In real life the security of the protocols plays an important role. How long a protocol should be secure is different depending on the situation. This demand can be expressed in the security level a protocol should fulfill. Thus, to be able to estimate the real life merit of our protocols we have to use groups with different security levels λ. The security level determines the size of the group order q and the size of the modulus p. Intuitively, the bigger λ the bigger are q and p.

The European Network of Excellence in Cryptology II (ECRYPT II) [oEiCI12] suggests a 160-bit subgroup modulo a $1\,248$-bit prime with a security level of 80-bits to have only short term protection, which means this security level can protect data only until 2017. Whereas a 256-bit subgroup modulo a

3 248-bit prime has a security level of 128-bits and therefore offers according to ECRYPT II medium to long term protection. This should guarantee security of the data over the next 20-30 years. To achieve real long term protection ECRYPT II suggests to use groups with higher security level, for instance a 384-bit subgroup modulo a 7 936-bit prime with a security level of 192-bits.

We decided to test our protocols on groups with the suggested security level. Furthermore, to test the influence of different parameters, e.g. group size or modulo size, we decided to use also groups which are not recommended by ECRYPT II and are not standard groups in research community.

We picked subgroups of 160-bit modulo a 1 024-bit prime, a 1 248-bit prime, a 1 536-bit prime, and a 3 248-bit prime. We also decided to use a 256-bit subgroup modulo a 1 536-bit prime, 2 432-bit prime, and a 3 248-bit prime. Lastly we look at subgroups of the size 384-bit modulo a 3 248-bit prime, and a 7 936-bit prime. Three of these groups have the parameter sizes recommended by ECRYPT II the rest are non-standard and we have to estimate the security level.

To estimate the security level of these groups we refer to Lenstra and Verheul [LV01] and Lenstra [Len04]. The security of a modular subgroup depends on the one hand on the ability to break the discrete logarithm on the group $\mathbb{G}$ itself. Lenstra [Len04] suggests choosing the size q such that the Pollard's ρ method cannot be successful until a desired year. On the other hand the security also depends on the ability to break the discrete logarithm in $\mathbb{Z}_p^*$, where p is prime and $\mathbb{G} \subset \mathbb{Z}_p^*$. That means p should be chosen big enough that no method, for instance the number sieve, can break the discrete logarithm in $\mathbb{Z}_p^*$ and at the same time find the discrete logarithm in $\mathbb{G}$.

Following the recommendations of Lenstra we conclude that a subgroup of size 160-bit has a security level of 80-bits, a subgroup of 256-bit a security level of 128-bits, and of size 384-bit a security level 192-bits respectively. So for bigger parameters q and p we get higher security.

For the moduli values we estimate that a 1 024-bit prime equals a security level of 72-bits and a 1 248-bit prime a security level of 78-bits, for all other security levels see Table 4.1.[1]

$\lvert p\rvert$	1 024	1 248	1 536	2 432	3 248	7 936
security level	72	78	85	102	114	161

Table 4.1: Estimated security level for primes p with different bit-sizes.

These values can be interpreted to mean that a subgroup with a 160-bit order modulo a 1 024-bit prime has a security level between 72 to 80-bits and a 160-bit subgroup modulo a 1 248-bit prime has a security level between 78 to 80-bits. The exact value is hard to identify, but it is not so important for our work. Important to know for our work is that for fixed group size the size of the modulo value determines the security level of the group. An increase of the moduli results in an increase of the security level. One has to be careful, though, and choose p and q in a way such that they represent a similar security level.

During our work we will use following rules. First, the security level of a modular group increases if the subgroup size stays fixed and the modular value gets bigger. Secondly, for fixed moduli sized the security level increases by nearly the same factor as the increase of the subgroup size. Lastly, bigger

[1] The numbers are estimated using www.keylength.com/en/2/

subgroup and moduli sizes results in higher security level than small values.

A few more words on the security and real life usability of our chosen groups. As seen above a 160-bit subgroup modulo a 1 024-bit prime offers no real security; however, we are using this group to compare our shuffle argument with former implementations. Moreover, 160-bit subgroups modulo 1 248-bit or 1 536-bit primes are also very vulnerable and offers no long term protection. However, we wanted to have a wide range of groups to test our implementation on. For the same reason, we picked 160-bit subgroups modulo primes with 2 432 or 3 248-bit. These groups are not practical in real life, both moduli values offers medium to long term protection which is paid by the increased size of the group elements and therefore expensive operations. But, this security is compromised by the short term security of the small group order. So, these groups are of theoretical interest only. The same holds for groups which have order of 256-bit length modulo 1 536 or 2 432-bit primes, or order of 384-bit length modulo a 3 248 bit prime; in this case the subgroup offers a higher security than the group $\mathbb{Z}_p^*$. The only groups which might be used in real-life applications are subgroups with 256-bit order modulo a 3 248-bit prime which offers medium to long term protection. Or subgroups with 384-bit prime order modulo a 7 984-bit prime, which offer long term protection.

In other words, we have chosen on purpose groups which are not practical in real life; however, the reason for this was to have a wide range of groups to test different parameters of our implementation.

4.1.2 Group Generation

To generate the modular groups and generators we decided to use a method outlined by Maurer [Mau95]. This method allows us to generate primes q and p, such that $q|p-1$ and there exists a group $\mathbb{G}$ with order q and $\mathbb{G} \subset \mathbb{Z}_p^*$.

For some optimizations, i.e. the Fast Fourier Transform, it is necessary to find a k-th root of unity. Our generation algorithm gives us enough control to generate q such that a k-th root of unity exist in $\mathbb{Z}_q$ and at the same time it is possible to find such a root of unity using only a few operations.

Maurer's idea builds on the following lemma, which is a special case of a theorem by Pocklington [Poc14].

Lemma 2 ([Mau95]). *Let $n = 2RF + 1$ and the prime factorization of F is $F = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_r^{\alpha_r}$. If there exists an integer a satisfying*

$$a^{n-1} \equiv 1 \mod n$$

and

$$gcd(a^{n-1}/p_i, n) = 1$$

for $i = 1, \dots, r$, then each prime factor p of n is of the form $p = mF + 1$ for some integer $m \geq 1$. Moreover, if $F > \sqrt{n}$, or F is odd and $F > R$, then n is prime.

The first step is to generate the prime order q of $\mathbb{G}$. A requirement for a k-th root of unity to exist in $\mathbb{Z}_q$ is that $k|q-1$; therefore, we chose $F = k \cdot p_1$, where p_1 is some random prime. To fulfill the lemma we have to choose our values in the right way. Assume q should be b_q bits long, and k is b_k bits long,

then we have $b_q - b_k$ bits to spare for p_1 and R. We can therefore chose p_1 and R to be primes which are $\frac{b_q - b_k}{2}$ bits long. Then, $F = kp_1 > \sqrt{n}$ and $n = 2RF + 1$ can be prime according to the lemma.

Next, we can chose random a and test the two condition of the lemma. To make certain that the algorithm terminates we test a small number of a's, if none of them fulfills the condition we pick new values for p_1 and R. In the case of succeeding a we have found a prime $q = 2kp_1R$, since all the conditions of the lemma are satisfied.

Next, we have to check if a is really a generator of $\mathbb{Z}_q$. Maurer tells us that this can be done by checking the equation

$$a^{\frac{n-1}{s}} \not\equiv 1 \mod n$$

for all prime factors s of R. As our R is prime we have to check this equation only once and if we succeed we know a is a generator of $\mathbb{Z}_q$.

A k-th root of unity can now easily obtained by setting $\omega = a^{\frac{q-1}{k}}$, which only costs one single exponentiation and one division.

After we have found our q we have to generate a prime p such that $q|p-1$. We can use the same approach as before. We pick a random prime p_1 which is $b_p - b_q$ bits long, where b_p is the required number of bits for p. If $p_1 > q$ we set in the lemma above $F = p_1$, else $F = q$, and test the condition for random numbers a. Again, if an a passes the test, we have found a prime number p.

Finally, we have to find a generator G of the group $\mathbb{G}$. The easiest way to find it is to take a generator of $\mathbb{Z}_p^*$ to the power $\frac{p-1}{2p_1}$. Thus, we have to check if our a is really a generator, which can be done according the same method as before. In the case a not a generator we will pick a new value p_1, otherwise we calculate G.

The final output of this algorithm are primes q and p, such that $q|p$, a generator G_q for $\mathbb{Z}_q$, a k-th root of unity ω in this field, and a generator $G \in \mathbb{Z}_q$ of the subgroup $\mathbb{G}$ with order q.

4.2 Optimizing Multiplications

During our protocol many multiplications take part, for some tasks the cost of the multiplication can be really expensive and dominant; thus, we have to optimize these tasks.

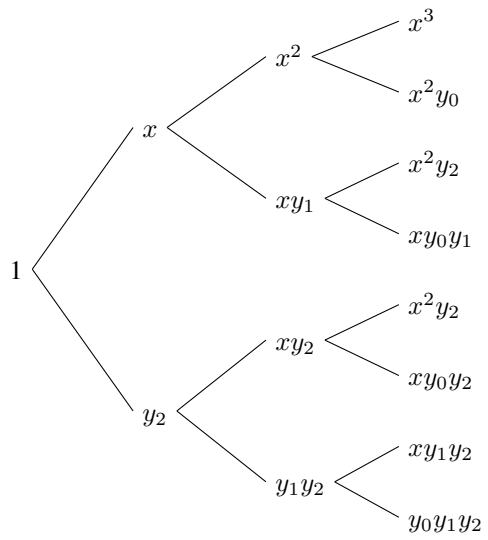
First, in the setting of the polynomial evaluation argument and the blacklist argument we have to multiply many big polynomials. The best asymptotic way in literature to solve this problem is to use the Fast Fourier Transform (FFT) [CT65]. The multiplication of two degree D polynomials costs $\Omega(D \log D)$ multiplications. For smaller degree polynomials other techniques give better performance, but to keep the implementation simple we only coded the FFT. For the implementation we used the classic version relying on loops.

The other problem, which leads to dominant computational cost, is to calculate sums like

$$\sum_{i_0,\ldots,i_d=0}^{i} a_{i_0,\ldots,i_d} \prod_{j=0}^{d} y_j^{i_j} x^{1-i_j}$$

where x, y_j can be single values, or vectors, or some function. Calculating all $\prod_{j=0}^{d} y_j^{i_j} x^{1-i_j}$ from

scratch can be quite expensive. But it is possible to reduce the number of multiplications needed. The idea is to use a binary tree of depth $d+1$ to generate all possible products $\prod_{j=0}^{d} y_j^{i_j} x^{1-i_j}$. The following tree for $d = 2$ illustrates the exact steps of a binry tree.



4.3 Multi-exponentiation Techniques

To calculate a commitment to a vector a we have to compute the product

$$\mathrm{com}_{ck}(a_1, \ldots, a_n; r) = H^r \prod_{i=1}^{n} G_i^{a_i},$$

this costs $n+1$ single exponentiations if done in a naïve way. Thus, the cost to calculate the commitments can dominant the run-time for big n or for many commitments. To optimize the calculation of such commitments we decided to implement multi-exponentiation techniques.

We picked to use Brickell et al.'s precomputation algorithm [BGMW98] for large n as described by Lim [Lim00]. This algorithm is used for all occurrences of multi-exponentiation equations with large n. To optimize also all other multi-exponentiation equations we decided to implement Lim-Lee's algorithm [Lim00] for $n \leq 1\,000$ and for $n = 2$ we choose the sliding window algorithm [Lim00].

For notational reasons we will describe our techniques for the slightly modified equation

$$\prod_{i=0}^{n} G_i^{a_i} \tag{4.1}$$

4.3.1 Sliding Window Algorithm

The sliding window algorithm gives the best performance for very small n, that means n is between 2 and 4. The idea is similar to the square-and-multiply algorithm to calculate $y = x^e$ the power of an integer. In the square-and-multiply algorithm we write the exponent e in binary and set first $y = x$. Then, we look at the second most significant bit. If this bit is 0 we assign $y = y^2$, otherwise we have $y = y^2 x$. Repeating these operations for all bits of the exponent, the algorithm calculates x^e. This method has an

obvious generalization instead of using the base 2 we expand the exponent e in base 2^w. More precisely, in each step we look at w bits of the binary expression of the exponent. The number of bits is called the window size of the algorithm.

Instead of using a fixed window size as above, e.g. 1 if we take the square-and-multiply algorithm, we use windows with variable sizes for the sliding window technique. This means, for each exponent we look at a different number of bits, which can be between 1 and a maximum size of w. To find the windows we slide from the most significant bit to the least significant bit and search for non-zero bit strings which are smaller or equal to the window size w and end with 1.

In more detail, we can express each exponent a_i as

$$a_i = \sum_{j=1}^{k_i} a_{i,j} 2^{l_{i,j}},$$

where $0 < a_{i,j} < 2^w$, $a_{i,j}$ is odd, w the window size and k_i the number of bit strings. We can define

$$G_{i,j} = G_i^{2j-1},$$

for $1 < j < 2^{w-1}$ and plug these precomputed values in our equation. This gives us

$$\prod_{i=0}^{n} G_i^{a_i} = \prod_{i=0}^{n} \left(\prod_{j=0}^{k_i - 1} G_i^{a_{i,j} 2^{l_{i,j}}} \right) = \prod_{i=0}^{n} \left(G_{i,(a_{i,j}+1)/2}^{2^{l_{i,j}}} \right).$$

To use this technique we can now pre-compute values $G_{i,j}$ and calculate the right hand side of the equation using a square-and-multiply algorithm.

We will mainly use the sliding window algorithm to commit to single values during the protocols, in this case the basis G_0 and G_1 are fixed by the commitment key throughout the protocol. For this reason and to speed up our implementation further we pre-compute values $G_{i,j}$ in the beginning and reuse the values for all commitments.

Window Size: Lim suggested a window size $w = 4$ for exponents $a_i \in \mathbb{Z}_q$ with $80 < |q| \leq 240$ and for $240 < |q| \leq 672$ a window size $w = 5$. To ensure that these recommendations are consistent with our implementation we tested different window sizes to find the optimal one.

To determine the value of w, we run the prover of Brands et al.'s polynomial argument, described in Section 6.5. The dominant part of this prover is the calculation of many commitments to single value and no other multi-exponentiations are calculated. For big degree D polynomials, the complete run-time is slow enough to see the merit of different window sizes. Table 4.2 shows the run-time for the prover for a degree $D = 100\,000$ polynomial with window sizes $w = 4, 5, 6, 7$. We used a subgroup of 160-bit prime modulo a 1 248-bit prime and subgroups with a 256-bit and 384-bit order q modulo a 3 248-bit prime.

We see that for $|q| = 160$ we get best performance for $w = 5$ and for both other q the best performance is given for the window size $\omega = 6$. These values differ slightly from the recommendation, but as

$\|q\|$	$\|p\|$	$w = 4$	$w = 5$	$w = 6$	$w = 7$
160	1 248	1 306 ms	**1 278** ms	1 325 ms	1 408 ms
256	3 248	8 607 ms	8 433 ms	**8 369** ms	8 757 ms
384	3 248	12 756 ms	12 194 ms	**12 075** ms	12 292 ms

Table 4.2: Run-time in ms of Brands et al.'s polynomial argument verifier for different window sizes for the sliding window algorithm and different modular groups for a polynomial with degree $D = 100\,000$.

they are optimal for our implementation we will stick to these values throughout the data collection.

4.3.2 Lim-Lee's Pre-computation Technique

For medium range values n we decided to use Lim-Lee's pre-computation technique, since its performance is better than the sliding window technique and Brickell et al.'s technique.

The idea behind this algorithm is that instead of calculating first all bases to the power of the exponent and the multiplying the result together, we can calculate the product bit-wise for the exponent. To speed up this technique, the product can be split into different blocks, the values of which can be calculated beforehand.

In more detail, we can write each exponent a_i in binary as $a_i = \sum_{i=0}^{t-1} a_{i,j} 2^j$ and plug this expression in the original equation 4.1. This gives us

$$\prod_{i=1}^{n} G_i^{a_i} = \prod_{i=1}^{n} G_i^{\sum_{j=0}^{t-1} a_{i,j} 2^j} = \prod_{j=0}^{t-1} \left(\prod_{i=1}^{n} G_i^{a_{i,j}} \right)^{2^j},$$

we can now rearrange the inner products to get $h = \lceil \frac{n}{w} \rceil$ smaller products

$$\prod_{j=0}^{t-1} \left(\prod_{i=1}^{n} G_i^{a_{i,j}} \right)^{2^j} = \prod_{j=0}^{t-1} \left(\prod_{k=1}^{h-1} \left(\prod_{i=kw}^{kw+w-1} G_i^{a_{i,j}} \right) \right)^{2^j}.$$

The innermost products can be precomputed as

$$G_{k,e} = \prod_{j=0}^{w-1} G_{kw+j}^{e_i}$$

for each $k = 0, \ldots, h-1$, where $0 < e = e_{w-1} \ldots e_1 e_0 2^w$ and $e_i \in \{0, 1\}$. Entering these values in our equation give us

$$\prod_{i=1}^{n} G_i^{a_i} = \prod_{j=0}^{t-1} \left(\prod_{k=0}^{h-1} G_{k,e(k)} \right)^{2^j},$$

where $e(k)$ is defined as

$$e(k) = a_{kw+w-1,j} \ldots a_{kw+1,j} a_{kw,j}.$$

The algorithm works now as follows. First we pre-compute all possible values $G_{k,e(k)}$, then the inner products and in the end we use a square-and-multiply algorithm to calculate the outer product.

Window Size: Similar to the window size algorithm the performance of the method depend on the window size w. Lim suggests a window size of $w = 6$ for our range of parameter. Again, we tried different values to determine the best window size for our implementation.

In this case we used the performance of Brands et al's blacklist verifier, see Section 7.5. In the un-optimized version this verifier has to calculate n multi-exponentiations to vectors of length n. This cost is dominant; thus, a good point to test the merit of different window sizes.

$D\backslash w$	3	4	5	6
10	8.35 ms	**8.25** ms	8.62 ms	9.10 ms
100	28.23 ms	27.29 ms	**26.70** ms	29.93 ms
1 000	160 ms	**147** ms	154 ms	174 ms

Table 4.3: Run-time in ms of Brands et al.'s blacklist argument verifier for different window sizes for Lim-Lee's algorithm on a group $\mathbb{G}$ with order 160-bit modulo a 1 248-bit prime and polynomials with degrees $D = 10, 100, 1\,000$.

$D\backslash w$	3	4	5	6
10	64 ms	**62** ms	64 ms	67 ms
100	217 ms	213 ms	**200** ms	216 ms
1 000	1 228 ms	1 107 ms	**1098** ms	1 168 ms

Table 4.4: Run-time in ms of Brands et al.'s blacklist argument verifier for different window sizes for Lim-Lee's algorithm on a group $\mathbb{G}$ with order 256-bit modulo a 3 248-bit prime and polynomials with degrees $D = 10, 100, 1\,000$.

$D\backslash w$	3	4	5	6
10	94 ms	93 ms	**92** ms	97 ms
100	318 ms	314 ms	**291** ms	310 ms
1 000	1 861 ms	1 643 ms	**1 596** ms	1 670 ms

Table 4.5: Run-time in ms of Brands et al.'s blacklist argument verifier for different window sizes for Lim-Lee's algorithm on a group $\mathbb{G}$ with order 384-bit modulo a 3 248-bit prime and polynomials with degrees $D = 10, 100, 1\,000$.

Tables 4.3, 4.4, 4.5 state the results for different window sizes $w = 3, 4, 5, 6$, and different values $n = 10, 100, 1\,000$. Each table shows the data for one of the three different groups $\mathbb{G}$ with $|q| = 160, 256, 384$. We see that different to the sliding window algorithm no window size gives us optimal performance for the subgroups with $|q| = 160, 256$; however, the different between the run-time for $w = 4, 5$ is very small. In the case of $|q| = 384$ window size $w = 5$ gives the best performance and therefore we sticked throughout the data collection to the window size $w = 5$.

4.3.3 Brickell et al.'s Pre-computation Algorithm

The last multi-exponentiation technique we implemented is Brickell et al.'s pre-computation algorithm, this method performs best if n is large.

The idea here is to represent each exponent a_i in base 2^w, where w is some fixed integer. This gives

us

$$a_i = \sum_{j=0}^{h-1} a_{i,j} 2^{jw}$$

with $h = \left\lceil \frac{|q|}{w} \right\rceil$ and $0 \leq a_{i,j} < 2^w$. Putting these expression in equation 4.1 gives us

$$\begin{aligned} \prod_{i=0}^{n} G_i^{a_i} &= \prod_{i=0}^{n} \left(\prod_{j=0}^{h-1} G_i^{a_{i,j} 2^{jw}} \right) \\ &= \prod_{j=0}^{h-1} \left(\prod_{i=0}^{n} G_i^{a_{i,j}} \right)^{2^{jw}} . \end{aligned}$$

The first step is to pre-compute the inner products and then multiply the precomputed values in a square-and multiply manner.

Window Size: The run-time of the algorithm depends on the window size w which depends on the group size and the length of n. Lim gives a way to calculate these values theoretically, but as the value also depends on the implementation we decided to find the best values in experiments.

To determine the best windows sizes we use again Brands et al.'s blacklist verifier without batching. In this case we tested different values w for a subgroup with order 160-bit modulo a prime with 1 248-bit, and subgroups $\mathbb{G}$ with a 256-bit and 384-bit order q modulo a 3 248-bit prime. For the vector length n we used the values $10\,000, 100\,000$ to see the different in the run-time. The results can be found in Table 4.6, 4.7, and 4.8.

$D \backslash w$	6	7	8
10 000	1 129 ms	**1 124** ms	1 143 ms
100 000	10 153 ms	**10 127** ms	10 183 ms

Table 4.6: Run-time in ms of Brands et al.'s blacklist argument verifier for different window sizes for Brickell et al.'s algorithm on a group $\mathbb{G}$ with order 160-bit modulo a 1 536-bit prime and polynomials with degrees $D = 10\,000, 100\,000$.

$D \backslash$ w	6	7	8
10 000	7 527 ms	**7 361** ms	7 451 ms
100 000	66 094 ms	64 550 ms	**64 545** ms

Table 4.7: Run-time in ms of Brands et al.'s blacklist argument verifier for different window sizes for Brickell et al.'s algorithm on a group $\mathbb{G}$ with order 256-bit modulo a 3 248-bit prime and polynomials with degrees $D = 10\,000, 100\,000$.

$D \backslash w$	6	7	8
10 000	10 822 ms	**10 795** ms	10 901 ms
100, 000	94 211 ms	**93 469** ms	93 923 ms

Table 4.8: Run-time in ms of Brands et al.'s blacklist argument verifier for different window sizes for Brickell et al.'s algorithm on a group $\mathbb{G}$ with order 384-bit modulo a 3 248-bit prime and polynomials with degrees $D = 10\,000, 100\,000$.

We see that in all cases a window size of $w = 7$ gives the best performance, besides of the case $n = 100\,000$ and $|q| = 256$. Anyway, the difference between the run-time for $w = 7$ and $w = 8$ is very small. Thus, we decided to use $w = 7$ for all our data collection in this work. This value differs much from the recommended values, however this gives us the best performance.

Chapter 5

Basic Protocols

In this chapter we will state and explain zero-knowledge arguments for different problems, which are needed as building blocks for the practical zero-knowledge arguments in the later chapters. However the arguments are also of interest as standalone protocols. For instance, such a problem can be for given $a, b, c \in \mathbb{Z}_q$ to show $a \cdot b = c$ without revealing a, b, c, see Section 5.1, this problem appears as subprotocol in various zero-knowledge arguments. The original technique is due to Chaum and Pedersen [CP92], we adjusted the technique such that the argument is a Σ-protocol. In Section 5.3 we describe a product argument to show for secret values a_i and b, that $\prod_{i=1}^{N} a_i = b$. The whole section is based on Groth's linear algebra techniques [Gro09] and was published at Eurocrypt 2012 [BG12]. Lastly, in Section 5.4, we will show that for committed vectors $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ it holds $\boldsymbol{a}_i \circ \boldsymbol{b}_i = \boldsymbol{c}_i$. This argument was not published before and is joint work with Jens Groth. All arguments described in this chapter are efficient themselves and can be used as building blocks for the practical zero-knowledge arguments in the later chapters.

5.1 Simple Product Argument

We will now give an argument how to prove the correctness of a product $a \cdot b = c \in \mathbb{Z}_q$, for committed values $a, b, c \in \mathbb{Z}_q$. The protocol is based on the techniques in [CP92], [CD98], and [NBMV99].

Statement: $\{\mathbb{G}, p, q\},\ ck,\ c_a, c_b, c_c \in \mathbb{G}$

Prover's witness: $a, b, c, r, s, t \in \mathbb{Z}_q$ such that

$$a \cdot b = c \qquad c_a = \mathrm{com}_{ck}(a; r) \qquad c_b = \mathrm{com}_{ck}(b; s) \qquad c_c = \mathrm{com}_{ck}(c; t).$$

Initial message: Compute

1. $c_d = \mathrm{com}_{ck}(d, \rho)$ where $d, \rho \leftarrow \mathbb{Z}_q$
2. $c_e = \mathrm{com}_{ck}(e, \sigma)$ where $e, \sigma \leftarrow \mathbb{Z}_q$
3. $c_f = c_b^d \mathrm{com}_{ck}(0; \tau)$ where $\tau \leftarrow \mathbb{Z}_q$

Send: c_d, c_e, c_f

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Calculate

1. $\overline{a} = d + xa \qquad \overline{r} = \rho + xr$
2. $\overline{b} = e + xb \qquad \overline{s} = \sigma + xs$
3. $\overline{t} = \tau - x(sa - t)$

Send: $\overline{a}, \overline{r}, \overline{b}, \overline{s}, \overline{t}$

Verification: Accept the argument if and only if

1. $c_d, c_e, c_f \in \mathbb{G}$
2. $\overline{a}, \overline{r}, \overline{b}, \overline{s}, \overline{t} \in \mathbb{Z}_q$
3. $c_a^x c_d = \mathrm{com}_{ck}(\overline{a}; \overline{r}) \qquad c_b^x c_e = \mathrm{com}_{ck}(\overline{b}; \overline{s})$
4. $c_c^x c_f = c_b^{\overline{a}} \mathrm{com}_{ck}(0; \overline{t})$

Theorem 17. *The protocol is a 3-round public-coin perfect $\Sigma-$protocol of committed values a, b, c such that $a \cdot b = c$.*

Proof. Perfect completeness can be seen by careful inspection of the verification equations.

Given challenge $x \leftarrow \mathbb{Z}_q^*$ the simulator picks $\overline{a}, \overline{r}, \overline{b}, \overline{s}, \overline{t} \leftarrow \mathbb{Z}_q$, and sets

$$c_d = c_a^{-x} \mathrm{com}_{ck}(\overline{a}; \overline{r}) \qquad c_e = c_b^{-x} \mathrm{com}_{ck}(\overline{b}; \overline{s}) \qquad c_f = c_c^{-x} c_b^{\overline{a}} \mathrm{com}_{ck}(0, \overline{t}).$$

The answers $\overline{a}, \overline{r}, \overline{b}, \overline{s}, \overline{t}$ are uniformly random in the real argument and also in the simulated argument. The unconditionally hiding property of the commitment schemes gives us that c_d, c_e, c_f follow the same distribution as in the real argument. Therefore, the protocol is SHVZK.

It remains to show that the protocol has perfect special soundness. Given two transcripts $\left(c_a, c_b, x_1, \overline{a}^{(1)}, \overline{r}^{(1)}, \overline{b}^{(1)}, \overline{s}^{(1)}, \overline{t}^{(1)}\right)$, $\left(c_a, c_b, x_2, \overline{a}^{(2)}, \overline{r}^{(2)}, \overline{b}^{(2)}, \overline{r}^{(2)}, \overline{t}^{(2)}\right)$ with $x_1 \neq x_2$, the extractor gets openings a, b, c, r, s, t by taking linear equations of the verifications. More precisely, we have the two answers satisfy

$$c_a^{x_1} c_d = \mathrm{com}_{ck}\left(\overline{a}^{(1)}; \overline{r}^{(1)}\right) \qquad c_2^{x_2} c_d = \mathrm{com}_{ck}\left(\overline{a}^{(2)}; \overline{r}^{(2)}\right).$$

Picking α_1, α_2 such that $\alpha_1 x_1 + \alpha_2 x_2 = 1$ and $\alpha_1 + \alpha_2 = 0$ gives us

$$c_a = c_a^{\alpha_1 x_1 + \alpha_2 x_2} c_d^{\alpha_1 + \alpha_2} = \mathrm{com}_{ck}(\alpha_1 \overline{a}^{(1)} + \alpha_2 \overline{a}^{(2)}; \alpha_1 \overline{r}^{(1)} + \alpha_2 \overline{r}^{(2)}).$$

Similar we get

$$c_b = \mathrm{com}_{ck}(\alpha_1 \overline{b}^{(1)} + \alpha_2 \overline{b}^{(2)}; \alpha_1 \overline{s}^{(1)} + \alpha_2 \overline{s}^{(2)}).$$

To extract c we take linear equations over

$$c_c^x c_f = c_b^{\overline{a}} \mathrm{com}_{ck}(0; \overline{t}),$$

using the same α_1, α_2 as before we get

$$\begin{aligned}
c_c &= c_c^{\alpha_1 x_1 + \alpha_2^x 2} c_f^{\alpha_1 + \alpha_2} \\
&= c_b^{\alpha_1 \overline{a}^{(1)} + \alpha_2 \overline{a}^{(2)}} \mathrm{com}_{ck}(0, \overline{t}^{(1)})^{\alpha_1} \mathrm{com}_{ck}(0; \overline{t}^{(2)})^{\alpha_2} \\
&= \mathrm{com}_{ck}\Big(\alpha_1^2 \overline{a}^{(1)} \overline{b}^{(1)} + \alpha_1 \alpha_2 (\overline{a}^{(1)} \overline{b}^{(2)} + \overline{a}^{(2)} \overline{b}^{(1)}) + \alpha_2^2 \overline{a}^{(2)} \overline{b}^{(2)}; \\
&\qquad \alpha_1^2 \overline{a}^{(1)} \overline{s}^{(1)} + \alpha_1 \alpha_2 (\overline{a}^{(1)} \overline{s}^{(2)} + \overline{a}^{(2)} \overline{s}^{(1)}) + \alpha_2^2 \overline{a}^{(2)} \overline{s}^{(2)} + \alpha_1 \overline{t}^{(1)} + \alpha_2 \overline{t}^{(2)} \Big)
\end{aligned}$$

Lastly, we have to argue that the extracted openings satisfy the statement. The commitment c_f contains $xc + bd - xab$ for random x by the binding property of the commitment scheme. Since c_f is fixed before the prover sees the challenge x, this implies that $c = ab$, and a, b, c are the same values as known by the prover. If this is not the case, the extractor could be used by the prover to find a second opening to their their commitments and the commitment scheme is broken.

We can conclude that the argument is a $\Sigma-$protocol. □

Efficiency: During the whole protocol 3 group elements and 5 field elements are transferred between the prover and verifier.

The prover has to calculate 6 exponentiations and the verifier 9 exponentiations in $\mathbb{G}$. The total number of multiplications are 6 for the prover and 4 for the verifier.

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 149 \rangle \subset \mathbb{Z}_{179}^*$, which has prime order $q = 89$.

The statement consists of $\{\mathbb{G}, \{p, q\} = \{\langle 149 \rangle, 179, 89\}$, $ck = \{G, H\} = \{149, 129\}$, $c_a = 144, c_b = 155, c_c = 77 \in \mathbb{G}$, and the claim that $a \cdot b = c$.

The prover knows $a = 88$, $b = 47$, and $c = a \cdot b = 42 \in \mathbb{Z}_{89}$, and $r = 13, s = 25, t = 67 \in \mathbb{Z}_{89}$ such that.

$$c_a = \mathrm{com}_{ck}(a; r) = 149^{88} \cdot 129^3 = 74 \in \mathbb{G} \quad c_b = \mathrm{com}_{ck}(b; s) = 155 \in \mathbb{G} \quad c_c = \mathrm{com}_{ck}(c; t) = 77 \in \mathbb{G}.$$

To prove knowledge of witnesses a, b the prover picks $d = 34, \rho = 36, e = 3, \sigma = 15$, computes

$$c_d = \mathrm{com}_{ck}(d; \rho) = 47 \qquad c_e = \mathrm{com}_{ck}(3, 15) = 177 \qquad c_f = c_b^d \cdot \mathrm{com}_{ck}(0; \rho) = 36,$$

and sends

$$c_d = 47 \quad c_e = 177 \quad c_f = 36$$

to the verifier.

The verifier picks challenge $x = 26 \in \mathbb{Z}_q^*$ and gives this value to the prover.

To answer the challenge the prover calculates

$$\overline{a} = d + xa = 34 + 26 \cdot 88 = 8 \qquad \overline{r} = \rho + xr = 36 + 26 \cdot 13 = 18$$
$$\overline{b} = e + xb = 3 + 26 \cdot 47 = 68 \qquad \overline{s} = \sigma + xs = 15 + 26 \cdot 25 = 42$$

$$\overline{t} = \rho - x(sa - t) = 36 - 26(25 \cdot 88 - 67) = 25$$

and send these values to the verifier.

The verifier checks if $c_d, c_e, c_f \in \mathbb{G}$, $\overline{a}, \overline{r}, \overline{b}, \overline{s}\overline{t} \in \mathbb{Z}_q$, and finally

$$c_a^x c_d = 173 = \mathrm{com}_{ck}(\overline{a}; \overline{r}) \quad (\checkmark)$$
$$c_b^x c_e = 161 = \mathrm{com}_{ck}(\overline{b}; \overline{s}) \quad (\checkmark)$$
$$c_c^x c_f = 60 = c_b^{\overline{a}} \mathrm{com}_{ck}(0, \overline{t}) \quad (\checkmark)$$

Thus, the verifier is convinced that the prover knows a, b, c such that $a \cdot b = c$.

5.2 Invertible Argument

We will now give an argument how to prove that an element $a \in \mathbb{Z}_q^*$ is invertible, that means $a \neq 0$. The protocol is based on the techniques in [Bra97] and [BDD07].

Statement: $\{\mathbb{G}, p, q\},\ ck,\ c_a \in \mathbb{G}$

Prover's witness: $a, r \in \mathbb{Z}_q$ such that $c_a = \mathrm{com}_{ck}(a; r) \quad a \neq 0$.

Initial message: Compute

1. $c_b = c_a^s \mathrm{com}_{ck}(0; -t)$ where $s, t \leftarrow \mathbb{Z}_q$

Send: c_b

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Calculate

1. $\overline{a} = s + xa^{-1} \qquad \overline{r} = t + xra^{-1}$

Send: $\overline{a}, \overline{r}$

Verification: Accept the argument if and only if

1. $c_b \in \mathbb{G}$
2. $\overline{a}, \overline{r} \in \mathbb{Z}_q$
3. $c_b = c_a^{\overline{a}} \mathrm{com}_{ck}(-x; \overline{r})$

Theorem 18. *The protocol is a 3-round public-coin perfect $\Sigma-$protocol of committed values a, such that $a \neq 0$.*

Proof. Perfect completeness can be seen by careful inspection of the verification equations.

Given challenge $x \leftarrow \mathbb{Z}_q^*$ the simulator picks $\overline{a}, \overline{r} \leftarrow \mathbb{Z}_q$, and sets $c_b = c_a^{\overline{a}} \mathrm{com}_{ck}(-x; \overline{r})$.

The answers $\overline{a}, \overline{r}$ are uniformly random in the real argument and also in the simulated argument. The unconditionally hiding property of the commitment schemes gives us that c_b follows the same distribution as in the real argument. Therefore, the protocol is SHVZK.

It remains to show that the protocol has perfect special soundness. Given two transcripts $\left(c_a, x_1, \overline{a}^{(1)}, \overline{r}^{(1)}\right)$, $\left(c_a, x_2, \overline{a}^{(2)}, \overline{r}^{(2)}\right)$ with $x_1 \neq x_2$, the extractor gets openings a, r, s, t using the Vandermonde matrix

$$M = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix}.$$

Since $x_1 \neq x_2$ the matrix M is invertible, multiplying $(s, a^{-1})^T$ with M^{-1} gives us openings s, a^{-1} and also a. In a similar way we can extract openings r, t.

Lastly, we have to argue that the extracted openings satisfy the statement. The commitment c_b contains $sa + xaa{-}1 - x$ for random x by the binding property of the commitment scheme. Since c_b is fixed before the prover sees the challenge x, this implies that a^{-1} is the inverse of a and therefore it follws that $a \neq 0$. Furthermore, a is the same value as known by the prover, if this is not the case, the extractor could be used by the prover to find a second opening to one of the commitments and the commitment scheme is broken.

We can conclude that the argument is a $\Sigma-$protocol. □

Efficiency: During the whole protocol 1 group elements and 2 field elements are transfered between the prover and verifier.

The prover has to calculate 2 exponentiations and the verifier 3 exponentiations in $\mathbb{G}$. The total number of multiplications are 4 for the prover and 3 for the verifier.

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 149 \rangle \subset \mathbb{Z}^*_{179}$, which has prime order $q = 89$.

The statement consists of $\{\mathbb{G}, \{p, q\} = \{\langle 149 \rangle, 179, 89\}$, $ck = \{G, H\} = \{149, 129\}$, $c_a = 144 \in \mathbb{G}$, and the claim that $a \neq 0$.

The prover knows $a = 56, r = 13 \in \mathbb{Z}_{89}$ such that $c_a = \mathrm{com}_{ck}(a; r)144 \in \mathbb{G}$.

To prove knowledge of witnesses a the prover picks $s = 34, t = 36$, computes

$$c_b = c_a^s \mathrm{com}_{ck}(0; -t) = 82$$

and sends $c_b = 82$ to the verifier.

The verifier picks challenge $x = 26 \in \mathbb{Z}^*_q$ and gives this value to the prover.

To answer the challenge the prover calculates

$$\overline{a} = s + xa^{-1} = 34 + 26 \cdot 62 = 44 \qquad \overline{r} = t + xra^{-1} = 36 + 26 \cdot 13 \cdot 62 = 77$$

and send these values to the verifier.

The verifier checks if $c_b \in \mathbb{G}$, $\overline{a}, \overline{r} \in \mathbb{Z}_q$, and finally

$$c_b = 82 = c_a^{\overline{a}} \mathrm{com}_{ck}(-x; \overline{r}) \quad (\checkmark)$$

Thus, the verifier is convinced that the prover knows a such that $a \neq 0$.

5.3 Product Argument

The whole section was published at Eurocrypt 2012 in [BG12]. This argument was also illustrated in Groth [Gro09] but we will give all the details and make a few minor improvements. Section 5.3.3 is based on work in [Gro10].

We will now describe an argument that a set of committed values have a particular product. More precisely, given commitments $\boldsymbol{c}_A$ to $A = \{a_{ij}\}_{i,j=1}^{n,m}$, and a value b we want to give an argument of knowledge for $\prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = b$. Our strategy is to compute a commitment

$$c_b = \mathrm{com}_{ck}\left(\prod_{j=1}^{m} a_{1j}, \ldots, \prod_{j=1}^{m} a_{nj}; s\right).$$

We give an argument of knowledge that c_b is correct, i.e. it contains $\prod_{j=1}^{m} a_{1j}, \ldots, \prod_{j=1}^{m} a_{nj}$. Next, we give an argument of knowledge that b is the product of the values inside c_b. We will present these two arguments in Sections 5.3.1 and 5.3.3. Here, we just give an overview of the protocol.

Statement: $\{\mathbb{G}, p, q\},\ ck,\ \boldsymbol{c}_A \in \mathbb{G}^m$ and $b \in \mathbb{Z}_q$.

Prover's witness: $A \in \mathbb{Z}^{n\times m}, \boldsymbol{r} \in \mathbb{Z}_q^m$ such that

$$\boldsymbol{c}_A = \mathrm{com}_{ck}(A; \boldsymbol{r}) \qquad \text{and} \qquad \prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = b.$$

Initial message:

1. Pick $s \leftarrow \mathbb{Z}_q$ and compute $c_b = \mathrm{com}_{ck}\left(\prod_{j=1}^{m} a_{1j}, \ldots, \prod_{j=1}^{m} a_{nj}; s\right)$ and send c_b to the verifier.
2. Engage in an SHVZK argument of knowledge as described in Section 5.3.1 of $c_b = \mathrm{com}_{ck}\left(\prod_{j=1}^{m} a_{1j}, \ldots, \prod_{j=1}^{m} a_{nj}; s\right)$, where $a_{11}, \ldots, a_{nm}$ are the committed values in $\boldsymbol{c}_A$.
3. Engage (in parallel) in an SHVZK argument of knowledge as described in Section 5.3.3 of b being the product of the committed values in c_b.

Verification: The verifier accepts if and only if

1. $c_b \in \mathbb{G}$
2. Both SHVZK arguments are convincing.

Theorem 19. *The protocol is a public coin perfect generalized $\Sigma-$protocol of openings*

$$a_{11}, \ldots, a_{nm}, r_1, \ldots, r_m \in \mathbb{Z}_q$$

such that $b = \prod_{i=1}^{n}\prod_{i=1}^{m} a_{ij}$.

Proof. Perfect completeness follows from the perfect completeness of the two underlying SHVZK arguments since the prover's opening of c_b gives a satisfying input to both arguments.

Perfect SHVZK follows from the perfect hiding property of the commitment scheme and the perfect SHVZK of the two underlying arguments. The simulator picks $c_b = \mathrm{com}_{ck}(0, \ldots, 0; s)$ for random $s \leftarrow \mathbb{Z}_q$ and runs the simulator for the two underlying arguments.

It remains to argue that we have a perfect generalized special soundness extractor. The extractor runs the extractors of the underlying arguments such that it gets openings $a_{11}, \ldots, a_{nm}, r_1, \ldots, r_m$ of $\boldsymbol{c}_A$ and an opening $b_1, \ldots, b_n, s$ of c_b such that

$$b_1 = \prod_{j=1}^{m} a_{1j} \qquad \ldots \qquad b_n = \prod_{j=1}^{m} a_{nj} \qquad \text{and} \qquad b = \prod_{i=1}^{n} b_i.$$

This implies that the extracted openings of $\boldsymbol{c}_A$ satisfy $\prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = b$ and are the same witnesses as known by the prover. If this were not the case the extractor could be used by the prover to find another opening for at least one of the commitments, and the commitment scheme would be broken. □

Efficiency: Before both parties can engage in the underlying multi Hadamard argument and zero argument, the prover sends one commitment. In total the communication cost consists of $3m$ group elements and $4n$ field elements and therefore is $O(m + n)$.

The prover has to calculate mn multiplications in $\mathbb{Z}_q$ to calculate the vector $\boldsymbol{b}$ and also the commitment to $\boldsymbol{b}$. Together with the cost of the underlying argument this give a computation cost of $mn+n+5m$ exponentiations in $\mathbb{G}$ and $m^2n + 5mn + 5n + 2m$ multiplications. Using Fast Fourier Transform as described in section 5.3.2 the number of multiplication can be reduced to $O(mn \log m)$. Furthermore, using multi-exponentiation techniques, described in section 4.3, the number of exponentiations can be reduced down to $O\left(\frac{mn}{\log n}\right)$.

The verifier calculates $6n + 5m$ exponentiations in and $8n + 5m$ multiplications in $\mathbb{G}$; thus the computation complexity is counting single exponentiations $O(n + m)$ or applying multi-exponentiation techniques $O\left(\frac{n}{\log n} + \frac{m}{\log m}\right)$, see Section 4.3.

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 46 \rangle \subset \mathbb{Z}^*_{179}$, which has prime order $q = 89$.

The statement contains $\{\mathbb{G}, p, q\} = \{\langle 46, \rangle, 179, 89\}$, $ck = \{G_1, \ldots, G_4, H\} = \{46, 177, 161, 100, 72\}$ and commitments $c_A = (22, 89, 67)$ and $b = \prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = 10$. The prover knows a 4×3 matrix

$$A = \begin{pmatrix} 63 & 10 & 21 \\ 55 & 72 & 72 \\ 88 & 76 & 75 \\ 47 & 84 & 56 \end{pmatrix} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3),$$

$b = \prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = 10 \in \mathbb{Z}_{179}$, and vector $\boldsymbol{r} = (1, 10, 25)^T$ such that

$$c_{A_1} = \mathrm{com}_{ck}(\boldsymbol{a}_1; r_1) = 22 \qquad c_{A_2} = \mathrm{com}_{ck}(\boldsymbol{a}_2; r_2) = 89 \qquad c_{A_3} = \mathrm{com}_{ck}(\boldsymbol{a}_3; r3) = 61.$$

The statement of the argument is c_A and $b = 10$.

The prover computes the Hadamard product of the vectors of A to get $\boldsymbol{b} = (58, 53, 85, 12)^T$,

commits themselves to $\boldsymbol{b}$ in $c_b = \mathrm{com}_{ck}(\boldsymbol{b}; s) = 74$ by picking $s = 5$.

Next, both side engage in a multi Hadamard product of $c_b = \mathrm{com}_{ck}\left(\prod_{j=1}^{m} a_{1j}, \ldots \prod_{j=1}^{m} a_{nj}; s\right)$ and in parallel they engage in a single value product argument of b being the product of the entries of vector $\boldsymbol{b}$. The description of these protocols can be found in the following sections.

5.3.1 Multi Hadamard Product Argument

We will give an argument for committed values $a_{11}, \ldots, a_{nm}$ and $b_1, \ldots, b_n$ satisfying

$$b_i = \prod_{j=1}^{m} a_{ij}.$$

Again it will be convenient to write this with vector notation. We have commitments $\boldsymbol{c}_A$ and c_b and the prover wants to argue knowledge of openings to tuples $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m, \boldsymbol{b} \in \mathbb{Z}_q^n$ such that $\boldsymbol{b} = \bigodot_{i=1}^{m} \boldsymbol{a}_i$, where $\bigodot$ stands for the Hadamard product, which is the entry-wise product of vectors.

The prover generates commitments $\boldsymbol{c}_B$ to the matrix B with columns

$$\boldsymbol{b}_1 = \boldsymbol{a}_1 \qquad \boldsymbol{b}_2 = \bigodot_{i=1}^{2} \boldsymbol{a}_i \qquad \ldots \qquad \boldsymbol{b}_{m-1} = \bigodot_{i=1}^{m-1} \boldsymbol{a}_i \qquad \boldsymbol{b}_m = \bigodot_{i=1}^{m} \boldsymbol{a}_i.$$

By picking $c_{B_1} = c_{A_1}$ and $c_{B_m} = c_b$ the prover guarantees $\boldsymbol{b}_1 = \boldsymbol{a}_1$ and $\boldsymbol{b}_m = \boldsymbol{b}$. The prover's strategy is to prove that for each $i = 1, \ldots, m-1$

$$\boldsymbol{b}_{i+1} = \boldsymbol{a}_{i+1}\boldsymbol{b}_i.$$

Since $\boldsymbol{b}_1 = \boldsymbol{a}_1$ and $\boldsymbol{b}_m = \boldsymbol{b}$ this shows $\boldsymbol{b} = \bigodot_{i=1}^{m} \boldsymbol{a}_i$.

We will use randomization to simplify the argument. The verifier will send a challenge x and the prover will demonstrate

$$\sum_{i=1}^{m-1} x^i \boldsymbol{b}_{i+1} = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1}(x^i \boldsymbol{b}_i).$$

Defining $c_{D_i} = c_{B_i}^{x^i}$ and $c_D = \prod_{i=1}^{m-1} c_{B_{i+1}}^{x^i}$ we get commitments to the vectors

$$\boldsymbol{d}_1 = x\boldsymbol{b}_1 \qquad \boldsymbol{d}_2 = x^2\boldsymbol{b}_2 \qquad \ldots \qquad \boldsymbol{d}_{m-1} = x^{m-1}\boldsymbol{b}_{m-1} \qquad \boldsymbol{d} = \sum_{i=1}^{m-1} x^i \boldsymbol{b}_{i+1}.$$

Thus, we have reduced the problem to demonstrating that the committed values satisfy

$$\boldsymbol{d} = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1} \circ \boldsymbol{d}_i.$$

The argument can be made more efficient by having the verifier send a challenge y and defining the bilinear map

$$* : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q \quad \text{by} \quad (a_1, \ldots, a_n)^T * (d_1, \ldots, d_n)^T = \sum_{j=1}^{n} a_j d_j y^j. \tag{5.1}$$

The prover will now use the zero argument from Section 5.3.2 to demonstrate that

$$0 = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1} * \boldsymbol{d}_i - \mathbf{1} * \boldsymbol{d},$$

which happens with negligible probability over y unless indeed $\boldsymbol{d} = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1} \circ \boldsymbol{d}_i$.

Statement: $\{\mathbb{G}, p, q\},\ \ ck,\ \boldsymbol{c}_A \in \mathbb{G}^m, c_b \in \mathbb{G}$

Prover's witness: $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m, \boldsymbol{r}$, and $\boldsymbol{b}, s$ such that

$$\boldsymbol{c}_A = \mathrm{com}_{ck}(A; \boldsymbol{r}) \qquad c_b = \mathrm{com}_{ck}(\boldsymbol{b}; s) \qquad \boldsymbol{b} = \bigodot_{i=1}^{m} \boldsymbol{a}_i.$$

Initial message: Set

1. $\boldsymbol{b}_1 = \boldsymbol{a}_1, \boldsymbol{b}_2 = \boldsymbol{a}_1 \circ \boldsymbol{a}_2, \ldots, \boldsymbol{b}_{m-1} = \boldsymbol{a}_1 \circ \cdots \circ \boldsymbol{a}_{m-1}$, and $\boldsymbol{b}_m = \boldsymbol{b}$
2. $c_{B_2} = \mathrm{com}_{ck}(\boldsymbol{b}_2; s_2), \ldots, c_{B_{m-1}} = \mathrm{com}_{ck}(\boldsymbol{b}_{m-1}; s_{m-1})$ where $s_2, \ldots, s_{m-1} \leftarrow \mathbb{Z}_q$
3. $s_1 = r_1$ and $s_m = s$
4. $c_{B_1} = c_{A_1}$ and $c_{B_m} = c_b$

Send: $\boldsymbol{c}_B$

Challenge: $x, y \leftarrow \mathbb{Z}_q^*$

Answer: 1. Define the bilinear map

$$* : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q \text{ by } (a_1, \ldots, a_n)^T * (d_1, \ldots, d_n)^T = \sum_{j=1}^{n} a_j d_j y^j.$$

2. Define $c_{D_i} = c_{B_i}^{x^i}$ and $c_D = \prod_{i=1}^{m-1} c_{B_{i+1}}^{x^i}$ and $c_{-1} = \mathrm{com}_{ck}(-\mathbf{1}; 0)$ and engage in the SHVZK zero argument described in Section 5.3.2 for the committed values satisfying

$$0 = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1} * \boldsymbol{d}_i - \mathbf{1} * \boldsymbol{d}.$$

The prover's witness in this argument consists of the openings of $c_{A_2}, \ldots, c_{A_m}, c_{-1}$ and the openings of $c_{D_1}, \ldots, c_{D_{m-1}}, c_D$. The latter openings can be computed as

$$\boldsymbol{d}_1 = x\boldsymbol{b}_1, \quad t_1 = xs_1, \quad \ldots, \quad \boldsymbol{d}_{m-1} = x^{m-1}\boldsymbol{b}_{m-1}, \quad t_{m-1} = x^{m-1}s_{m-1},$$

and

$$\boldsymbol{d} = \sum_{i=1}^{m-1} x^i \boldsymbol{b}_{i+1} \qquad t = \sum_{i=1}^{m-1} x^i s_{i+1}.$$

Verification: Check

1. $c_{B_2}, \ldots, c_{B_{m-1}} \in \mathbb{G}$
2. $c_{B_1} = c_{A_1}, c_{B_m} = c_b$
3. Define $c_{D_i} = c_{B_i}^{x^i}$, $c_D = \prod_{i=1}^{m-1} c_{B_{i+1}}^{x^i}$ and $c_{-1} = \mathrm{com}_{ck}(-\mathbf{1}; 0)$. Accept if the underlying zero argument is valid.

Theorem 20. *The protocol is a public coin perfect generalized $\Sigma-$protocol of committed vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m, \boldsymbol{b}$ such that $\boldsymbol{b} = \bigodot_{i=1}^{m} \boldsymbol{a}_i$.*

Proof. It is straightforward to check that the inputs to the underlying zero argument are correct. Therefore, perfect completeness follows from the perfect completeness of the underlying zero argument.

Given challenges x, y the simulator picks $s_2, \ldots, s_{m-1} \leftarrow \mathbb{Z}_q$, computes

$$c_{B_2} = \mathrm{com}_{ck}(\mathbf{0}; s_2) \qquad \ldots \qquad c_{B_{m-1}} = \mathrm{com}_{ck}(\mathbf{0}; s_{m-1}),$$

and runs the SHVZK simulator for the underlying zero-knowledge argument. The perfect hiding property of the commitments and the perfect SHVZK of the underlying zero argument shows that this is a perfect simulation.

It remains to argue that we have perfect generalized special soundness. The extractor E runs the extractor of the zero argument to get openings of $c_{A_2}, \ldots, c_{A_m}, c_{-1}$, and $c_{D_1}, \ldots, c_{D_{m-1}}, c_D$ of the form $\boldsymbol{a}_2, r_2, \ldots, \boldsymbol{a}_m, r_m, -\mathbf{1}, 0$, and $\boldsymbol{d}_1, t_1, \ldots, \boldsymbol{d}_{m-1}, t_{m-1}, \boldsymbol{d}, t$. This gives us openings of $\boldsymbol{c}_B$ computed as

$$\boldsymbol{b}_1 = x^{-1}\boldsymbol{d}_1 \qquad s_1 = x^{-1}t_1 \qquad \ldots \qquad \boldsymbol{b}_{m-1} = x^{1-m}\boldsymbol{d}_{m-1} \qquad s_{m-1} = x^{1-m}t_{m-1}$$

$$\boldsymbol{b}_m = x^{1-m}(\boldsymbol{d} - \sum_{i=1}^{m-2} x^i \boldsymbol{b}_{i+1}) \qquad s_m = x^{1-m}(t - \sum_{i=1}^{m-2} x^i s_{i+1}).$$

Since $c_{B_1} = c_{A_1}$, and $c_{B_m} = c_b$ we automatically get openings $\boldsymbol{a}_1 = \boldsymbol{b}_1, r_1 = s_1$ and $\boldsymbol{b} = \boldsymbol{b}_m, s = s_m$ of these commitments.

The remaining question is whether the extracted witness satisfies the statement. The binding property of the commitment scheme implies that $c_{D_1}, \ldots, c_{D_{m-1}}$ contain openings $\boldsymbol{d}_1 = x\boldsymbol{b}_1, \ldots, \boldsymbol{d}_{m-1} = x^{m-1}\boldsymbol{b}_{m-1}$, and c_D contains $\boldsymbol{d} = \sum_{i=1}^{m-1} x^i \boldsymbol{b}_{i+1}$ for the randomly chosen x. Since $\boldsymbol{c}_B$ is fixed before seeing the challenges $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_{m-1}, \boldsymbol{d}$ are determined before the prover sees the y that defines the bilinear map $*$. The special soundness property of the underlying zero argument implies

$$\mathbf{1} * \boldsymbol{d} = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1} * \boldsymbol{d}_i,$$

which has negligible probability of holding for random y unless $\boldsymbol{d} = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1} \circ \boldsymbol{d}_i$. This implies

$$\sum_{i=1}^{m-1} x^i \boldsymbol{b}_{i+1} = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1} \circ (x^i \boldsymbol{b}_i),$$

which has negligible probability of holding for random x unless $\boldsymbol{b}_2 = \boldsymbol{a}_2 \circ \boldsymbol{b}_1, \ldots, \boldsymbol{b}_m = \boldsymbol{a}_m \circ \boldsymbol{b}_{m-1}$.

This shows the extracted openings satisfy $\boldsymbol{b} = \boldsymbol{b}_m = \bigodot_{i=1}^{m} \boldsymbol{a}_i$ as required in the statement and they are the same witnesses as known by the prover. If this were not the case the extractor could be used by the prover to find another opening for at least one of the commitments, and the commitment scheme would be broken. □

Efficiency: The communication cost consists of m group elements plus the cost of one single value argument, so in total m group element and $2n$ field elements. Therefore, the communication cost is $O(m+n)$.

The prover has to commit herself to the vectors $\boldsymbol{b}_i$ and doing this they needs to calculate $mn + m - 2n$ exponentiations in $\mathbb{G}$ and $2mn$ multiplications. Together with the underlying protocol the prover calculates $mn+m+n$ exponentiations in $\mathbb{G}$ and $2mn+7n$ multiplications. Using multi-exponentiations techniques, see Section 4.3, the cost of the exponentiations can be reduced to $O\left(\frac{mn}{\log n}\right)$.

Before the verifier can engage in the single value argument they has to calculate $m+n$ exponentiations in $\mathbb{G}$ and $n+m$ multiplications. The total number is $3n+m$ exponentiations and $5n+m$ multiplications in $\mathbb{G}$. Using multi-exponentiation techniques, see Section 4.3, this cost is $O\left(\frac{n}{\log n} + \frac{m}{\log m}\right)$.

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 46 \rangle \subset \mathbb{Z}_{179}^*$, which has prime order $q = 233$. This is the underlying Hadamard product for the product argument example, but can also be observed on its own.

The statement consists of $\{G, p, q\} = \{\langle 47 \langle, 179, 89\}$, $ck = \{G_1, \ldots, G_4, H\} = \{47, 177, 161, 100, 72\}$, and commitments $c_A = (22, 61, 69) \in \mathbb{G}^3$ and $c_b = \mathrm{com}_{ck}(\boldsymbol{b}; s) = 74 \in \mathbb{G}$

The prover knows witnesses

$$A = \begin{pmatrix} 63 & 10 & 21 \\ 55 & 72 & 72 \\ 88 & 76 & 75 \\ 47 & 84 & 56 \end{pmatrix} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3),$$

$\boldsymbol{r} = (11025)$, $\boldsymbol{b} = (58, 53, 85, 12)^T$, and $s = 5$ such that

$$c_A = \mathrm{com}_{ck}(A; \boldsymbol{r}) = (22, 89, 61) \qquad c_b = \mathrm{com}_{ck}(\boldsymbol{b}; s) = 74 \qquad \boldsymbol{b} = \bigodot_{i=1}^{m} \boldsymbol{a}_i.$$

The prover wants to convince the verifier that $\boldsymbol{b}$ is indeed the Hadamard product of the $a_i's$.

The prover first sets $\boldsymbol{b}_1 = \boldsymbol{a}_1$, $\boldsymbol{b}_2 = \boldsymbol{a}_1 \circ \boldsymbol{a}_2 = (7, 44, 13, 32)^T$, $\boldsymbol{b}_3 = \boldsymbol{b}$, and $s_1 = r_1 = 1, s_3 = s = 5$, picks $s_2 = 64$ and commits themselves to $\boldsymbol{b}_2$ in $c_{B_2} = \mathrm{com}_{ck}(\boldsymbol{b}_2; s_2) = 61 \in \mathbb{G}$. Lastly, the prover sends

$$c_{B_1} = c_{A_1} = 22 \qquad c_{B_2} = 61 \qquad c_{B_3} = c_b = 169$$

to the verifier.

The verifier picks challenges $x = 62$ and $y = 8$, and both parties define the bilinear map

$$* : \mathbb{Z}_{179}^4 \times \mathbb{Z}_{179}^4 \to \mathbb{Z}_{179} \text{ by } (a_1, \ldots, a_4)^T * (d_1, \ldots, d_4)^T = \sum_{i=1}^{4} a_i d_i y^i = \sum_{i=1}^{4} a_i d_i 8^i.$$

Then they set

$$c_{D_1} = c_{B_1}^x = 161 \quad c_{D_2} = c_{B_2}^{x^2} = 149 \quad c_{D_3} = c_{B_3}^{x^3} = 27,$$

$$c_D = c_{B_2}^x c_{B_3}^{x^2} = 9 \quad c_{-1} = \mathrm{com}_{ck}(-1; 0) = 144.$$

The openings of $c_{D_1}, c_{D_2}, c_{D_3}$ are

$$\boldsymbol{d}_1 = x\boldsymbol{b}_1 = (7, 18, 17, 66)^T \quad t_1 = xs_1 = 62,$$

$$\boldsymbol{d}_2 = x^2\boldsymbol{b}_2 = (30, 36, 43, 10)^T, \quad t_2 = x^2 s_2 = 20,$$

$$\boldsymbol{d}_3 = x^3\boldsymbol{b}_3 = (18, 59, 56, 10)^T, \quad t_3 = x^3 s_3 = 19$$

and the opening of c_D is

$$\boldsymbol{d} = x\boldsymbol{b}_2 + x^2\boldsymbol{b}_3 = (85, 69, 5, 26)^T, \qquad t = 48.$$

These openings can be easily calculated by the prover and they can engage in a zero argument, described in the next section, for the committed values satisfying $0 = \boldsymbol{a}_2 * \boldsymbol{d}_1 + \boldsymbol{a}_3 * \boldsymbol{d}_2 - \mathbf{1} * d$.

The verifier accepts if $c_{B_2} \in \mathbb{G}$ and $c_{B_1} = c_{A_1}, c_{B_3} = c_b$, and the zero argument is valid.

5.3.2 Zero Argument

Given a bilinear map $* : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q$ and commitments to $\boldsymbol{a}_1, \boldsymbol{b}_0, \ldots, \boldsymbol{a}_m, \boldsymbol{b}_{m-1}$ the prover wants to show that $0 = \sum_{i=1}^{m} \boldsymbol{a}_i * \boldsymbol{b}_{i-1}$. The bilinear map can be defined by $y \in \mathbb{Z}_q$ as described in equation 5.1, or by the inner product of vectors.

The prover picks $\boldsymbol{a}_0, \boldsymbol{b}_m \leftarrow \mathbb{Z}_q^n$ and calculates commitments to these values. The prover computes for $k = 0, \ldots, 2m$ the values

$$d_k = \sum_{\substack{0 \le i, j \le m \\ j = (m-k)+i}} \boldsymbol{a}_i * \boldsymbol{b}_j$$

and commits to them. Observe that $d_{m+1} = \sum_{i=1}^{m} \boldsymbol{a}_i * \boldsymbol{b}_{i-1} = 0$. The prover sets $c_{D_{m+1}} = \mathrm{com}_{ck}(0; 0)$ such that the verifier can see $d_{m+1} = 0$.

The verifier picks a challenge x and uses the homomorphic property to compute commitments to $\boldsymbol{a} = \sum_{i=0}^{m} x^i \boldsymbol{a}_i$ and $\boldsymbol{b} = \sum_{j=0}^{m} x^{m-j} \boldsymbol{b}_j$, the prover will provide openings of these commitments. The verifier can also compute a commitment to $\sum_{k=0}^{2m} d_k x^k$ and the prover opens this commitment. We observe that if everything is computed correctly by the prover then

$$\sum_{k=0}^{2m} d_k x^k = \Big(\sum_{i=0}^{m} x^i \boldsymbol{a}_i \Big) * \Big(\sum_{j=0}^{m} x^{m-j} \boldsymbol{b}_j \Big).$$

The verifier checks this equation by testing if $\prod_{k=0}^{2m} c_{D_k}^{x^k} = \mathrm{com}_{ck}(\boldsymbol{a} * \boldsymbol{b}; t)$. There is negligible chance that the prover convinces the verifier unless the polynomials are identical. Since $d_{m+1} = \mathrm{com}_{ck}(0; 0)$ the coefficient of x^{m+1} is 0 showing that $0 = \sum_{i=1}^{m} \boldsymbol{a}_{i+1} * \boldsymbol{b}_i$,

Statement: $\{\mathbb{G}, p, q\},\ ck,\ \boldsymbol{c}_A, \boldsymbol{c}_B \in \mathbb{G}^m$ and a specification of a bilinear map

$$* : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q.$$

Prover's witness: $A = \{\boldsymbol{a}_i\}_{i=1}^m \in \mathbb{Z}_q^{n \times m}, \boldsymbol{r} \in \mathbb{Z}_q^m$, and $B = \{\boldsymbol{b}_i\}_{i=0}^{m-1}, \boldsymbol{s} = (s_0, \ldots, s_{m-1}) \in \mathbb{Z}_q^m$ such that

$$\boldsymbol{c}_A = \mathrm{com}_{ck}(A; \boldsymbol{r}) \qquad \boldsymbol{c}_B = \mathrm{com}_{ck}(B; \boldsymbol{s}) \qquad 0 = \sum_{i=1}^{m} \boldsymbol{a}_i * \boldsymbol{b}_{i-1}.$$

Initial message: Compute:

1. $c_{A_0} = \mathrm{com}_{ck}(\boldsymbol{a}_0; r_0) \quad c_{B_m} = \mathrm{com}_{ck}(\boldsymbol{b}_m; s_m) \quad$ where $\boldsymbol{a}_0, \boldsymbol{b}_m \leftarrow \mathbb{Z}_q^n$ and $r_0, s_m \leftarrow \mathbb{Z}_q$
2. for $k = 0, \ldots, 2m$:
$$d_k = \sum_{\substack{0 \le i,j \le m \\ j=(m-k)+i}} \boldsymbol{a}_i * \boldsymbol{b}_j$$
3. $\boldsymbol{c}_D = \mathrm{com}_{ck}(\boldsymbol{d}; \boldsymbol{t})$ where $\boldsymbol{t} = (t_0 \ldots t_{2m}) \leftarrow \mathbb{Z}_q^{2m+1}$ and $t_{m+1} = 0$

Send: $c_{A_0}, c_{B_m}, \boldsymbol{c}_D$

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Calculate

1. $\overline{\boldsymbol{a}} = \sum_{i=0}^{m} x^i \boldsymbol{a}_i \quad$ and $\quad \overline{r} = \sum_{i=0}^{m} x^i r_i$
2. $\overline{\boldsymbol{b}} = \sum_{j=0}^{m} x^{m-j} \boldsymbol{b}_j \quad$ and $\quad \overline{s} = \sum_{j=0}^{m} x^{m-j} s_j$
3. $\overline{t} = \sum_{k=0}^{2m} x^k t_k$.

Send: $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \overline{r}, \overline{s}, \overline{t}$

Verification: Accept if and only if

1. $c_{A_0}, c_{B_m} \in \mathbb{G}, \quad \boldsymbol{c}_D \in \mathbb{G}^{2m+1}$ and $c_{D_{m+1}} = \mathrm{com}_{ck}(0; 0) \in \mathbb{G}$
2. $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}} \in \mathbb{Z}_q^n$ and $\overline{r}, \overline{s}, \overline{t} \in \mathbb{Z}_q$
3. $\prod_{i=0}^{m} c_{A_i}^{x^i} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}}, \overline{r}) \qquad \prod_{j=0}^{m} c_{B_j}^{x^{m-j}} = \mathrm{com}_{ck}(\overline{\boldsymbol{b}}; \overline{s})$
4. $\prod_{k=0}^{2m} c_{D_k}^{x^k} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}} * \overline{\boldsymbol{b}}; \overline{t})$

Theorem 21. *The protocol is a public coin perfect generalized $\Sigma-$protocol of committed values $\boldsymbol{a}_1, \boldsymbol{b}_0, \ldots, \boldsymbol{a}_m, \boldsymbol{b}_{m-1}$ such that $0 = \sum_{i=1}^{m} \boldsymbol{a}_i * \boldsymbol{b}_{i-1}$.*

Proof. We have $d_{m+1} = \sum_{i=1}^{m} \boldsymbol{a}_i * \boldsymbol{b}_{i-1} = 0$ and

$$\boldsymbol{a} * \boldsymbol{b} = \left(\sum_{i=0}^{m} x^i \boldsymbol{a}_i\right) * \left(\sum_{j=0}^{m} x^{m-j} \boldsymbol{b}_j\right) = \sum_{k=0}^{2m} \sum_{\substack{0 \le i,j \le m \\ j=(m-k)+i}} \boldsymbol{a}_i * \boldsymbol{b}_j = \sum_{k=0}^{2m} d_k x^k.$$

Perfect completeness now follows by direct verification.

The argument is perfect SHVZK. The simulator picks, on challenge x, $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}} \leftarrow \mathbb{Z}_q^n$, and $\overline{r}, \overline{s}, t_0, t_1, \ldots, t_m, t_{m+2}, \ldots, t_{2m} \leftarrow \mathbb{Z}_q$ and defines $t_{m+1} = 0$. It computes

$$c_{A_0} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}}; \overline{r}) \prod_{i=1}^{m} c_{A_i}^{-x^i} \qquad c_{B_m} = \mathrm{com}_{ck}(\overline{\boldsymbol{b}}; \overline{s}) \prod_{j=0}^{m-1} c_{B_j}^{-x^{m-j}} \qquad \overline{t} = \sum_{k=0}^{2m} t_k x^k$$

$$c_{D_0} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}} * \overline{\boldsymbol{b}}; t_0) \qquad c_{D_1} = \mathrm{com}_{ck}(0; t_1) \qquad \ldots \qquad c_{D_{2m}} = \mathrm{com}_{ck}(0; t_{2m}).$$

The simulated argument is $(c_{A_0}, c_{B_m}, \boldsymbol{c}_D, x, \overline{\boldsymbol{a}}, \overline{r}, \overline{\boldsymbol{b}}, \overline{s}, \overline{t})$. To see that this is a perfect simulation note that the c_{D_i}'s are perfectly hiding commitments and $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \overline{r}, \overline{s}, \overline{t}$ are uniformly random both in a real argument and in the simulation. Conditioned on those values the commitments $c_{A_0}, c_{B_m}, c_{D_0}, c_{D_{m+1}}$ are uniquely determined by the verification equations; thus, real arguments and simulated arguments have identical probability distributions.

It remains to prove that we have perfect generalized special soundness. Given $2m$ accepting transcripts with different challenges $x_0, \ldots, x_{2m}$ satisfying for each x_ℓ that $c_{D_{m+1}} = \mathrm{com}_{ck}(0; 0)$ and

$$\prod_{i=0}^{m} c_{A_i}^{x_\ell^i} = \mathrm{com}_{ck}\left(\overline{\boldsymbol{a}}^{(\ell)}, r^{(\ell)}\right) \qquad \prod_{j=0}^{m} c_{B_j}^{x_\ell^{m-j}} = \mathrm{com}_{ck}\left(\overline{\boldsymbol{b}}^{(\ell)}; \overline{s}^{(\ell)}\right)$$

$$\prod_{k=0}^{2m} c_{D_k}^{x_\ell^k} = \mathrm{com}_{ck}\left(\overline{\boldsymbol{a}}^{(\ell)} * \overline{\boldsymbol{b}}^{(\ell)}; \overline{t}^{(\ell)}\right),$$

the extractor E can find a witness. Since the vectors $(1, x_\ell, \ldots, x_\ell^{2m})$ form the columns of a transposed Vandermonde matrix and all the x_ℓ's are different we can find the inverse matrix X^{-1}. Define

$$\boldsymbol{d}_x = \left(\overline{\boldsymbol{a}}^{(0)} * \overline{\boldsymbol{b}}^{(0)}, \ \ldots, \ \overline{\boldsymbol{a}}^{(2m)} * \overline{\boldsymbol{b}}^{(2m)}\right)$$

and

$$\boldsymbol{t}_x = \left(\overline{t}^{(0)}, \ \ldots, \ \overline{t}^{(2m)}\right),$$

this gives for each $i = 0, \ldots, 2m$ an opening of c_{D_i} applying

$$\boldsymbol{c}_D = (\boldsymbol{c}_D^X)^{X^{-1}} = \mathrm{com}_{ck}(\boldsymbol{d}_x; \boldsymbol{t}_x)^{X^{-1}} = \mathrm{com}_{ck}(\boldsymbol{d}_x X^{-1}; \boldsymbol{t}_x X^{-1}).$$

The extractor gets openings of $c_{A_0}, \ldots, c_{A_m}$ and $c_{B_0}, \ldots, c_{B_m}$ in a similar manner.

Having openings of $\boldsymbol{c}_A, \boldsymbol{c}_B, \boldsymbol{c}_D$ to values $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_m, \boldsymbol{b}_0, \ldots, \boldsymbol{b}_m, d_0, \ldots, d_{2m}$ the binding property

of the commitment scheme implies that the answer to a random challenge x is of the form

$$\overline{\boldsymbol{a}} = \sum_{i=0}^{m} x^i \boldsymbol{a}_i \qquad \overline{\boldsymbol{b}} = \sum_{j=0}^{m} x^{m-j} \boldsymbol{b}_j \quad \text{satisfying} \quad \sum_{k=0}^{2m} d_k x^k = \overline{\boldsymbol{a}} * \overline{\boldsymbol{b}}.$$

This implies

$$\sum_{k=0}^{2m} d_k x^k = \left(\sum_{i=0}^{m} x^i \boldsymbol{a}_i \right) * \left(\sum_{j=0}^{m} x^{m-j} \boldsymbol{b}_j \right).$$

The Schwartz-Zippel lemma implies that the prover has negligible chance of making an acceptable argument unless $d_{m+1} = \sum_{i=1}^{m} \boldsymbol{a}_i * \boldsymbol{b}_{i-1}$. Since $d_{m+1} = 0$ this gives us that the extracted openings satisfy $0 = \sum_{i=1}^{m} \boldsymbol{a}_i * \boldsymbol{b}_{i-1}$. The extracted openings are also the same witnesses known by the prover. Otherwise the extractor could be used by the prover to break the commitment scheme and find two openings for at least one commitment with non-negligible probability. □

Efficiency: The communication cost is $O(m+n)$ since $2m$ commitments and $2n$ field elements are sent between the two parties during the protocol.

The prover has to commit themselves to two vectors in $\mathbb{Z}_q^n$ and another $2m$ single elements, so in total $2n + 4m$ exponentiations are calculated. This number can be reduced to $O\left(\frac{n}{\log n} + \frac{m}{\log m}\right)$ using multi-exponentiations techniques, described in Section 4.3.

On top of this the prover has to calculate $m^2 n$ multiplications in $\mathbb{Z}_q$ to generate the d_i's and another $2mn + 2n + 5m$ multiplications in the rest of the protocol. The cost of the $m^2 n$ multiplication can be dominant, but using Fast Fourier Transform as described below it is possible to reduce these $m^2 n$ multiplications down $O(mn \log m)$ multiplications.

The verifier only needs to calculate $4n+5m$ multiplications and $4m+3n$ exponentiations in $\mathbb{G}$, applying multi-exponentiations techniques, Section 4.3, this relates to $O\left(\frac{n}{\log n} + \frac{m}{\log m}\right)$ exponentiations.

Reducing the prover's computation: Naïvely, the prover would use approximately $m^2 n$ multiplications in $\mathbb{Z}_q$ to first compute the products $\boldsymbol{a}_i * \boldsymbol{b}_j$ and then summing them to compute the d_k's. Using Fast Fourier Transform techniques, see also section 4.2, this can be reduced to $O(mn \log m)$ multiplications in $\mathbb{Z}_q$ if m and q are chosen such that $2m$ is a power of 2 and $2m | q-1$.

We will now outline how the Fast Fourier Transform can be used. Let $\omega \in \mathbb{Z}_q$ be a $2m$-root of unity, i.e. $\omega^{2m} = 1$. The idea is to evaluate the polynomial $\sum_{k=0}^{2m} d_k x^k$ in $2m+1$ different points $x = 1, \omega, \omega^2, \ldots, \omega^{2m-1}$ and then use polynomial interpolation to recover the coefficients $d_0, \ldots, d_{2m}$.

The polynomial interpolation is fairly efficient in our setting since it only applies to single elements in $\mathbb{Z}_q$. The evaluation of the polynomial in $x = 0$ is also easy but the evaluations in $x = 1, \omega, \omega^2, \ldots, \omega^{2m-1}$ take time. Using the Fast Fourier Transform this can be done with $O(mn \log m)$ multiplications though. We compute $\boldsymbol{a}(x) = \sum_{i=0}^{m} x^i \boldsymbol{a}_i$ and $\boldsymbol{b}(x) = \sum_{j=0}^{m} x^{m-j} \boldsymbol{b}_j$ in the $2m$ roots of unity at a cost of $O(mn \log m)$ multiplications and using $2m$ evaluations of the bilinear map $*$ this gives us $\sum_{k=0}^{2m} d_k x^k = \boldsymbol{a}(x) * \boldsymbol{b}(x)$ evaluated in $x = 1, \omega, \ldots, \omega^{2m-1}$.

It is possible to reduce the prover's computation even further to $O(mn)$ multiplications using interactive techniques similar to the ones we described for the multi-exponentiation argument in Section 8.3.

This comes at the cost of increased round complexity though. We refer to [Gro09] for details of how to do it in a logarithmic number of rounds.

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 46 \rangle \subset \mathbb{Z}^*_{179}$, which has prime order $q = 89$. The statement is the zero-argument used in the Hadamard product argument; that means the bilinear map is defined as

$$\mathbb{Z}^4_{179} \times \mathbb{Z}^4_{179} \to \mathbb{Z}_{179} : (a_1, \ldots, a_4)^T * (b_1, \ldots, b_4)^T = \sum_{i=1}^{4} a_i b_i 8^i.$$

The statement contains $\{\mathbb{G}, p, q\} = \{\langle 46 \rangle, 179, 89\}$, $ck = \{G_1, \ldots, G_4, H\} = \{46, 177, 161, 100, 72\}$, and commitments $c_A = (c_{A_2}, c_{A_3}, c_{-1}) = (89, 67, 144)$ and $c_B = (c_{D_1}, c_{D_2}, c_D) = (161, 149, 9)$, where $c_{A_2}, c_{A_3}, c_{-1}, c_{D_1}, c_{D_2}, c_D$ are defined in the example of section 5.3.1.

The prover witness are

$$\boldsymbol{a}_1 = (63, 55, 88, 77)^T \quad \boldsymbol{a}_2 = (10, 72, 76, 84)^T \quad \boldsymbol{a}_3 = \boldsymbol{-1} \quad \boldsymbol{r} = (10, 25, 0),$$

$$\boldsymbol{b}_0 = (7, 18, 27, 66)^T \quad \boldsymbol{b}_1 = (30, 36, 43, 10)^T \quad \boldsymbol{b}_2 = (85, 69, 5, 26)^T \quad \boldsymbol{s} = (62, 20, 48),$$

such that $0 = \sum_{i=1}^{3} a_i * b_{i-1}$.

The prover picks $\boldsymbol{a}_0 = (11, 1, 74, 25)^T$, $\boldsymbol{b}_3 = (16, 63, 88, 21)$, $r_0 = 56$, and $s_3 = 14$ and commits themselves to $\boldsymbol{a}_0, \boldsymbol{b}_3$:

$$c_{A_0} = \mathrm{com}_{ck}(\boldsymbol{a}_0; r_0) = 26 \qquad c_{B_3} = \mathrm{com}_{ck}(\boldsymbol{b}_3; s_3) = 83.$$

Then the prover calculates for $k = 0, \ldots, 6$ the values $d_k = \sum_{\substack{0 \le i,j \le m \\ j=(m-k)+i}} \boldsymbol{a}_i * \boldsymbol{b}_j$:

$$d_0 = \boldsymbol{a}_0 * \boldsymbol{b}_3 = 19 \qquad d_1 = \boldsymbol{a}_0 * \boldsymbol{b}_2 + \boldsymbol{a}_1 * \boldsymbol{b}_3 = 83 \qquad d_2 = \boldsymbol{a}_0 * \boldsymbol{b}_1 + \boldsymbol{a}_1 * \boldsymbol{b}_2 + \boldsymbol{a}_2 * \boldsymbol{b}_3 = 20$$

$$d_3 = \boldsymbol{a}_0 * \boldsymbol{b}_0 + \boldsymbol{a}_1 * \boldsymbol{b}_1 + \boldsymbol{a}_2 * \boldsymbol{b}_2 + \boldsymbol{a}_3 * \boldsymbol{b}_3 = 24 \qquad d_4 = \boldsymbol{a}_1 * \boldsymbol{b}_0 + \boldsymbol{a}_2 * \boldsymbol{b}_1 + \boldsymbol{a}_3 * \boldsymbol{b}_2 = 0$$

$$d_5 = \boldsymbol{a}_2 * \boldsymbol{b}_0 + \boldsymbol{a}_3 * \boldsymbol{b}_1 = 13 \qquad d_6 = \boldsymbol{a}_3 * \boldsymbol{b}_0 = 85$$

The prover picks $\boldsymbol{t} = (13, 30, 54, 2, 0, 59, 23)^T$ and calculate commitments to the d_i's.

$$c_{d_0} = G_1^{d_0} H^{t_0} = 177 \qquad c_{d_1} = 144 \qquad c_{d_2} = 75 \qquad c_{d_3} = 156$$

$$c_{d_4} = 1 \qquad c_{d_5} = 173 \qquad c_{d_6} = 51$$

The commitments

$$c_{A_0} = 26, \qquad c_{B_3} = 83 \qquad c_D = (177, 144, 75, 156, 173, 51)$$

are then send to the verifier.

The verifier picks the challenge $x = 52$ and the prover calculates answers

$$\overline{\boldsymbol{a}} = \boldsymbol{a}_0 + \boldsymbol{a}_1 + x^2\boldsymbol{a}_2 + x^3\boldsymbol{a}_3 = (11, 64, 2, 79)^T \qquad \overline{r} = r_0 + xr_1 + x^2r_2 + x^3r_3 = 14$$

$$\overline{\boldsymbol{b}} = x^3\boldsymbol{b}_0 + x^2\boldsymbol{b}_1 + x\boldsymbol{b}_2 + \boldsymbol{b}_3 = (58, 0, 86, 48)^T \qquad \overline{s} = x^3s_0 + x^2s_1 + xs_2 + s_3 = 55$$

$$\overline{t} = t_0 + xt_1 + x^2t_2 + x^3t_3 + x^4t_4 + x^5t_5 + x^6t_6 = 69.$$

The answers are sent to the verifier and they check if the commitments belongs to $\mathbb{G}$ and the answers are valid. Then the verifier checks:

$$c_{A_0}c_{A_1}^{x}c_{A_2}^{x^2}c_{A_3}^{x^3} = 64 = \mathrm{com}_{ck}(\overline{\boldsymbol{a}}; \overline{r}) \qquad (\checkmark)$$

$$c_{B_0}^{x^3}c_{B_1}^{x^2}c_{B_2}^{x}c_{B_3} = 172 = \mathrm{com}_{ck}(\overline{\boldsymbol{b}}; \overline{s}) \qquad (\checkmark)$$

$$c_{d_0}c_{d_1}^{x}c_{d_2}^{x^2}c_{d_3}^{x^3}c_{d_4}^{x^4}c_{d_5}^{x^5}c_{d_6}^{x^6} = 74 = \mathrm{com}_{ck}(\overline{\boldsymbol{a}} * \overline{\boldsymbol{b}}; \overline{t}) \qquad (\checkmark)$$

Now the verifier is convinced that $\sum_{i=1}^{3} \boldsymbol{a}_i * \boldsymbol{b}_{i-1} = 0$.

5.3.3 Single Value Product Argument

The following 3-move argument of knowledge of committed single values having a particular product can be found in [Gro10]. Given committed values $\boldsymbol{a} \in \mathbb{Z}_q$ and public known $b \in \mathbb{Z}_q$ we will give a 3-move argument of knowledge that

$$b = \prod_{i=1}^{n} a_i.$$

Statement: $\{\mathbb{G}, p, q\},\ ck,\ c_a \in \mathbb{G}$ and $b \in \mathbb{Z}_q$

Prover's witness: $\boldsymbol{a} \in \mathbb{Z}_q^n, r \in \mathbb{Z}_q$ such that

$$c_a = \mathrm{com}_{ck}(\boldsymbol{a}; r) \qquad \text{and} \qquad b = \prod_{i=1}^{n} a_i.$$

Initial message: Compute

1. $b_1 = a_1 \qquad b_2 = a_1a_2 \qquad \ldots \qquad b_n = \prod_{i=1}^{n} a_i$
2. $c_d = \mathrm{com}_{ck}(\boldsymbol{d}; r_d)$ where $\boldsymbol{d} \leftarrow \mathbb{Z}_q^n$ and $r_d \leftarrow \mathbb{Z}_q$
3. $\delta_1 = d_1, \delta_n = 0$ and $\delta_2, \ldots, \delta_{n-1} \leftarrow \mathbb{Z}_q$, $s_1, s_x \leftarrow \mathbb{Z}_q$

$$c_\delta = \mathrm{com}_{ck}(-\delta_1 d_2, \ldots, -\delta_{n-1}d_n; s_1)$$

$$c_\Delta = \mathrm{com}_{ck}(\delta_2 - a_2\delta_1 - b_1d_2, \ldots, \delta_n - a_n\delta_{n-1} - b_{n-1}d_n; s_x)$$

Send: c_d, c_δ, c_Δ

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Compute

1. $\overline{\boldsymbol{a}} = x\boldsymbol{a} + \boldsymbol{d}$ and $\overline{r} = xr + r_d$
2. $\overline{\boldsymbol{b}} = x\boldsymbol{b} + \boldsymbol{\delta}$
3. $\overline{s} = xs_x + s_1$

Send: $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \overline{r}, \overline{s}$

Verification: The verifier accepts if only if

1. $c_d, c_\delta, c_\Delta \in \mathbb{G}$
2. $\overline{r}, \overline{s} \in \mathbb{Z}_q$ and $\overline{\boldsymbol{a}}, \overline{b} \in \mathbb{Z}_q^n$
3. $c_a^x c_d = \mathrm{com}_{ck}(\overline{a}_1, \ldots, \overline{a}_n; \overline{r})$ $\quad c_\Delta^x c_\delta = \mathrm{com}_{ck}(x\overline{b}_2 - \overline{b}_1\overline{a}_2, \ldots, x\overline{b}_n - \overline{b}_{n-1}\overline{a}_n; \overline{s})$
4. $\overline{b}_1 = \overline{a}_1$ $\quad \overline{b}_n = xb$.

Theorem 22. *The protocol is a public coin perfect generalized $\Sigma-$protocol of an opening $a_1, \ldots, a_n, r$ such that $c_a = \mathrm{com}_{ck}(\boldsymbol{a}; r)$ and $b = \prod_{i=1}^n a_i$.*

Proof. Perfect completeness follows from

$$x\overline{b}_i - \overline{b}_{i-1}\overline{a}_i = x(xb_i + \delta_i) - (xb_{i-1} + \delta_{i-1})(xa_i + d_i) = x(\delta_i - \delta_{i-1}a_i - b_{i-1}d_i) - \delta_{i-1}d_i$$

for $i = 2, \ldots, n$ since $b_i = b_{i-1}a_i$.

We will now argue that we have perfect SHVZK. The simulator gets the challenge x and has to make a convincing transcript. It picks $\overline{r} \leftarrow \mathbb{Z}_q$, $\overline{\boldsymbol{a}} \leftarrow \mathbb{Z}_q^n$ and sets $c_d = \boldsymbol{c}_a^{-x}\mathrm{com}_{ck}(\overline{\boldsymbol{a}}; \overline{r})$. It picks at random s_x and sets $c_\Delta = \mathrm{com}_{ck}(0, \ldots, 0; s_x)$. It picks at random $\overline{b}_2, \ldots, \overline{b}_{n-1}, \overline{s} \leftarrow \mathbb{Z}_q$, sets $\overline{b}_1 = \overline{a}_1$ and $\overline{b}_n = xb$ and computes $c_\delta = c_\Delta^{-x}\mathrm{com}_{ck}(x\overline{b}_2 - \overline{b}_1\overline{a}_2, \ldots, x\overline{b}_n - \overline{b}_{n-1}\overline{a}_n; \overline{s})$. To see this is a perfect simulation note that c_Δ is a perfectly hiding commitment just like in a real proof, and $\overline{a}_1, \ldots, \overline{a}_n, \overline{r}$, and $\overline{b}_1, \ldots, \overline{b}_n, \overline{s}$ are distributed just like in a real proof. These choices uniquely determine c_d, c_δ according to the verification equations giving us that simulated and real proofs are identically distributed.

Finally, we will show that the protocol has perfect generalized special soundness. Given two accepting transcripts with challenges $x_1, x_2, x_1 \neq x_2$, we have

$$c_a^{x_1} c_d = \mathrm{com}_{ck}\left(\overline{a}_1^{(1)}, \ldots, \overline{a}_n^{(1)}; \overline{r}^{(1)}\right) \qquad c_a^{x_2} c_d = \mathrm{com}_{ck}\left(\overline{a}_1^{(2)}, \ldots, \overline{a}_n^{(2)}; \overline{r}^{(2)}\right),$$

which implies $c_a^{x_1 - x_2} = \mathrm{com}_{ck}(\overline{a}_1^{(1)} - \overline{a}_1^{(2)}, \ldots, \overline{a}_n^{(1)} - \overline{a}_n^{(2)}; \overline{r}^{(1)} - \overline{r}^{(1)})$ and gives us an opening of c_a. The equation $c_d = c_a^{-x}\mathrm{com}_{ck}(\overline{a}_1, \ldots, \overline{a}_n; \overline{r})$ then gives us an opening of c_d. Similarly, we can get openings of c_δ, c_Δ.

It remains to argue that there is negligible probability of extracting an opening $a_1, \ldots, a_n, r$ of $\boldsymbol{c}_a$ such that $b \neq \prod_{i=1}^n a_i$. Using $\overline{b}_1 = \overline{a}_1$ and $x\overline{b}_i = \overline{b}_{i-1}\overline{a}_i + p_1(x)$ where $p_1(x)$ is a degree 1 polynomial

in x, it follows that the verification equations imply

$$x^n b = x^{n-1}\overline{b}_n = \prod_{i=1}^{n} \overline{a}_i + p_{n-1}(x),$$

where p_{n-1} is a fixed degree $n-1$ polynomial determined by the committed values. Since $\overline{a}_i = xa_i + d_i$ the Schwartz-Zippel lemma implies that there is negligible probability of satisfying this equation for random x unless indeed $b = \prod_{i=1}^{n} a_j$. Furthermore, the extracted openings are the same openings known by the prover. If this were not the case the prover could use the extractor to find another opening for at least one of the commitments, and the commitment scheme would be broken. □

Efficiency: The communication consist of 3 group elements and $2n$ field elements, so the communication complexity is $O(n)$.

The prover has to calculate $3n$ commitments; therefore, $3n$ exponentiations in $\mathbb{G}$ and a total number of $8n$ multiplications. Whereas the verifier only has to calculate $2n$ exponentiations in $\mathbb{G}$ and $4n$ multiplications. Thus, both parties have a computation cost of $O(n)$, since most exponentiations arise from commitments to vectors this cost can reduced to $O\left(\frac{n}{\log n}\right)$ applying multi-exponentiation techniques, see Section 4.3.

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 46 \rangle \subset \mathbb{Z}^*_{179}$, which has prime order $q = 89$.

The statement contains $\{\mathbb{G}, p, q\} = \{\langle 46 \rangle, 179, 89\}$, $ck = \{G_1, \ldots, G_4, H\} = \{46, 177, 161, 100, 72\}$, $c_a = 74$, the commitment to vector $\boldsymbol{b}$ from the product argument example, and of $b = 10$. The prover knows $\boldsymbol{a} = (58, 53, 85, 12)^T \in \mathbb{Z}^4_{179}$ and $r = 5$, such that $c_a = \mathrm{com}_{ck}(\boldsymbol{a}; r)$ and $b = \prod_{i=1}^{4} a_i$.

For the initial message the prover computes

$$b_1 = a_1 = 58, b_2 = a_1 \cdot a_2 = 48, b_3 = b_2 \cdot b_3 = 75, b_4 = b_3 \cdot a_4 = b = 10,$$

then picks $\boldsymbol{d} = (21, 18, 77, 10)^T$, $r_d = 4$ and calculates

$$c_d = \mathrm{com}_{ck}(\boldsymbol{d}; r_d) = 56.$$

Next the prover sets $\delta_1 = d_1 = 21, \delta_4 = 0$ and picks $\delta_2 = 32$,
$\delta_3 = 28, s_1 = 81, s_x = 2$ and computes commitments

$$c_\delta = \mathrm{com}_{ck}(-\delta_1 d_2, -\delta_2 d_3, -\delta_3 d_4, ; s_1) = \mathrm{com}_{ck}(0, 9, 10; 81) = 129$$

$$c_\Delta = \mathrm{com}_{ck}(\delta_2 - a_2\delta_1 - b_1 d_2, \delta_3 - a_3\delta_2 - b_2 d_3, \delta_4 - a_4\delta_3 - b_3 d_4; s_x) = \mathrm{com}_{ck}(36, 8, 1; 2) = 146.$$

The prover sends

$$c_d = 56 \qquad c_\delta = 129 \qquad c_\Delta = 146$$

to the verifier and receives the challenge $x = 52$. Then the prover calculates answers

$$\overline{\boldsymbol{a}} = x\boldsymbol{a} + \boldsymbol{d} = (12, 49, 16, 75)^T \qquad \overline{r} = xr + r_d = 28$$

$$\overline{b}_1 = xb_1 + \delta_1 = 11 \qquad \overline{b}_2 = 49 \qquad \overline{b}_3 = 16 \qquad \overline{b}_4 = 75$$

and returns these values to the verifier.

The verifier checks if the commitments belong to $\mathbb{G}$ and the answers are in $\mathbb{Z}_{89}$, and

$$c_a^x c_d = 77 = \mathrm{com}_{ck}(\overline{\boldsymbol{a}}; \overline{r}) \qquad (\checkmark)$$

$$c_\Delta^x c_\delta = 45 = \mathrm{com}_{ck}(x\overline{b}_2 - \overline{b}_1\overline{a}_2, x\overline{b}_3 - \overline{b}_2\overline{a}_3, x\overline{b}_4 - \overline{b}_3\overline{a}_4, x\overline{b}_5 - \overline{b}_4\overline{a}_5; \overline{s}) \qquad (\checkmark),$$

and checks if $b_1 = a_1$ and $xb = 75 = b_5$.

5.3.4 Single Value Product Argument For Secret b

The single value product argument above can be easily adopted to show that for given committed valued $\boldsymbol{a} \in \mathbb{Z}_q^n$ and committed value $b \in \mathbb{Z}_q$, it holds that $b = \prod_{i=1}^n a_i$.

Statement: $\{\mathbb{G}, p, q\},\ ck,\ c_a,\ c_b \in \mathbb{G}$

Prover's witness: $\boldsymbol{a} \in \mathbb{Z}_q^n, b, r, s \in \mathbb{Z}_q$ such that

$$c_a = \mathrm{com}_{ck}(\boldsymbol{a}; r) \qquad c_b = \mathrm{com}_{ck}(b; s) \qquad b = \prod_{i=1}^{n} a_i.$$

Initial message: Compute

1. $b_1 = a_1 \qquad b_2 = a_1 a_2 \qquad \ldots \qquad b_n = \prod_{i=1}^n a_i$
2. $c_d = \mathrm{com}_{ck}(\boldsymbol{d}; r_d)$ where $\boldsymbol{d} \leftarrow \mathbb{Z}_q^n$ and $r_d \leftarrow \mathbb{Z}_q$
3. $\delta_1 = d_1, \ldots, \delta_n \leftarrow \mathbb{Z}_q$, $s_1, t_1, t_x \leftarrow \mathbb{Z}_q$

$$c_{\delta_n} = \mathrm{com}_{ck}(\delta_n; s_1)$$

$$c_\delta = \mathrm{com}_{ck}(-\delta_1 d_2, \ldots, -\delta_{n-1} d_n; t_1)$$

$$c_\Delta = \mathrm{com}_{ck}(\delta_2 - a_2\delta_1 - b_1 d_2, \ldots, \delta_n - a_n\delta_{n-1} - b_{n-1} d_n; t_x)$$

Send: $c_d, c_{\delta_n}, c_\delta, c_\Delta$

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Compute

1. $\overline{\boldsymbol{a}} = x\boldsymbol{a} + \boldsymbol{d}$ and $\overline{r} = xr + r_d$
2. $\overline{\boldsymbol{b}} = x\boldsymbol{b} + \boldsymbol{\delta}$

3. $\overline{s} = xs + s_1 \qquad \overline{t} = xt_x + t_1$

Send: $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \overline{r}, \overline{s}, \overline{t}$

Verification: The verifier accepts if only if

1. $c_d, c_\delta, c_\Delta \in \mathbb{G}$
2. $\overline{r}, \overline{s} \in \mathbb{Z}_q \quad$ and $\quad \overline{\boldsymbol{a}}, \overline{\boldsymbol{b}} \in \mathbb{Z}_q^n$
3. $c_a^x c_d = \mathrm{com}_{ck}(\overline{a}; \overline{r}) \qquad c_\Delta^x c_\delta = \mathrm{com}_{ck}(x\overline{b}_2 - \overline{b}_1\overline{a}_2, \ldots, x\overline{b}_n - \overline{b}_{n-1}\overline{a}_n; \overline{t})$
4. $\overline{b}_1 = \overline{a}_1 \qquad c_b^x c_{\delta_n} = \mathrm{com}_{ck}(\overline{b}_n; \overline{s})$.

Theorem 23. *The protocol is a public coin perfect generalized $\Sigma-$protocol of an opening $a_1, \ldots, a_n, r$ such that $c_a = \mathrm{com}_{ck}(\boldsymbol{a}; r), c_b = \mathrm{com}_{ck}(b; s)$ and $b = \prod_{i=1}^n a_i$.*

Proof. Perfect completeness follows from

$$x\overline{b}_i - \overline{b}_{i-1}\overline{a}_i = x(xb_i + \delta_i) - (xb_{i-1} + \delta_{i-1})(xa_i + d_i) = x(\delta_i - \delta_{i-1}a_i - b_{i-1}d_i) - \delta_{i-1}d_i$$

for $i = 2, \ldots, n$ since $b_i = b_{i-1}a_i$.

We will now argue that we have perfect SHVZK. The simulator gets the challenge x and has to make a convincing transcript. It picks $\overline{r} \leftarrow \mathbb{Z}_q$, $\overline{\boldsymbol{a}} \leftarrow \mathbb{Z}_q^n$ and sets $c_d = \boldsymbol{c}_a^{-x}\mathrm{com}_{ck}(\overline{\boldsymbol{a}}; \overline{r})$. It picks at random s_x and sets $c_\Delta = \mathrm{com}_{ck}(0, \ldots, 0; s_x)$. It picks at random $\overline{b}_2, \ldots, \overline{b}_n, \overline{s}, \overline{t} \leftarrow \mathbb{Z}_q$, sets $\overline{b}_1 = \overline{a}_1$ and computes $c_\delta = c_\Delta^{-x}\mathrm{com}_{ck}(x\overline{b}_2 - \overline{b}_1\overline{a}_2, \ldots, x\overline{b}_n - \overline{b}_{n-1}\overline{a}_n; \overline{t})$ and $c_{\delta_n} = \mathrm{com}_{ck}(\overline{b}_n; \overline{s})c_b^{-x}$. To see this is a perfect simulation note that c_Δ and c_{δ_n} are perfectly hiding commitments just like in a real proof, and $\overline{a}_1, \ldots, \overline{a}_n, \overline{r}$, and $\overline{b}_1, \ldots, \overline{b}_n, \overline{s}, \overline{t}$ are distributed just like in a real proof. These choices uniquely determine $c_d, c_{\delta_n}, c_\delta$ according to the verification equations giving us that simulated and real proofs are identically distributed.

Finally, we will show that the protocol has perfect generalized special soundness. Given two accepting transcripts with challenges $x_1, x_2, x_1 \neq x_2$, we have

$$c_a^{x_1} c_d = \mathrm{com}_{ck}\left(\overline{a}_1^{(1)}, \ldots, \overline{a}_n^{(1)}; \overline{r}^{(1)}\right) \qquad c_a^{x_2} c_d = \mathrm{com}_{ck}\left(\overline{a}_1^{(2)}, \ldots, \overline{a}_n^{(2)}; \overline{r}^{(2)}\right),$$

which implies $c_a^{x_1 - x_2} = \mathrm{com}_{ck}\left(\overline{a}_1^{(1)} - \overline{a}_1^{(2)}, \ldots, \overline{a}_n^{(1)} - \overline{a}_n^{(2)}; \overline{r}^{(1)} - \overline{r}^{(1)}\right)$ giving us an opening of c_a. The equation $c_d = c_a^{-x}\mathrm{com}_{ck}(\overline{a}_1, \ldots, \overline{a}_n; \overline{r})$ then gives us an opening of c_d. Similarly, we can get openings of c_δ, c_Δ.

It remains to argue that there is negligible probability of extracting an opening $a_1, \ldots, a_n, b, r, s$ of $\boldsymbol{c}_a$ and c_b such that $b \neq \prod_{i=1}^n a_i$. Using $\overline{b}_1 = \overline{a}_1$ and $x\overline{b}_i = \overline{b}_{i-1}\overline{a}_i + p_1(x)$ where $p_1(x)$ is a degree 1 polynomial in x, it follows that the verification equations imply

$$x^n b = x^{n-1}\overline{b}_n = \prod_{i=1}^n \overline{a}_i + p_{n-1}(x),$$

where p_{n-1} is a fixed degree $n-1$ polynomial determined by the committed values. Since $\overline{a}_i = xa_i + d_i$ the Schwartz-Zippel lemma implies that there is negligible probability of satisfying this equation for random x unless indeed $b = \prod_{i=1}^{n} a_j$. Furthermore, the extracted openings are the same openings known by the prover. If this were not the case the prover could use the extractor to find another opening for at least one of the commitments, and the commitment scheme would be broken. □

Efficiency: The communication consists of 3 group elements and $2n$ field elements and therefore is $O(n)$.

The prover has to calculate $3n$ commitments; thus, $3n$ exponentiations in $\mathbb{G}$ and a total number of $8n$ multiplications. Whereas the verifier only has to calculate $2n$ exponentiations in $\mathbb{G}$ and $4n$ multiplications. It follows that both parties have a computation cost of $O(n)$, since most exponentiations arise from commitments to vectors this cost can reduced to $O\left(\frac{n}{\log n}\right)$ applying multi-exponentiation techniques, see section 4.3.

5.4 Hadamard Product Argument

In this section, we will give an argument of knowledge for committed vectors

$$\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{c}_1, \ldots, \boldsymbol{a}_m, \boldsymbol{b}_m, \boldsymbol{c}_m \in \mathbb{Z}_q^n$$

satisfying

$$\boldsymbol{a}_i \circ \boldsymbol{b}_i = \boldsymbol{c}_i,$$

which will be used in Section 6.4. Similar techniques to the multi Hadamard product in Section 5.3.1 can be used to give this argument but this requires 7 rounds of interaction. For more details please see Groth [Gro09]. Since we want to minimize the round complexity we will give an improved construction that only uses 3 rounds and has the same communication complexity as Groth's Hadamard product argument.

The work leading to this section was done together with Jens Groth, he had the idea underlying the new argument.

Following the construction of quadratic arithmetic programs in Section 4 of Gennaro et al. [GGPR13] we use Lagrange interpolation polynomials in our construction. Recall that given a set of values $\Omega = \{\omega_1, \ldots, \omega_m\} \subset \mathbb{Z}_q$ the Lagrange interpolation polynomials are

$$l_i(X) = \frac{\prod_{j \neq i}(X - \omega_j)}{\prod_{j \neq i}(\omega_i - \omega_j)} \quad \text{for } i = 1, \ldots, m.$$

We also define $l(X) = \prod_{j=1}^{m}(X - \omega_j)$. Our protocols will work no matter what the choice of $\omega_1, \ldots, \omega_m$ is as long as they are different but for efficiency purposes it may be useful to choose them as roots of unity. When they are roots of unity we may use the Fast Fourier Transform to speed up some of our

calculations. A key property satisfied by Lagrange polynomials is

$$l_i(X) = \begin{cases} 1 & \mod X - \omega_i \\ 0 & \mod \frac{l(X)}{X-\omega_i} \end{cases} \text{ for } i = 1, \ldots, m\ .$$

To explain the main concepts let us first focus on completeness and knowledge soundness for a moment, zero-knowledge is easy to add but we will do this later and ignore zero-knowledge right now. The idea is that the prover will demonstrate

$$\left(\sum_{i=1}^{m} l_i(X)\boldsymbol{a}_i\right) \circ \left(\sum_{j=1}^{m} l_j(X)\boldsymbol{b}_j\right) - \sum_{i=1}^{m} l_i(X)\boldsymbol{c}_i = \boldsymbol{\Delta}(X)l(X) \tag{5.2}$$

for some $\boldsymbol{\Delta(X)} \in (\mathbb{Z}_q[X])^n$. Knowledge soundness and completeness will follow from the fact that for all $k = 1, \ldots, m$ we have

$$\left(\sum_{i=1}^{m} l_i(X)\boldsymbol{a}_i\right) \circ \left(\sum_{j=1}^{m} l_j(X)\boldsymbol{b}_j\right) - \sum_{i=1}^{m} l_i(X)\boldsymbol{c}_i \equiv \boldsymbol{a}_k \circ \boldsymbol{b}_k - \boldsymbol{c}_k \mod X - \omega_k,$$

as $l_i(X) \equiv 1 \mod X - \omega_i$ and $l_i(X) \equiv 0 \mod X - \omega_k$ for $i \neq k$.

Since $l(X) \equiv 0 \mod X - \omega_k$ it is therefore only possible for (5.2) to hold if indeed $\boldsymbol{a}_k \circ \boldsymbol{b}_k - \boldsymbol{c}_k = 0$ for all $k = 1, \ldots, m$. This is what will show us that the extracted witness is valid in the argument of knowledge.

For completeness please observe that if $\boldsymbol{a}_k \circ \boldsymbol{b}_k - \boldsymbol{c}_k = 0$

$$\begin{aligned} \left(\sum_{i=1}^{m} l_i(X)\boldsymbol{a}_i\right) \circ \left(\sum_{j=1}^{m} l_j(X)\boldsymbol{b}_j\right) - \sum_{i=1}^{m} l_i(X)\boldsymbol{c}_i &\equiv 0 \mod X - \omega_1 \\ &\vdots \\ \left(\sum_{i=1}^{m} l_i(X)\boldsymbol{a}_i\right) \circ \left(\sum_{j=1}^{m} l_j(X)\boldsymbol{b}_j\right) - \sum_{i=1}^{m} l_i(X)\boldsymbol{c}_i &\equiv 0 \mod X - \omega_m. \end{aligned}$$

Since $\omega_i \neq \omega_j$ for $i \neq j$ all the $X - \omega_i$ are coprime and therefore the Chinese Remainder Theorem tells us that

$$\left(\sum_{i=1}^{m} l_i(X)\boldsymbol{a}_i\right) \circ \left(\sum_{j=1}^{m} l_j(X)\boldsymbol{b}_j\right) - \sum_{i=1}^{m} l_i(X)\boldsymbol{c}_i \equiv 0 \mod l(X).$$

This means $\boldsymbol{\Delta}(X)$ exists and the prover can compute it, which will give us completeness.

The next step is to explain how the prover can demonstrate that (5.2) holds. The prover will send commitments $c_{\Delta_0}, \ldots, c_{\Delta_m}$ to $\boldsymbol{\Delta}_i \in \mathbb{Z}_q^n$ such that $\boldsymbol{\Delta}(X) = \sum_{i=0}^{m} \boldsymbol{\Delta}_i X^i$. On random challenge

$x \leftarrow \mathbb{Z}_q^* \setminus \Omega$ the prover opens

$$\prod_{i=1}^{m} c_{a_i}^{l_i(x)} \qquad \prod_{i=1}^{m} c_{b_i}^{l_i(x)} \qquad \prod_{i=1}^{m} c_{c_i}^{l_i(x)} \qquad \prod_{i=0}^{m} c_{\Delta_i}^{x^i}$$

to

$$\overline{\boldsymbol{a}} = \sum_{i=1}^{m} \boldsymbol{a}_i l_i(x) \qquad \overline{\boldsymbol{b}} = \sum_{i=1}^{m} \boldsymbol{b}_i l_i(x) \qquad \overline{\boldsymbol{c}} = \sum_{i=1}^{m} \boldsymbol{u}_i l_i(x) \qquad \overline{\boldsymbol{\Delta}} = \sum_{i=0}^{m} \boldsymbol{\Delta}_i x^i.$$

The verifier can now check

$$\overline{\boldsymbol{a}} \circ \overline{\boldsymbol{b}} - \overline{\boldsymbol{c}} = \overline{\boldsymbol{\Delta}} l(x),$$

which has negligible probability over $x \leftarrow \mathbb{Z}_q$ unless (5.2) holds.

To make the whole argument zero-knowledge we have to avoid $\overline{\boldsymbol{a}}$, $\overline{\boldsymbol{b}}$, and $\overline{\boldsymbol{c}}$ leaking any information. The prover therefore picks $\boldsymbol{a}_0, \boldsymbol{b}_0, \boldsymbol{c}_0 \leftarrow \mathbb{Z}_q^n$ at random and defines

$$\overline{\boldsymbol{a}} = \boldsymbol{a}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{a}_i l_i(x) \qquad \overline{\boldsymbol{b}} = \boldsymbol{b}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{b}_i l_i(x) \qquad \overline{\boldsymbol{c}} = \boldsymbol{c}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{c}_i l_i(x).$$

As $\boldsymbol{a}_0, \boldsymbol{b}_0, \boldsymbol{c}_0$ are picked randomly these sums do not leak any information about the $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ unless $l(x) = 0$, which never happens as the challenges are not in Ω . We also have that $l(X) \equiv 0 \mod X - \omega_i$ for all $i = 1, \ldots, m$ so $\boldsymbol{\Delta}(X) \in (\mathbb{Z}_q[X])^n$ exists such that we have $\overline{\boldsymbol{a}} \circ \overline{\boldsymbol{b}} - \overline{\boldsymbol{c}} = \overline{\boldsymbol{\Delta}}(X) l(X)$.

Statement: $\{\mathbb{G}, p, q\},\ ck = \{G_1, \ldots, G_n, H\},\ c_{a_1}, c_{b_1}, c_{c_1}, \ldots, c_{a_1}, c_{b_1}, c_{c_m} \in \mathbb{G}$

Prover's witness: $\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{c}_1, \ldots, \boldsymbol{a}_m, \boldsymbol{b}_m, \boldsymbol{c}_m \in \mathbb{Z}_q^n$ and $r_1, s_1, t_1, \ldots, r_m, s_m, t_m \in \mathbb{Z}_q$ such that

$$\boldsymbol{a}_i \circ \boldsymbol{b}_i = \boldsymbol{c}_i \qquad c_{a_i} = \mathrm{com}_{ck}(\boldsymbol{a}_i; r_i) \qquad c_{b_i} = \mathrm{com}_{ck}(\boldsymbol{b}_i; s_i) \qquad c_{c_i} = \mathrm{com}_{ck}(\boldsymbol{c}_i; t_i)$$

Initial message: Compute

1. $c_{a_0} = \mathrm{com}_{ck}(\boldsymbol{a}_0; r_0)$ where $\boldsymbol{a}_0 \leftarrow \mathbb{Z}_q^n$ and $r_0 \leftarrow \mathbb{Z}_q$
2. $c_{b_0} = \mathrm{com}_{ck}(\boldsymbol{b}_0; s_0)$ where $\boldsymbol{b}_0 \leftarrow \mathbb{Z}_q^n$ and $s_0 \leftarrow \mathbb{Z}_q$
3. $c_{c_0} = \mathrm{com}_{ck}(\boldsymbol{c}_0; t_0)$ where $\boldsymbol{c}_0 \leftarrow \mathbb{Z}_q^n$ and $t_0 \leftarrow \mathbb{Z}_q$
4. $\boldsymbol{\Delta}_0, \ldots, \boldsymbol{\Delta}_m \in \mathbb{Z}_q^n$ such that

$$\left(\sum_{i=0}^{m} \boldsymbol{\Delta}_i X^i\right) l(X) = \left(\boldsymbol{a}_0 l(X) + \sum_{i=1}^{m} \boldsymbol{a}_i l_i(X)\right) \circ \left(\boldsymbol{b}_0 l(X) + \sum_{i=1}^{m} \boldsymbol{b}_i l_i(X)\right) - \left(\boldsymbol{c}_0 l(X) + \sum_{i=1}^{m} \boldsymbol{c}_i l_i(X)\right).$$

5. $c_{\Delta_0} = \mathrm{com}_{ck}(\boldsymbol{\Delta}_0; \rho_0), \ldots, c_{\Delta_m} = \mathrm{com}_{ck}(\boldsymbol{\Delta}_m; \rho_m)$ where $\rho_0, \ldots, \rho_m \leftarrow \mathbb{Z}_q$

Send: $c_{a_0}, c_{b_0}, c_{c_0}, c_{\Delta_0}, \ldots, c_{\Delta_m}$

Challenge: $x \leftarrow \mathbb{Z}_q^* \setminus \Omega$

Answer: Compute

$$\overline{\boldsymbol{a}} = \boldsymbol{a}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{a}_i l_i(x) \qquad \overline{\boldsymbol{b}} = \boldsymbol{b}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{b}_i l_i(x) \qquad \overline{\boldsymbol{c}} = \boldsymbol{c}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{c}_i l_i(x)$$

$$\overline{r} = r_0 l(x) + \sum_{i=1}^{m} r_i l_i(x) \qquad \overline{s} = s_0 l(x) + \sum_{i=1}^{m} s_i l_i(x) \qquad \overline{t} = t_0 l(x) + \sum_{i=1}^{m} t_i l_i(x)$$

$$\overline{\rho} = \left(\sum_{i=0}^{m} x^i \rho_i \right) l(x)$$

Send: $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \overline{\boldsymbol{c}}, \overline{r}, \overline{s}, \overline{t}, \overline{\rho}$

Verification: Accept if and only if

1. $c_{a_0}^{l(x)} \prod_{i=1}^{m} c_{a_i}^{l_i(x)} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}}; \overline{r}) \qquad c_{b_0}^{l(x)} \prod_{i=1}^{m} c_{b_i}^{l_i(x)} = \mathrm{com}_{ck}(\overline{\boldsymbol{b}}; \overline{s})$
2. $c_{c_0}^{l(x)} \prod_{i=1}^{m} c_{c_i}^{l_i(x)} = \mathrm{com}_{ck}(\overline{\boldsymbol{c}}; \overline{t}) \qquad \left(\prod_{i=0}^{m} c_{\Delta_i}^{x^i} \right)^{l(x)} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}} \circ \overline{\boldsymbol{b}} - \overline{\boldsymbol{c}}; \overline{\rho})$

Theorem 24. *The protocol above is a public coin generalized perfect $\Sigma-$protocol of committed vectors $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ such that $\boldsymbol{a}_i \circ \boldsymbol{b}_i = \boldsymbol{c}_i$.*

Proof. Perfect completeness by inspection, where it is useful to keep in mind that

$$l_i(X) l_j(X) \equiv 0 \mod X - \omega_k$$

unless $i = j = k$.

Given a challenge x such that $l(x) \neq 0$ the SHVZK simulator picks $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \overline{\boldsymbol{c}} \leftarrow \mathbb{Z}_q^m$, $\overline{r}, \overline{s}, \overline{t}, \overline{\rho} \leftarrow \mathbb{Z}_q$ at random and $c_{\Delta_1}, \ldots, c_{\Delta_m} \leftarrow \mathbb{G}$. It then sets

$$c_{a_0} = \left(\mathrm{com}_{ck}(\overline{\boldsymbol{a}}; \overline{r}) \prod_{i=1}^{m} c_{a_i}^{-l_i(x)} \right)^{l(x)^{-1}} \qquad c_{b_0} = \left(\mathrm{com}_{ck}(\overline{\boldsymbol{b}}; \overline{s}) \prod_{i=1}^{m} c_{b_i}^{-l_i(x)} \right)^{l(x)^{-1}}$$

$$c_{c_0} = \left(\mathrm{com}_{ck}(\overline{\boldsymbol{c}}; \overline{t}) \prod_{i=1}^{m} c_{c_i}^{-l_i(x)} \right)^{l(x)^{-1}} \qquad c_{\Delta_0} = \left(\mathrm{com}_{ck}(\overline{\boldsymbol{a}} \circ \overline{\boldsymbol{b}} - \overline{\boldsymbol{c}}; \overline{\rho}) \prod_{i=1}^{m} c_{\Delta_i}^{-x^i} \right)^{l(x)^{-1}}.$$

The simulated argument is indistinguishable from a real argument on the same challenge. Due to the random choice of $\boldsymbol{a}_0, \boldsymbol{b}_0, \boldsymbol{c}_0, r_0, s_0, t_0$, and ρ_0, $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \overline{\boldsymbol{c}}$, and $\overline{r}, \overline{s}, \overline{t}, \overline{\rho}$ are uniformly random in the real argument just as they are in the simulation. The commitments $c_{\Delta_1}, \ldots, c_{\Delta_m}$ are due to the perfect hiding property of the commitment scheme also uniformly random just as in the simulation. Finally, the $c_{a_0}, c_{b_0}, c_{c_0}$ are determined uniquely by the verification equations once the other variables are determined. Therefore, the simulated argument is indistinguishable from a real argument and we have SHVZK.

The last step is to show that we have generalized special soundness. Given $2m + 1$ accepting

transcripts with different challenges the extractor can construct matrix

$$M = \begin{pmatrix} l(x_1) & l_1(x_1) & \dots & l_m(x_1) \\ & & \vdots & \\ l(x_{m+1}) & l_1(x_{m+1}) & \dots & l_m(x_{m+1}) \end{pmatrix}$$

and M is invertible, see Appendix .1 for the proof. This means

$$\begin{pmatrix} \boldsymbol{a}_0 & r_0 \\ \vdots & \vdots \\ \boldsymbol{a}_m & r_m \end{pmatrix} = M^{-1} \begin{pmatrix} \overline{\boldsymbol{a}}^{(1)} & \overline{r}^{(1)} \\ \vdots & \vdots \\ \overline{\boldsymbol{a}}^{(m+1)} & \overline{r}^{(m+1)} \end{pmatrix}$$

gives us openings of $c_{a_0}, \dots, c_{a_m}$. We get openings of $c_{b_0}, c_{c_0}, c_{\Delta_0}, \dots, c_{b_m}, c_{c_m}, c_{\Delta_m}$ to $\boldsymbol{b}_0, \boldsymbol{c}_0, \boldsymbol{\Delta}_0, \dots, \boldsymbol{b}_m, \boldsymbol{c}_m, \boldsymbol{\Delta}_m$ respectively in a similar fashion.

The binding property of the commitment scheme means that for all $2m+1$ transcript the answers to the challenge x must be of the form

$$\overline{\boldsymbol{a}} = \boldsymbol{a}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{a}_i l_i(x), \qquad \overline{\boldsymbol{b}} = \boldsymbol{b}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{b}_i l_i(x), \qquad \overline{\boldsymbol{c}} = \boldsymbol{c}_0 l(x) + \sum_{i=1}^{m} \boldsymbol{c}_i l_i(x)$$
$$\overline{r} = r_0 l(x) + \sum_{i=1}^{m} r_i l_i(x), \qquad \overline{s} = s_0 l(x) + \sum_{i=1}^{m} s_i l_i(x), \qquad \overline{t} = t_0 l(x) + \sum_{i=1}^{m} t_i l_i(x)$$

$$\overline{\rho} = (\sum_{i=0}^{m} x^i \rho_i) l(x).$$

Since we have

$$\overline{\boldsymbol{a}} \circ \overline{\boldsymbol{b}} - \overline{\boldsymbol{c}} = \left(\sum_{i=0}^{m} \boldsymbol{\Delta}_i x^u \right) l(x)$$

for $2m+1$ choices of challenge x we deduce that we have two identical degree $2m$ polynomials

$$\overline{\boldsymbol{a}}(X) \circ \overline{\boldsymbol{b}}(X) - \overline{\boldsymbol{c}}(X) = \overline{\boldsymbol{\Delta}}(X) l(X).$$

This implies for all $k = 1, \dots, m$ that $\boldsymbol{a}_k \circ \boldsymbol{b}_k = \boldsymbol{c}_k \bmod X - \omega_k$, so the extracted vectors satisfy the Hadamard product conditions. Furthermore, the extracted openings are exactly the witnesses of the prover. If this is not the case, the extractor of the argument could be used by the prover to find a second opening of at least one of the commitments with non-negligible probability and the commitment scheme is broken. □

Efficiency. The prover sends m group elements and $3n$ field elements and so the total communication cost is $O(m+n)$ group and field elements.

The prover's cost is roughly mn exponentiations in $\mathbb{G}$ to compute the commitments plus the cost to calculate the $\boldsymbol{\Delta}_i$s.

The $\mathbf{\Delta}_i$s are defined such that they satisfy the equation

$$\left(\sum_{i=0}^{m} \mathbf{\Delta}_i X^i\right) l(X) = \left(\boldsymbol{a}_0 l(X) + \sum_{i=1}^{m} \boldsymbol{a}_i l_i(X)\right) \circ \left(\boldsymbol{b}_0 l(X) + \sum_{i=1}^{m} \boldsymbol{b}_i l_i(X)\right) - \left(\boldsymbol{c}_0 l(X) + \sum_{i=1}^{m} \boldsymbol{c}_i l_i(X)\right).$$

A straightforward calculation of the $\mathbf{\Delta}_i$s would cost $O(nm^2)$ multiplications in $\mathbb{Z}_q$. However, if $\omega_1, \ldots, \omega_m$ are roots of unity, we may use FFT to reduce the cost to $O(nm \log m)$.

The plan is to evaluate $\sum_{i=0}^{m} \mathbf{\Delta}_{\boldsymbol{i}} X^i$ in $m+1$ different points and use Lagrange interpolation to get the $\mathbf{\Delta}_i$s. We can use the fact that $l_i(\omega_k) = \delta_{ik}$, where δ_{ik} is Kronecker's delta, to get a fast evaluation of $l(\omega_k) + \sum_{i=1}^{m} \boldsymbol{a}_i l_i(\omega_k) = \boldsymbol{a}_k$. We can now use FFT to get evaluation in additional $m+1$ points in $O(nm \log m)$ multiplications. In a similar way, we can get $2m+1$ evaluation points for $\boldsymbol{b}_0 l(X) + \sum_{i=1}^{m} \boldsymbol{b}_i l_i(X)$ and $\boldsymbol{c}_0 l(X) + \sum_{i=1}^{m} \boldsymbol{c}_i l_i(X)$. Multiplying them together gives us $2m+1$ evaluation points for the right hand side, and dividing with the $m+1$ points for which $l(X) \neq 0$ gives us $m+1$ evaluation points of $\sum_{i=0}^{m} \mathbf{\Delta}_i X^i$. This gives a total of $O(mn \log m)$ multiplications in $\mathbb{Z}_q$.

The cost of the $O(mn)$ exponentiations is the dominant part of the prover's computation but can be further reduced by multi-exponentiation techniques to the equivalent of $O\left(\frac{mn}{\log n}\right)$ single exponentiations.

The verifier's computation is $4m + 4n$ exponentiation and $4m + 4n$ multiplications in $\mathbb{G}$. So for both parties the computations cost is $O(m+n)$, but the number of exponentiations can be reduced to $O(\frac{m}{\log m} + \frac{n}{\log n})$ using the techniques described in Section 4.3.

Chapter 6

Zero-Knowledge Polynomial Arguments

The theoretical part of Section 6.2 was published at Eurocrypt 2013 [BG13] together with Jens Groth. Also, Section 6.4 is joint work with Jens Groth, but as yet unpublished. Furthermore, Section 6.3 and most of the practical work of Section 6.2 is unpublished so far. Lastly, in Section 6.5 we recap Brands et al.'s [BDD07] ideas, and also discuss results based on our implementation of their protocol.

6.1 Introduction

In many cryptographic applications a party wants to prove possession of a secret value u that fulfills a certain property. Since polynomials are widely used a natural question is for instance given a polynomial $P(X)$ and a value v whether the secret u satisfies $P(u) = v$ in prime order field $\mathbb{Z}_q$. Applications for such arguments are for instance non-membership proofs or membership proofs, see Section 7.2, possession of a digital signature, or electronic cash protocols. Section 6.2 and some other small parts of this chapter were published at Eurocrypt 2013 [BG13].

We propose a special honest verifier zero-knowledge argument of knowledge for two committed values u, v satisfying $P(u) = v$ for a given polynomial $P(X) \in \mathbb{Z}_q[X]$ of degree D, where q is prime. Given the coefficients of the polynomial $P(X) = \sum_{i=0}^{D} a_i X^i$ and two Pedersen commitments our zero-knowledge argument can demonstrate knowledge of openings of the Pedersen commitments to values u and v such that $P(u) = v$.

The argument is a perfect Σ-protocol: the prover sends a message, the verifier picks a challenge uniformly at random from $\mathbb{Z}_p$, and the prover answers the challenge. It has perfect completeness, perfect special honest verifier zero-knowledge, perfect generalized special soundness, and computational soundness, which is based on the discrete logarithm assumption in $\mathbb{G}$.

Our polynomial evaluation argument is highly efficient. The communication complexity is $O(\log D)$ group and field elements, which is very small compared to the statement size of D field elements. Furthermore, former techniques based on the discrete logarithm achieved square root communication complexity at best [BDD07, Gro09].

The prover computes $O(\log D)$ exponentiations and $O(D \log D)$ multiplications in $\mathbb{Z}_q$, and the verifier calculates $O(\log D)$ exponentiations and $O(D)$ multiplications in $\mathbb{Z}_q$. The hidden constants in the expressions are small and the argument very efficient in practice as illustrated by a concrete implementa-

tion. Again, this is an improvement to earlier techniques which achieved at best square root complexity for both parties [BDD07].

We also show how this technique can be adjusted to work with multi-variate polynomials $P(X_1, \ldots, X_n)$. Our approach communicates only $O((\log D)^n)$ elements, where D is the maximum degree of each variable X_i, which is small compared to the statement size of D^n field elements. Both parties have to compute $O((\log D)^n)$ exponentiations. Similar to our single-variate polynomial argument our protocol has a standard 3-round structure.

Next, we will explain how one can prove correctness of evaluation of L polynomials $P_i(X)$ at the same time. All former approaches [FO97, Bra97, CS97, BDD07, CD98, Gro09, Gro11, KZG10] would lead to L times the original cost, as the argument has to be repeated L times in parallel. Repeating our new polynomial argument in parallel would lead to $O(L \log D)$ cost, but our actual technique has the reduced cost of $O(\sqrt{L} \log D)$, which improves the state of the art by a big step. The verifier has to compute $O(\sqrt{L} \log D)$ exponentiations instead of expected $O(L \log D)$ exponentiations and the new technique requires only 3 rounds of interaction.

To confirm our theoretical findings we implemented our polynomial argument and the polynomial argument based on Brands et al. [BDD07], since this argument performs best of all earlier arguments based on the discrete logarithm.

As expected our communication cost is very small, consisting only of a few kilobytes, compared to Brands et al.'s communication which consists of a few megabytes for big degree polynomials. Our verifier runs faster than Brands et al.'s verifier, and our prover is more efficient for reasonable degree D. However, for big size D our computation time for the whole protocol is faster.

6.1.1 Techniques

In many multi-exponentiation techniques the exponents a_i are written in binary, which leads to improved performance. We will apply this idea to reduce the cost of our polynomial evaluation argument to logarithmic cost. In more detail, we will rewrite the terms X^i as $X^i = \prod_{j=0}^{d} (X^{2^j})^{i_j}$, where $i = i_0 \ldots i_d$ in binary.

The trick allows us to commit only to $\log D = d$ values $u, u^2, u^4, \ldots u^D$, by using sophisticated combinations of these values combined with the homomorphic properties of the commitment scheme we get the desired argument for $P(u) = v$. This reduces our communication to $O(\log(D))$ group elements, which is a huge improvement over former work.

Unfortunately, this reduction suffers from a high computation complexity. The bottleneck is to calculate a new polynomial $Q(X)$, which is expensive to calculate. However, calculation the polynomial in a binary tree fashion reduced the complexity and the performance becomes efficient for medium range parameters.

To prove correctness of many polynomial $P^{(1)}(X), \ldots, P^{(L)}(X)$ at the same time we will use batch verification to reduce the communication cost. We will arrange the polynomials in a $m \times n$ matrix, $L = mn$, and show the correctness of n polynomials at the same time using Lagrange interpolation polynomials [GGPR13] and the length reducing property of the general Pedersen commitment.

6.1.2 Former Work

Given two committed values u, v we give a zero-knowledge argument that $P(u) = v$ for a public polynomial $P(X)$ of degree D. Kilian [Kil92] gave a communication efficient argument for circuit satisfiability and several other general purpose zero-knowledge arguments for NP-languages exist [IKOS07, GKR08]. However, since these arguments are not tailored for the discrete logarithm setting using them would require a costly NP-reduction.

Fujisaki and Okamoto [FO97] looked at polynomial evaluation in a RSA-based context but their zero-knowledge argument had linear complexity in the degree of the polynomial and both parties have to perform a linear number of exponentiations. The general techniques by Brands [Bra97], Camenisch and Stadler [CS97] are based on the discrete logarithm, but yield likewise linear complexity for the communication and the computation.

The most efficient zero-knowledge argument for polynomial evaluation so far stems from Brands et al. [BDD07] non-membership proof, which is also based on the discrete logarithm assumption. The work has a communication complexity of $O(\sqrt{D})$ where D is the number of elements in the set and both parties can perform in $O(\sqrt{D})$ time. The conversion in a polynomial argument add only cheap integer product arguments and therefore the asymptotic complexity is the same for the polynomial argument.

Kate et al.'s report in [KZG10] on a new commitment scheme for polynomials based on pairings. They show how to prove evaluation of secret polynomials in a public know value x, but it is also possible to convert their argument in a general polynomial evaluation argument. This argument is very light on the communication, sending only a few elements, however the computation complexity is $O(D)$ exponentiations for both parties.

In the prime order groups setting there are already several general zero-knowledge techniques for the satisfiability of arithmetic circuits that can demonstrate the correctness of a polynomial evaluation. Using Cramer and Damgård [CD98] we would get a linear communication complexity and linear computation for this problem. Using Groth [Gro09] we would get a communication complexity of $O(\sqrt{D})$ group elements. The verifier only need to perform $O(\sqrt{D})$ number of exponentiations, but the prover has to commit themselves to all gates which cost a linear number of exponentiations.

Using stronger assumptions and a pairing-based argument by Groth [Gro11] we could reduce the communication cost down to $O(D^{\frac{1}{3}})$ group elements. The verifier only needs to calculate $O(D^{\frac{1}{3}})$ exponentiations during the verification. But again the prover has to commit themselves to the values in each gate, which cost $O(D)$ commitment operations for the prover. These operations are either exponentiations or calculation of a pairing.

We can see that all former solutions suffer on at least one aspect. They have either high computation complexity for the prover, high computation for the verifier, or high communication complexity, or suffer from all these flaws. Kates et al.'s [KZG10] protocol has lowest communication complexity so far, and also Groth's [Gro11] protocol has low communication cost, but both protocols suffer from high computation and are based on stronger assumption, which are not as well studied as the discrete logarithm assumption. Brands et al.'s [BDD07] technique is based on this assumption and achieve the

best asymptotic computation cost for both parties so far and still has a low communication of $O(\sqrt{D})$ elements.

6.2 Polynomial Evaluation Argument with Logarithmic Cost

Given a polynomial $P(X) = \sum_{i=0}^{D} a_i X^i$ and two commitments c_{u_0}, c_v, we will describe an argument of knowledge of openings of the commitments to values u and v such that $P(u) = v$. The notation c_{u_0} for the commitment to $u = u^{2^0}$ matches other commitments c_{u_j} to u^{2^j} that the prover will construct in the argument.

By padding with zero-coefficients we can without loss of generality assume $D = 2^{d+1} - 1$. It is useful to write i in binary, i.e. $i = i_0 \dots i_d$ where $i_j \in \{0, 1\}$. We can then rewrite the term X^i as

$$X^i = X^{\sum_{j=0}^{d} i_j 2^j} = \prod_{j=0}^{d} \left(X^{2^j}\right)^{i_j}.$$

Substituting this in the polynomial we get

$$P(X) = \sum_{i=0}^{D} a_i X^i = \sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} \left(X^{2^j}\right)^{i_j}.$$

The prover will commit themselves to $u^2, u^4, \dots, u^{2^d}$ and prove that when inserted into the rewritten polynomial we have

$$\sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} \left(u^{2^j}\right)^{i_j} = v.$$

Since $d = \lfloor \log D \rfloor$ the prover only makes a logarithmic number of commitments, which will help to keep the communication cost low. Standard techniques can be used to give arguments of knowledge that the commitments $c_{u_1}, \dots, c_{u_d}$ to $u^{2^1}, \dots, u^{2^d}$ are well-formed and indeed contain the correct powers of u.

To show the committed powers of u in $c_0, c_1, \dots, c_d$ evaluate to the committed v the prover picks random values $f_0, \dots, f_d \leftarrow \mathbb{Z}_p$ and defines a new polynomial

$$Q(X) = \sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} (X u^{2^j} + f_j)^{i_j} X^{1-i_j} = X^{d+1} P(u) + X^d \delta_d + \dots + X\delta_1 + \delta_0.$$

The idea behind this choice of $Q(X)$ is that for each i_j either an $X u^{2^j}$ factor is included or an X factor is included, hence $P(u)$ is the coefficient of X^{d+1}. Each f_j on the other hand is not multiplied by X and will therefore only affect the lower degree coefficients $\delta_0, \dots, \delta_d$ of $Q(X)$.

The prover will now demonstrate that the coefficient of X^{d+1} in the secret $Q(X)$ is the same as v in a way that cancels out the $\delta_0, \dots, \delta_d$ coefficients. The prover sends the verifier commitments $c_{f_0}, \dots, c_{f_d}$ to $f_0, \dots, f_d$ and commitments $c_{\delta_0}, \dots, c_{\delta_d}$ to $\delta_0, \dots, \delta_d$. Afterwards, the verifier will pick a random challenge $x \leftarrow \mathbb{Z}_q^*$. The prover will now open suitable products of the commitments in a way such that

the verifier can check that the committed values u, v satisfy

$$Q(x) = x^{d+1}v + x^d\delta_d + \ldots + \delta_0.$$

More precisely, after receiving the challenge x the prover opens each product $c_j^x c_{f_j}$ to $\overline{f_j} = xu^{2^j} + f_j$.

The prover opens

$$c_v^{x^{d+1}} \prod_{j=0}^{d} c_{\delta_j}^{x^j}$$

to

$$\overline{\delta} = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j}.$$

Note that the verifier can calculate $\overline{\delta}$ themselves and therefore only accepts the opening if

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j} = x^{d+1}v + x^d\delta_d + \ldots + x\delta_1 + \delta_0.$$

This has negligible probability of being true unless $P(u) = v$.

Returning to the commitments $c_1, \ldots, c_d$ to $u^{2^1}, \ldots, u^{2^d}$ we said the prover could use standard techniques to show that the commitments contain the correct powers of u. To do this the prover sends some commitments c_{fu_j} to $f_j u^{2^j}$ to the verifier and later opens the commitments $c_{u_{j+1}}^x c_{u_j}^{-\overline{f_j}} c_{fu_j}$ to

$$xu^{2^{j+1}} - (xu^{2^j} + f_j)u^{2^j} + f_j u^{2^j} = 0.$$

The full polynomial evaluation argument is given below.

Statement: $\{\mathbb{G}, p, q\},\ ck,\ c_{u_0}, c_v \in \mathbb{G}$ and

$$P(X) = \sum_{i=0}^{D} a_i X^i = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} (X^{2^j})^{i_j} \in \mathbb{Z}_q[X].$$

Prover's witness: $u, v, r_0, t \in \mathbb{Z}_q$ such that

$$c_{u_0} = \mathrm{com}_{ck}(u; r_0) \qquad c_v = \mathrm{com}_{ck}(v; t) \qquad P(u) = v$$

Initial message: Compute

1. $c_{u_1} = \mathrm{com}_{ck}(u^{2^1}; r_1) \quad \ldots \quad c_{u_d} = \mathrm{com}_{ck}(u^{2^d}; r_d)$ where $r_1, \ldots, r_d \leftarrow \mathbb{Z}_q$
2. $c_{f_0} = \mathrm{com}_{ck}(f_0; s_0) \quad \ldots \quad c_{f_d} = \mathrm{com}_{ck}(f_d; s_d)$ where $f_0, s_0, \ldots, f_d, s_d \leftarrow \mathbb{Z}_q$
3. $\delta_0 \quad \ldots \quad \delta_d \in \mathbb{Z}_q$ such that

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j} = X^{d+1}v + \sum_{j=0}^{d} X^j\delta_j$$

4. $c_{\delta_0} = \mathrm{com}_{ck}(\delta_0; t_0), \ldots, c_{\delta_d} = \mathrm{com}_{ck}(\delta_d; t_d)$ where $t_0, \ldots, t_d \leftarrow \mathbb{Z}_q$
5. $c_{fu_0} = \mathrm{com}_{ck}(f_0 u^{2^0}; \xi_0) \quad \ldots \quad c_{fu_{d-1}} = \mathrm{com}_{ck}(f_{d-1} u^{2^{d-1}}; \xi_{d-1})$ where $\xi_0, \ldots, \xi_{d-1} \leftarrow \mathbb{Z}_q$

Send: $c_{u_1}, \ldots, c_{u_d}, c_{f_0}, \ldots, c_{f_d}, c_{\delta_0}, \ldots, c_{\delta_d}, c_{fu_0}, \ldots, c_{fu_{d-1}}$

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Compute for all j

1. $\overline{f}_j = xu^{2^j} + f_j \qquad \overline{r}_j = xr_j + s_j$
2. $\overline{t} = x^{d+1}t + \sum_{i=0}^{d} t_i x^i$
3. $\overline{\xi}_j = xr_{j+1} - \overline{f}_j r_j + \xi_j$

Send: $\overline{f}_0, \overline{r}_0, \ldots, \overline{f}_d, \overline{r}_d, \overline{t}, \overline{\xi}_0, \ldots, \overline{\xi}_{d-1}$

Verification: Accept if and only if for all j

1. $c_{u_j}^x c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j; \overline{r}_j)$
2. $c_{u_{j+1}}^x c_{u_j}^{-\overline{f}_j} c_{fu_j} = \mathrm{com}_{ck}(0; \overline{\xi}_j)$
3. $c_v^{x^{d+1}} \prod_{i=0}^{d} c_{\delta_i}^{x^i} = \mathrm{com}_{ck}\left(\overline{\delta}; \overline{t}\right)$ with

$$\overline{\delta} = \sum_{i_0, \ldots, i_d = 0}^{1} a_{i_0 \ldots i_d} \prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j}$$

Theorem 25. *Assuming the discrete logarithm assumption holds the polynomial evaluation argument is a public coin perfect generalized $\Sigma-$protocol of openings of c_{u_0} and c_v to u and v such that $P(u) = v$.*

Proof. Perfect completeness follows by careful inspection.

We will now argue that we have perfect SHVZK. On challenge $x \in \mathbb{Z}_q^*$ the simulator picks $c_{u_1}, \ldots, c_{u_d}, c_{\delta_1}, \ldots, c_{\delta_d} \leftarrow \mathbb{G}$ and $\overline{f}_0, \overline{r}_0, \ldots, \overline{f}_d, \overline{r}_d, \overline{t}, \overline{\xi}_0, \ldots, \overline{\xi}_{d-1} \leftarrow \mathbb{Z}_q$ and sets for all j

$$c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j; \overline{r}_j) c_{u_j}^{-x} \qquad c_{fu_j} = \mathrm{com}_{ck}(0; \overline{\xi}_j) c_{u_{j+1}}^{-x} c_{u_j}^{\overline{f}_j}$$

and

$$c_{\delta_0} = \mathrm{com}_{ck}\left(\sum_{i_0, \ldots, i_d = 0}^{1} a_{i_0 \ldots i_d} \prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j}; \overline{t} \right) c_v^{-x^{d+1}} \prod_{i=1}^{d} c_{\delta_i}^{-x^i}.$$

This is a perfect simulation. In a real argument $c_{u_1}, \ldots, c_{u_d}, c_{\delta_1}, \ldots, c_{\delta_d}$ are uniformly random perfectly hiding commitments as in the simulation. In a real argument $\overline{f}_0, \overline{r}_0, \ldots, \overline{f}_d, \overline{r}_d, \overline{t}, \overline{\xi}_0, \ldots, \overline{\xi}_{d-1} \in \mathbb{Z}_q$ are also uniformly random because of the random choice of $f_0, r_0, \ldots, f_d, r_d, t_0, \xi_0, \ldots, \xi_{d-1}$. Finally, both in the simulation and in the real argument these choices through the verification equations uniquely determine the values of $c_{f_0}, \ldots, c_{f_d}, c_{\delta_0}$ and $c_{fu_0}, \ldots, c_{fu_{d-1}}$. This means simulated and real arguments have identical probability distributions.

Finally, we have to show that we have perfect generalized special soundness. Given $d+2$ accepting transcripts with different challenges x_i the extractor E can extract witnesses. Given $\overline{f}_j^{(1)}, \overline{r}_j^{(1)}$ and $\overline{f}_j^{(2)}, \overline{r}_j^{(2)}$ in the first two answers to challenges x_1 and x_2 the extractor can take linear combinations of the verification equations to get openings of the commitments c_{u_j}. More precisely, we have that the two answers satisfy

$$c_{u_j}^{x_1} c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j^1; \overline{r}_j^1) \qquad\qquad c_{u_j}^{x_2} c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j^2; \overline{r}_j^2).$$

Picking α_1, α_2 such that $\alpha_1 x_1 + \alpha_2 x_2 = 1$ and $\alpha_1 + \alpha_2 = 0$ gives us

$$c_{u_j} = c_{u_j}^{\alpha_1 x_1 + \alpha_2 x_2} c_{f_j}^{\alpha_1 + \alpha_2} = \mathrm{com}_{ck}(\alpha_1 \overline{f}_j^{(1)} + \alpha_2 \overline{f}_j^{(2)}; \alpha_1 \overline{r}_j^{(1)} + \alpha_2 \overline{r}_j^{(2)}),$$

which is an opening of c_{u_j}.

Other types of linear combinations of the verification equations give us openings of the other commitments c_{f_j}, c_{fu_j}, c_v and c_{δ_i} the prover sends in the initial message. In the case of c_{δ_i} we find the linear combination as follows. Let

$$M = \begin{pmatrix} 1 & x_1 & \dots & x_1^{d+1} \\ \vdots & & & \vdots \\ 1 & x_{d+2} & \dots & x_{d+2}^{d+1} \end{pmatrix}.$$

Since it is a Vandermonde matrix with different $x_1, \dots, x_{d+2}$ it is invertible. By taking linear combinations of the verification equations

$$c_v^{x^{d+1}} \prod_{i=0}^{d} c_{\delta_i}^{x^i} = \mathrm{com}_{ck}\left(\sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} \overline{f}_j^{i_j} x^{1-i_j}; \overline{t} \right)$$

for different challenges $x_1, \dots, x_{d+2}$ we get that

$$\begin{pmatrix} \delta_0 & t_0 \\ \vdots & \vdots \\ \delta_d & t_d \\ v & s \end{pmatrix} = M^{-1} \begin{pmatrix} \sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} (\overline{f}_j^{(1)})^{i_j} x_1^{1-i_j} & \overline{t}_{(1)} \\ \vdots & \vdots \\ \sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} (\overline{f}_j^{(d+2)})^{i_j} x_{d+2}^{1-i_j} & \overline{t}_{(d+2)} \end{pmatrix}$$

which gives us openings of $c_{\delta_0}, \dots, c_{\delta_d}, c_v$.

We now have openings to all the commitments. Because the commitments are binding, each answer must be computed as they are by an honest prover in the argument. Therefore, the verification equations

$$c_j^x c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j, \overline{r}_j)$$

give us $\overline{f}_j = x u_j + f_j$, where u_j is the extracted value in c_j and f_j is the extracted value in c_{f_j}. The verification equations

$$c_{u_{j+1}}^x c_{u_j}^{-\overline{f}_j} c_{fu_j} = \mathrm{com}_{ck}(0; \overline{\xi}_j)$$

now give us that the committed values satisfy

$$xu_{j+1} - (xu_j + f_j)u_j + \phi_j = 0$$

for $j = 0, \ldots, d-1$ with u_j being the value inside c_j and ϕ_j being the value inside c_{fu_j}. Since each of the polynomial equalities is of degree 1 and holds for $d+2$ different challenges we see that $u_{j+1} = u_j u_j$. Since $u_0 = u$ this gives us $u_1 = u^{2^1}, u_2 = u^{2^2}, \ldots, u_d = u^{2^d}$.

Turning to the verification equation

$$c_v^{x^{d+1}} \prod_{i=0}^{d} c_{\delta_i}^{x^i} = \mathrm{com}_{ck}\left(\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0 \ldots i_d} \prod_{j=0}^{d} \overline{f}_j^{i_j} x^{1-i_j}; \overline{t} \right)$$

we now have that this corresponds to the degree $d+1$ polynomial equation

$$X^{d+1}v + X^d\delta_d + \ldots + X\delta_1 + \delta_0 = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0 \ldots i_d} \prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j}.$$

With $d+2$ different values $x_1, \ldots, x_{d+2}$ satisfying the equation, we conclude the two polynomials are identical. Looking at the coefficient for X^{d+1} we conclude that the extracted openings of c_{u_0} and c_v satisfy $P(u) = v$. Furthermore, the extracted openings u, v are the same openings known by the prover. If this were not the case the prover could use the extractor to find another opening for at least one of the commitments, and the commitment scheme would be broken. □

Efficiency: The communication consists of $4d$ group elements and $3d$ field elements.

The prover needs $8d$ exponentiations to compute the commitments and has to calculate the δ_is. These values are defined to satisfy

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0 \ldots i_d} \prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j} = X^{d+1}v + \sum_{j=0}^{d} X^j \delta_j.$$

The prover can calculate the degree d polynomials $\prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j}$ in a binary-tree fashion for all choices of $i_0 \ldots, i_d \in \{0,1\}$ at a cost of dD multiplications in $\mathbb{Z}_q$. Multiplying with the $a_{i_0 \ldots i_d}$'s uses another dD multiplications. The total cost for the prover is therefore $8d$ exponentiations in $\mathbb{G}$ and $2dD$ multiplications in $\mathbb{Z}_q$, plus $4d$ multiplications in $\mathbb{G}$.

The verifier can check the argument using $6d$ exponentiations in $\mathbb{G}$ since the exponent x is used twice in the verification equations and can compute the sum

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_d \ldots i_0} \prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j},$$

in a binary tree fashion for all choices of $i_0, \ldots, i_d \in \{0,1\}$ using $2D$ multiplications in $\mathbb{Z}_q$.

We have ignored small constants in the calculations above and just focused on the dominant terms. Using the sliding window technique, from Section 4.3.2, we can reduce the computational burden of the

exponentiations. Applying randomized verification and other tricks [BGR98, Gro10] it is possible to reduce the computation even further for the prover and verifier, so the estimates we have given above are quite conservative.

Example: Let $\mathbb{G} = \langle G = 172 \rangle \subset \mathbb{Z}^*_{179}$, which has prime order $q = 89$ and let $H = 74$. The statement consists of the polynomial $P(X) = X^4 + 46X^3 + 18X^2 + 2X + 81 \in \mathbb{Z}_{89}[X]$ and commitments

$$c_{u_0} = 142, c_v = 139,$$

and $\{\mathbb{G}, p, q\} = \{\langle 172 \rangle, 179, 89\}, ck = \{G, H\} = \{172, 74\}$. We have $d = \lfloor \log 4 \rfloor = 2$.

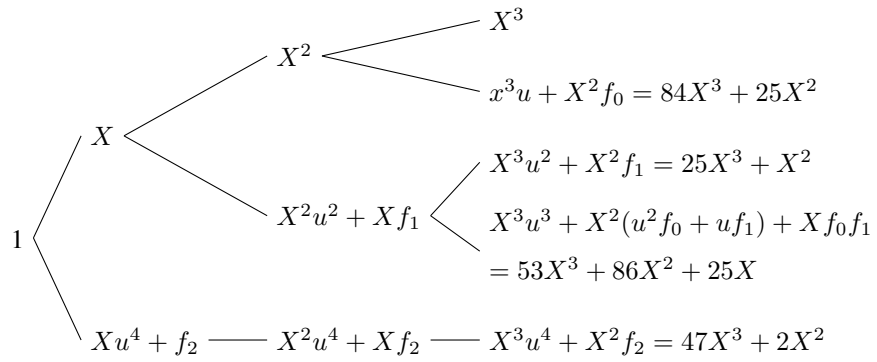
The prover knows values $u = 84, v = P(u) = 24 \in \mathbb{Z}_{89}$ and $r_0 = 15, t = 43 \in \mathbb{Z}_{89}$ such that $c_{u_0} = G^u H^{r_0} = 142 \in \mathbb{G}$ and $c_v = \mathrm{com}_{ck}(v; t) = 139 \in \mathbb{G}$. To prove the knowledge of the witnesses the prover first picks $r_1 = 16$ and $r_2 = 80$ at random from $\mathbb{Z}_{89}$ and computes

$$c_{u_1} = \mathrm{com}_{ck}(u^2; r_1) = 1 \qquad c_{u_2} = \mathrm{com}_{ck}(u^3; r_2) = 142.$$

The prover also picks $f_0 = 25, f_1 = 1, f_2 = 47, s_0 = 52, s_1 = 44, s_2 = 57$ randomly from $\mathbb{Z}_{89}$ and sets

$$c_{f_0} = \mathrm{com}_{ck}(f_0; s_0) = 142 \qquad c_{f_1} = \mathrm{com}_{ck}(f_1; s_1) = 51 \qquad c_{f_2} = \mathrm{com}_{ck}(f_2; s_2) = 106.$$

For the next step $\delta_0, \delta_1, \delta_2$ are needed and therefore the prover calculates for $i = i_2 i_1 i_0 \in \{0, 1, 2, 3, 4\}$ the five products $\prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j}$ using a binary tree .



The prover takes the a_i and multiplies them on the result of the binary tree, to get

$$\begin{aligned}
i = 0: &\quad a_0 \cdot X^3 = 81X^3 \\
i = 1: &\quad a_1 \cdot (84X^3 + 25X^2) = 79X^3 + 50X^2 \\
i = 2: &\quad a_2 \cdot (25X^3 + X^2) = 5X^3 + 18X^2 \\
i = 3: &\quad a_3 \cdot (53X^3 + 86X^2 + 25X) = 35X^3 + 40X^2 + 82X \\
i = 4: &\quad a_4 \cdot (2X^3 + 47X^2) = 2X^3 + 47X^2
\end{aligned}$$

Lastly, to extract the values δ_i the prover adds for $i = 0, 1, 2$ the parameters of X^i.

$$\delta_0 = 0 \qquad \delta_1 = 82 \qquad \delta_2 = 50 + 18 + 40 + 47 = 66 \mod 89.$$

Now the prover picks $t_0 = 77, t_1 = 26, t_2 = 81$ at random and commits to the δ_i's.

$$c_{\delta_0} = \mathrm{com}_{ck}(\delta_0, t_0) = 149 \qquad c_{\delta_1} = 106 \qquad c_{\delta_2} = 29.$$

Finally, the prover calculates $f_0 u = 53$, $f_1 u^2 = 25$, computes the commitment to these values by picking $\xi_0 = 36, \xi_1 = 88$ and gets

$$c_{fu_0} = 101 \qquad c_{fu_1} = 116.$$

The prover sends all commitments to the verifier, more precisely the values

$$c_{u_1} = 1 \quad c_{u_2} = 142 \quad c_{f_0} = 142 \quad c_{f_1} = 51, \quad c_{f_2} = 106$$

$$c_{\delta_0} = 149 \quad c_{\delta_1} = 129 \quad c_{\delta_2} = 29 \quad c_{fu_0} = 101 \quad c_{fu_1} = 116.$$

The verifier returns challenge $x = 21 \in \mathbb{Z}_{89}$ and the prover calculates answers

$$\overline{f}_0 = xu + f_0 = 9 \qquad \overline{f}_1 = 81 \qquad \overline{f}_2 = 0$$
$$\overline{r}_0 = xr_0 + s_0 = 11 \qquad \overline{r}_1 = 24 \qquad \overline{r}_2 = 46$$

$$\overline{t} = x^3 t + t_0 X^0 + t_1 X + t_2 X^2 = 69$$

$$\overline{\xi}_0 = xr_1 - \overline{f}_0 r_0 + \xi_0 = 59 \qquad \overline{\xi}_1 = 27,$$

and sends all values to the verifier.

The verifier checks first if all commitments are in $\mathbb{G}$ and all answers valid numbers in $\mathbb{Z}_{89}$ and tests for $i = 0, 1, 2$ if $c_{u_i}^x c_{f_i} = \mathrm{com}_{ck}(\overline{f}_i; \overline{r}_i)$.

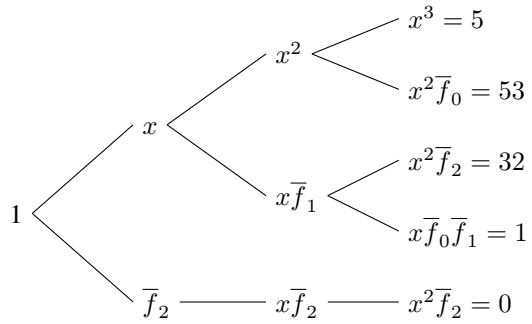
$$c_{u_0}^x c_{f_0} = 65 = \mathrm{com}_{ck}(\overline{f}_0; \overline{r}_0) \quad (\checkmark) \qquad c_{u_1}^x c_{f_1} = 51 = \mathrm{com}_{ck}(\overline{f}_1; \overline{r}_1) \quad (\checkmark)$$

$$c_{u_2}^x c_{f_2} = 46 = \mathrm{com}_{ck}(\overline{f}_2; \overline{r}_2) \quad (\checkmark)$$

Next the verifier checks $c_{ui+1}^x c_{u_i}^{-\overline{f}_i} c_{fu_i} = \mathrm{com}_{ck}(0, \overline{\xi}_i)$ for $i = 0, 1$

$$c_{u_1}^x c_{u_0}^{-\overline{f}_0} = 161 = \mathrm{com}_{ck}(0; \overline{\xi}_0) \quad (\checkmark) \qquad c_{u_2}^x c_{u_1}^{-\overline{f}_1} = 101 = \mathrm{com}_{ck}(0; \overline{\xi}_1) \quad (\checkmark)$$

Then the verifier calculates $\overline{\delta} = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0 \ldots i_d} \prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j}$ in a binary tree fashion.



The verifier multiplies the values by the a_i's and adds the results together, to get

$$\overline{\delta} = a_0 5 + a_1 53 + a_2 32 + a_3 1 + a_4 0 = 65 \in \mathbb{Z}_{89}.$$

and can check the last equation

$$c_v^{x^3} c_{\delta_2}^{x^2} c_{\delta_1}^{x^1} c_{\delta_0}^{x^0} = 22 = \mathrm{com}_{ck}\left(\overline{\delta}; \overline{t}\right) \quad (\checkmark).$$

6.2.1 Implementation and Practical Results

We implemented our polynomial evaluation argument to analyze the real life behavior of the protocols. Therefore, we used different modular subgroups with different security levels. Security levels were estimated following [Len04], and the approximate security levels can be found in Section 4.1.1. We have chosen two 160-bit subgroups modulo a 1 248-bit and a 1 536-bit prime. We have also chosen three different groups with $|q| = 256$ and $|p| = 1\,536, 2\,432$, and $3\,248$. The last two groups we are using are 384-bit subgroups modulo a 3 248-bit and a 7 936-bit prime.

Some of these groups are not standard and are not used in the research community, the reason that we picked these groups is that we wanted to explore the merit of different parameters, for example size of q versus size of p.

	Prover			**Verifier**		
D	Conservative	Optimized	Ratio	Conservative	Optimized	Ratio
10	19 ms	13 ms	0.66	18 ms	17 ms	0.95
100	35 ms	24 ms	0.67	31 ms	30 ms	0.98
1 000	57 ms	41 ms	0.71	45 ms	45 ms	1.00
5 000	125 ms	104 ms	0.83	65 ms	67 ms	1.01
10 000	202 ms	181 ms	0.90	78 ms	81 ms	1.00
50 000	764 ms	714 ms	0.97	140 ms	143 ms	1.00
100 000	1 505 ms	1 420 ms	0.98	216 ms	217 ms	1.00
500 000	7 407 ms	7 392 ms	1.00	695 ms	689 ms	0.99
1 000 000	15 541 ms	15 573 ms	1.00	1 319 ms	1 313 ms	1.00

Table 6.1: Run-time in ms of the polynomial evaluation argument on a group $\mathbb{G}$ with 256-bit order modulo a 1 536-bit prime for degree D between 10 and 1 000 000 for the conservative and the optimized version, and the ratio between the two versions.

We implemented a conservative version of our argument which contains the level of optimization we used to calculate the computation cost of the parties, that means we only optimized processes which contain a lot of multiplications. We optimized this version by including the sliding window technique in the commitments. To obtain our results we run the protocols for each set of parameters 100 times and calculated the mean. The experiments were carried out on a Mac Book Pro with 2.54 GHz CPU and 4 GB RAM.

Table 6.1 states the results of different degree polynomials evaluated in a 256-bit subgroup modulo a 1 536-bit prime for the conservative version and for the optimized versions, as well as the ration between both versions. The influence of the sliding window techniques for the prover is noticeable for small degree D; however, for large D the ratio between the two versions converge to 1. This indicates that the multiplications are accountable for the largest part of the run-time.

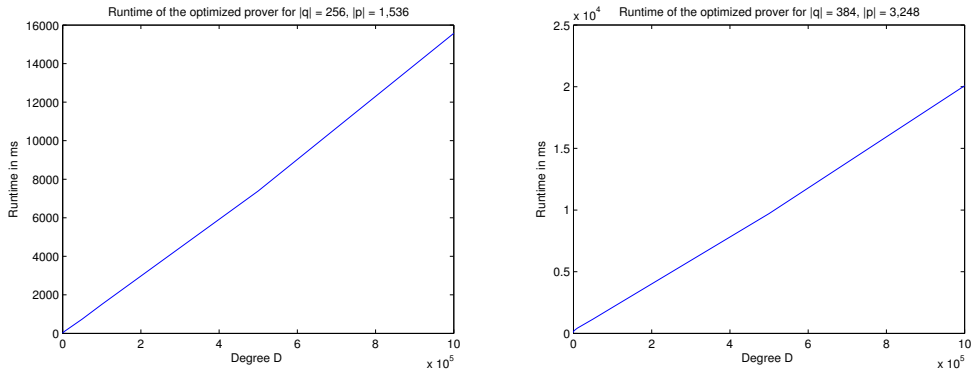


Figure 6.1: Run-time of our polynomial argument prover plotted against the degree D for $|q| = 256$ and $|p| = 1\,536$, and $|q| = 384$ and $|p| = 3\,248$

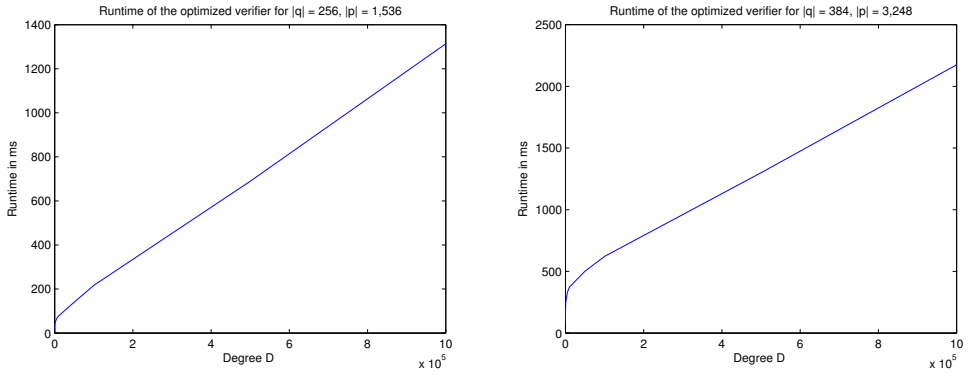


Figure 6.2: Run-time of our polynomial argument verifier plotted against the degree D for $|q| = 256$ and $|p| = 1\,536$, and $|q| = 384$ and $|p| = 3\,248$

Figure 6.1 shows the run time of the optimized version plotted against the degree D. We notice that the curve follows more or less a straight line. This result is surprising if we just look at the number of exponentiations; however, the calculation of the δ_i's seems to influence the runtime. To calculate the δ_i's $2D \log D$ multiplications are needed, this is a superlinear number of multiplications and we would

expect also a superlinear growth in the curve. However, the cost of a multiplication is less than the cost of an exponentiations and this can explain the linear growth for the runtime.

We see that the influence of the optimization is more or less non-existent for the verifier, which is a little bit surprising. On the one hand they have to calculate less exponentiations, on the other hand the number of multiplications is smaller, so one would expect a similar optimization level as for the prover. However, it seems that in this case the number of multiplications becomes dominant even for small degree D and therefore the run time of both versions is similar.

This assumption is supported by the graph of the optimized version, see Figure 6.2. Again the graph is a straight line, and this indicated that the run time is dominated by the number of multiplications.

	Prover			**Verifier**		
D	Conservative	Optimized	Ratio	Conservative	Optimized	Ratio
10	73 ms	45 ms	0.62	66 ms	61 ms	0.93
100	132 ms	82 ms	0.62	116 ms	111 ms	0.96
1,000	196 ms	127 ms	0.65	167 ms	163 ms	0.98
5,000	309 ms	217 ms	0.70	224 ms	219 ms	0.98
10,000	401 ms	305 ms	0.76	249 ms	244 ms	0.98
50,000	1 008 ms	900 ms	0.89	336 ms	334 ms	0.99
100 000	1 804 ms	1 695 ms	0.94	425 ms	426 ms	1.00
500 000	7 862 ms	7 745 ms	0.99	1 944 ms	935 ms	0.99
1 000 000	16 145 ms	16 149 ms	1.00	1 598 ms	1 585 ms	0.99

Table 6.2: Run-time in ms of the polynomial evaluation argument on a group $\mathbb{G}$ with 256-bit order modulo a 3 248-bit prime for degree D between 10 and 1 000 000 for the conservative and the optimized version, and the ratio between the two versions.

	Prover			**Verifier**		
D	Conservative	Optimized	Ratio	Conservative	Optimized	Ratio
10	454 ms	289 ms	0.64	415 ms	401 ms	0.97
100	843 ms	506 ms	0.60	742 ms	702 ms	0.95
1 000	1 204 ms	734 ms	0.61	1 044 ms	1 012 ms	0.97
5 000	1 633 ms	1 021 ms	0.63	1 363 ms	1 336 ms	0.98
10 000	1 854 ms	1 181 ms	0.64	1 489 ms	1 451 ms	0.97
50 000	2 772 ms	1 976 ms	0.71	1 767 ms	1 736 ms	0.98
100 000	3 826 ms	2 978 ms	0.78	1 967 ms	1 936 ms	0.98
500 000	11 360 ms	10 644 ms	0.94	2 827 ms	2 757 ms	0.98
1 000 000	21 493 ms	20 829 ms	0.97	3 782 ms	3 689 ms	0.98

Table 6.3: Run-time in ms of the polynomial evaluation argument on a group $\mathbb{G}$ with 384-bit order modulo a 7 936-bit prime for degree D between 10 and 1 000 000 for the conservative and the optimized version, and the ratio between the two versions.

Table 6.2 and 6.3 states the data for the conservative version and the optimized version for a 256-bit subgroup modulo a 3 248-bit prime and for a 384-bit subgroup modulo a 7 936-bit prime. In these cases the calculation of the δ_i's get also dominant, but the noticeable effect on the run time occurs for even bigger D. This phenomenon can be explained by the fact that the cost of a single exponentiation rise higher than the cost of a multiplication, which stays relatively cheap.

In table 6.4 we can find the results of the optimized version for a 160-bit subgroup modulo a 1 248-

$\|q\| = 160$	$\|p\| = 1\,248$		$\|p\| = 1\,536$	
D	Prover	Verifier	Prover	Verifier
10	7 ms	8 ms	9 ms	11 ms
100	12 ms	14 ms	16 ms	20 ms
1 000	23 ms	21 ms	29 ms	29 ms
5 000	77 ms	33 ms	85 ms	44 ms
10 000	143 ms	42 ms	154 ms	54 ms
50 000	635 ms	95 ms	656 ms	109 ms
100 000	1 330 ms	161 ms	1 362 ms	175 ms
500 000	6 773 ms	571 ms	6 787 ms	585 ms
1 000 000	14 231 ms	1 113 ms	14 289 ms	1 126 ms

Table 6.4: Run-time in ms of the polynomial evaluation argument for different degree D between 10 and 1 000 000 for the optimized version on a group with 160-bit order modulo a 1 248-bit prime and a 1 536-bit prime.

bit prime and a 1 536-bit prime. Whereas Table 6.5 and Table 6.6 show data of the optimized version for a 256-bit subgroup modulo different primes p and a 384-bit subgroup modulo different size primes.

$\|q\| = 256$	$\|p\| = 1\,536$		$\|p\| = 2\,432$		$\|p\| = 3\,248$	
D	Prover	Verifier	Prover	Verifier	Prover	Verifier
10	13 ms	17 ms	28 ms	39 ms	45 ms	61 ms
100	24 ms	30 ms	51 ms	70 ms	82 ms	111 ms
1 000	41 ms	45 ms	80 ms	103 ms	127 ms	163 ms
5 000	106 ms	67 ms	154 ms	141 ms	217 ms	219 ms
10 000	182 ms	81 ms	232 ms	161 ms	305 ms	244 ms
50 000	714 ms	143 ms	766 ms	233 ms	900 ms	334 ms
100 000	1 420 ms	217 ms	1 489 ms	313 ms	1 695 ms	426 ms
500 000	7 392 ms	689 ms	7 411 ms	795 ms	7 745 ms	935 ms
1 000 000	15 573 ms	1 313 ms	15 441 ms	1 411 ms	16 149 ms	1 585 ms

Table 6.5: Run-time in ms of the polynomial evaluation argument for different degree D between 10 and 1 000 000 for the optimized version on a group with 256-bit order modulo a 1 536-bit a 2 432-bit and a 3 248-bit prime.

$\|q\| = 384$	$\|p\| = 3\,248$		$\|p\| = 7\,936$	
D	Prover	Verifier	Prover	Verifier
10	67 ms	91 ms	289 ms	401 ms
100	123 ms	167 ms	506 ms	702 ms
1 000	191 ms	249 ms	734 ms	1 012 ms
5 000	308 ms	330 ms	1 021 ms	1 336 ms
10 000	425 ms	372 ms	1 181 ms	1 451 ms
50 000	1 155 ms	501 ms	1 976 ms	1 736 ms
100 000	2 116 ms	622 ms	2 978 ms	1 936 ms
500 000	9 718 ms	1 300 ms	10 644 ms	2 757 ms
1 000 000	20 083 ms	2 175 ms	20 829 ms	3 689 ms

Table 6.6: Run-time of the polynomial evaluation argument for different degree D between 10 and 1 000 000 for the optimized version on a group with 384-bit order modulo a 3 248-bit prime and a 7 936-bit prime.

As expected the run-time of both parties gets larger, if the subgroup size stays fixed and the moduli

value p increases. But, this increase is not linear with the increase of the moduli value. For instance for the 256-bit subgroups, see table 6.5, $|p| = 2\,432$ is 1.6 times of $|p| = 1\,536$; however the run-time for the prover is for big degree polynomials approximately the same for both choices of parameters. The moduli size of the last group $|p| = 3\,248$ is approximately double the size of the prime $|p| = 1\,536$. However, the run-time for the prover for big degree polynomials is approximately the same for both settings.

For small D the reverse is true, in this case the run-time is between 2-4 times slower for the bigger moduli p. Again this is due to the fact that the cost of single exponentiations increase for bigger moduli and this cost influences the run-time before the cost of the multiplications becomes dominant. This effect is the same for all fixed subgroups.

For the verifier the run-time of small degree D is for big modulo around 3-4 times slower than for small modulo values. For polynomials with bigger degree the factor gets smaller. However, the bigger the prime p is, the longer it takes for the factor to shrink. For instance for $|q| = 256$ there is a 20 % different between the run-time of $D = 1\,000\,000$ for $|p| = 1\,536$ and $|p| = 3\,248$; whereas for $|q| = 384$ the difference is still 70% for the same D, and $|p| = 3\,248$ and $|p| = 7\,936$.

The run-time of both parties seems to be dominated by the multiplications in $\mathbb{Z}_q$. If the size of the subgroup is fixed the cost of these multiplications is the same independently of the size of the modulo p. So, the results support the belief that the run-time is dominated by the multiplication.

In the reverse case the size of the modulo prime p stays fixed and the size of the subgroup changes. We tested this setting on a 160-bit subgroup and a 256-bit subgroup modulo a 1 536-bit prime, and a 160-bit subgroups, a 256-bit subgroup, and a 384-bit subgroup modulo a 3 248-bit prime. The result can be found in table 6.7 and 6.8.

$\|p\| = 1\,536$	$\|q\| = 160$		$\|q\| = 256$	
D	Prover	Verifier	Prover	Verifier
10	9 ms	11 ms	13 ms	17 ms
100	16 ms	20 ms	24 ms	30 ms
1 000	29 ms	29 ms	41 ms	45 ms
5 000	85 ms	44 ms	106 ms	67 ms
10 000	154 ms	54 ms	182 ms	81 ms
50 000	656 ms	109 ms	714 ms	143 ms
100 000	1 362 ms	175 ms	1 420 ms	217 ms
500 000	6 787 ms	585 ms	7 39 ms	689 ms
1 000 000	14 289 ms	1 126 ms	15 573 ms	1 313 ms

Table 6.7: Run-time of the polynomial evaluation argument for different degree D between 10 and 1 000 000 for the optimized version on a group modulo a 1 536-bit prime and order 160-bit or 256-bit.

$\|p\| = 3\,248$	$\|q\| = 160$		$\|q\| = 256$		$\|q\| = 384$	
D	Prover	Verifier	Prover	Verifier	Prover	Verifier
10	30 ms	39 ms	45 ms	61 ms	67 ms	91 ms
100	55 ms	73 ms	82 ms	112 ms	123 ms	167 ms
1 000	82 ms	105 ms	128 ms	161 ms	191 ms	249 ms
5 000	159 ms	150 ms	234 ms	219 ms	308 ms	330 ms
10 000	241 ms	172 ms	340 ms	246 ms	425 ms	372 ms
50 000	749 ms	237 ms	1 072 ms	340 ms	1 155 ms	501 ms
100 000	1 454 ms	308 ms	2 112 ms	439 ms	2 116 ms	622 ms
500 000	6 923 ms	740 ms	7 745 ms	935 ms	9 718 ms	1 300 ms
1 000 000	14 456 ms	1 288 ms	16 149 ms	1 585 ms	20 083 ms	2 175 ms

Table 6.8: Run-time of the polynomial evaluation argument for different degree D between 10 and 1 000 000 for the optimized version on groups modulo a 3 248-bit prime for different order sizes 160-bit, 256-bit and 384-bit.

As the execution time of the prover is dominated by the multiplications, we expect that the run time differs for the prover for different subgroup sizes. This expectation is confirmed by the data, we see that for increasing subgroup size the performance of the prover takes more time. The same can be observed for the verifier.

We can conclude that for medium to large degree polynomials the increase of the run-time is insignificant compared to the increase of the security level. For instance keeping the modulo value fixed and increasing the subgroup size from 160-bit to 256-bit, increase the security from around 80-bit to 128-bit, but the run-time for big D is only 10% larger. Therefore, it is possible to adjust the parameters to achieve higher security without compromising performance too much.

In general we can say that our new polynomial evaluation argument is practical. For small and medium size degrees D our argument is very fast, even for big security parameters. For small and medium term protection and big D the complete protocol runs in around 10 seconds and for long term protection in around 20 seconds. These values seem not practical but can be reduced further by using more sophisticated implementation techniques, for instance multi-threading. Furthermore, randomization of the verification would speed up the protocol further. Taking also in account that the computer we did our experiments on, were on the lower side of speed and memory, we can expect that the results above will speed up in future on newer machines.

D	Round 1	Round 2	Round 3	Statement
10	7 KB	0.08 KB	1 KB	1 KB
100	13 KB	0.08 KB	2 KB	8 KB
1 000	18 KB	0.08 KB	2 KB	72 KB
5 000	24 KB	0.08 KB	3 KB	381 KB
10 000	25 KB	0.08 KB	3 KB	761 KB
50 000	29 KB	0.08 KB	4 KB	3.9 MB
100 000	31 KB	0.08 KB	4 KB	7.8 MB
500 000	35 KB	0.08 KB	4 KB	38.9 MB
1 000 000	37 KB	0.08 KB	5 KB	77.8 MB

Table 6.9: Size of each round and of the statement for different degree D for a 256-bit subgroup modulo a 1 526-bit prime.

The actual size of each round are given in Table 6.9 for $|q| = 256$ and $|p| = 1\,536$. The size of the challenge in round 2 consists only of one single element in $\mathbb{Z}_q$ and as expected this element can be neglected, it does not increase the size of the whole argument. In round 1 $4d$ elements are sent against $3d$ elements in round 3, but the elements in round 3 are field elements from $\mathbb{Z}_q$ and therefore smaller than the commitments in round 1. So, we expect that the size of the round 3 is smaller than round 1. More precisely if the size of a field element is f-bit and a group element is nf-bits, than the size of round 1 is around $4/3n$ bigger than round 1. This estimate is confirmed in our experiment, for instance in the case of $|q| = 256$ and $|p| = 1\,536$, we expect round 1 to be roughly 8 times bigger than the data from round 3 and in reality this is the case. A consequence of this is that the data of round 3 compared to round 1 has more or less no influence of the size of the whole argument.

We also see that the size of the statement is much bigger that the argument size, besides for small degrees $D < 1000$. Thus, we can conclude that our polynomial argument is sublinear in the statement size, but not in the witness size, which consists only of 4 field elements.

$\|q\|$	160	160	160	256	256	256	384	384
$\|p\|$	1 248	1 536	3 248	1 536	2 432	3 248	3 248	7 984
10	6 KB	8 KB	15 KB	8 KB	12 KB	16 KB	16 KB	37 KB
100	11 KB	14 KB	28 KB	15 KB	22 KB	28 KB	29 KB	65 KB
1 000	16 KB	20 KB	40 KB	21 KB	31 KB	41 KB	42 KB	97 KB
5 000	21 KB	26 KB	52 KB	27 KB	41 KB	53 KB	55 KB	127 KB
10 000	23 KB	28 KB	56 KB	29 KB	44 KB	57 KB	59 KB	136 KB
50 000	26 KB	32 KB	64 KB	33 KB	50 KB	65 KB	67 KB	156 KB
100 000	28 KB	34 KB	68 KB	35 KB	53 KB	70 KB	71 KB	166 KB
500 000	31 KB	38 KB	76 KB	39 KB	60 KB	78 KB	80 KB	186 KB
1 000 000	33 KB	40 KB	80 KB	42 KB	63 KB	82 KB	84 KB	196 KB

Table 6.10: Size whole argument for different degree D for all choices of groups $\mathbb{G}$.

Table 6.10 states the size of a single argument for all choices of groups and degree D, the size of the argument is very small, only consisting of a few kilo bytes. Even for high security levels the size of the argument requires very few disk space. Thus, the protocol is very small and can be used also in applications which have low speed connections.

6.3 Multivariate Polynomial Argument

We will demonstrate now how one can adapt the polynomial argument from the former section to multivariate polynomials

$$P(X_1, \ldots, X_N) = \sum_{i_1,\ldots,i_N=0}^{D} a_{i_1 \ldots i_N} X_1^{i_1} \ldots X_N^{i_N}.$$

Given a multivariate polynomial $P(X_1, \ldots, X_N)$ and committed values $u_1, \ldots, u_N$ in $c_{u_{10}}, \ldots, c_{u_{N0}}$ the prover wants to show that for committed value v it holds $v = P(u_1, \ldots, u_N)$.

Similar to Section 5.3 we will write i_j in binary, i.e. $i_j = i_{j0} \ldots i_{jd}$, where $i_{jk} \in \{0, 1\}$ and without loss of generality $D = 2^{d+1} - 1$. We can write all terms $X_j^{i_j} = \prod_{k=0}^{d} \left(X_j^{2^k}\right)^{i_{jk}}$, and plugging these

terms in the polynomial gives us

$$\begin{aligned} P(X_1,\ldots,X_N) &= \sum_{i_1,\ldots,i_N=0}^{D} a_{i_1\ldots i_n} \prod_{k=0}^{d} \left(X_1^{2^k}\right)^{i_{ik}} \cdot \ldots \cdot \prod_{k=0}^{d} \left(X_N^{2^k}\right)^{i_{Nk}} \\ &= \sum_{i_{10},\ldots,i_{Nd}=0}^{1} a_{i_{10}\ldots i_{Nd}} \prod_{j=1}^{N}\prod_{k=0}^{d} \left(X_j^{2^k}\right)^{i_{jk}}. \end{aligned}$$

The prover now picks $f_{jk} \leftarrow \mathbb{Z}_q$ for $j = 1,\ldots,N$ and $k = 0,\ldots,d$, and defines a new polynomial as

$$\begin{aligned} Q(X_1,\ldots,X_n) &= \sum_{i_{i0},\ldots,i_{nd}=0}^{1} a_{i_{10}\ldots i_{Nd}} \prod_{j=1}^{N}\prod_{k=0}^{d} (X_j u_j^{2^k} + f_{jk})^{i_{jk}} X_j^{1-i_{jk}} \\ &= X_1^{d+1}\ldots X_N^{d+1} v + X_1^{d} X_2^{d+1}\ldots X_N^{d+1}\delta_\nu + \ldots + X_N\delta_1 + \delta_1 \\ &= X_1^{d+1}\ldots X_N^{d+1} v + \sum_{\substack{k_1,\ldots,k_N=0 \\ \setminus\{k_1=\ldots=k_N=d+1\} \\ \wedge l=\sum_{i=1}^{d+1} k_i (d+1)^i}}^{d+1} X_1^{k_1}\ldots X_N^{k_N}\delta_l \end{aligned}$$

where $\nu = (d+1)^N - 1$. For each $i_{j,k}$ either a factor $X_j u_j^{2^k}$ or a factor X_j is included in the polynomial $Q(X_1,\ldots,X_n)$. More precisely, the inner product for a coefficient $a_{10\ldots i_{ND}}$ equals for fixed j

$$\prod_{k=0}^{d} (X_j u_j^{2^k} + f_{jk})^{i_{jk}} X_j^{1-i_{jk}} = X_j^{d+1} \prod_{k=0}^{d} \left(u_j^{2^k}\right)^{i_{jk}} + \alpha(X_j),$$

this gives us

$$a_{i_{10}\ldots i_{Nd}} \prod_{j=1}^{N}\prod_{k=0}^{d} (X_j u_j^{2^k} + f_{jk})^{i_{jk}} X_j^{1-i_{jk}} = a_{i_{10}\ldots i_{Nd}} X_1^{d+1}\ldots X_N^{d+1} \prod_{j=1}^{N}\prod_{k=0}^{d} (u_j^{2^k})^{i_{jk}} + \beta(X_1,\ldots,X_N).$$

If we add all terms together we get that the coefficient of $X_1^{d+1}\ldots X_n^{d+1}$ is $P(u_1,\ldots,u_n) = v$.

The prover commits themselves for all $j = 1,\ldots,N$ to $u_j^2,\ldots,u_j^{2^d}$ in $c_{u_{j1}},\ldots,c_{u_{jd}}$. Furthermore, to the values f_{jk} in $c_{f_{jk}}$ for all k and also to values $\delta_0,\ldots,\delta_\nu$ in $c_{\delta_0},\ldots,c_\nu$. After seeing the random challenges $x_1,\ldots,x_N \in \mathbb{Z}_q^*$ the prover opens for all j and k the products $c_{u_{jk}}^{x} c_{j_{jk}}$ to $\overline{f}_{jk} = x u_j^{2^k} + f_{jk}$.

The verifier computes

$$\overline{\delta} = \sum_{i_{10},\ldots,i_{Nd}=0}^{1} a_{i_{10}\ldots i_{Nd}} \prod_{j=1}^{N}\prod_{k=0}^{d} \overline{f}_{jk}^{\,i_{jk}} x_j^{1-i_{jk}}$$

and opens

$$c_v^{x_1^{d+1}\ldots x_N^{d+1}} \prod_{\substack{k_1,\ldots,k_N=0 \\ \setminus\{k_1=\ldots=k_N=d+1\} \\ \wedge l=\sum_{i=1}^{d+1} k_i (d+1)^i}}^{d+1} c_{\delta_l}^{x_1^{k_1}\ldots x_N^{k_N}}$$

to $\overline{\delta}$. Unless $v = P(u_1, \ldots, u_N)$ this check has a negligible probability of being true as

$$\begin{aligned}\overline{\delta} &= \sum_{i_{10},\ldots,i_{Nd}=0}^{1} a_{i_{10}\ldots i_{Nd}} \prod_{j=1}^{N}\prod_{k=0}^{d} \overline{f}_{jk}^{\,i_{jk}} x_j^{1-i_{jk}} \\ &= x_1^{d+1}\ldots x_n^{d+1} v + x_1^d x_2^{d+1}\ldots x_N^{d+1}\delta_\nu + \ldots + x_N\delta_1 + \delta_0.\end{aligned}$$

In parallel the prover will give standard 3-move zero-knowledge arguments for all $c_{u_{jk}}$ that these commitments indeed contain the successive powers of $u_1, \ldots, u_N$, for $j = 1, \ldots, N$ and $k = 0, \ldots, k$.

Statement: $\{\mathbb{G}, p, q\},\ ck,\ c_{u_{10}}, \ldots, c_{u_{N0}}, c_v \in \mathbb{G}$ and

$$P(X_1, \ldots, X_N) = \sum_{i_1,\ldots,i_N=0}^{D} a_{i_1\ldots i_N} X_1^{i_1}\ldots X_N^{i_N} \in \mathbb{Z}_p[X_1, \ldots, X_N]$$

Prover's witness: $u_1, \ldots, u_N, v, r_{10}, \ldots, r_{N0}, t \in \mathbb{Z}_q$ such that $v = P(u_1, \ldots, u_N)$ and

$$c_v = \mathrm{com}_{ck}(v; t) \qquad c_{u_{j0}} = \mathrm{com}_{ck}(u_j; r_{j0}), \quad j = 1, \ldots, N$$

Initial message: Compute

1. $c_{u_{j1}} = \mathrm{com}_{ck}(u_j^2; r_{j1}), \ldots, c_{u_{jd}} = \mathrm{com}_{ck}(u_j^{2^d}; r_{jd})$ for all j where $r_{j1}, \ldots, r_{jd} \leftarrow \mathbb{Z}_q$
2. $\boldsymbol{c}_{f_j} = \mathrm{com}_{ck}(\boldsymbol{f}_j; \boldsymbol{s}_j)$ where $\boldsymbol{f}_j, \boldsymbol{s}_j \leftarrow \mathbb{Z}_q^{d+1}$ for $j = 1, \ldots, N$
3. $\nu = (d+1)^N - 1$ and $\delta_0, \ldots, \delta_\nu \in \mathbb{Z}_q$ such that

$$\begin{aligned}\sum_{i_{10},\ldots,i_{Nd}=0}^{1} a_{i_{10}\ldots i_{Nd}} \prod_{j=1}^{N}\prod_{k=0}^{d} (X_j u_j^{2^k} + f_{jk})^{i_{jk}} X^{1-i_{jk}} \\ = X_1^{d+1}\ldots X_N^{d+1} v + X_1^d X_2^{d+1}\ldots X_N^{d+1}\delta_\nu + \ldots + X_N\delta_1 + \delta_0\end{aligned}$$

4. $\boldsymbol{c}_\delta = \mathrm{com}_{ck}(\boldsymbol{\delta}; \boldsymbol{t})$ where $\boldsymbol{t} \leftarrow \mathbb{Z}_q^\nu$
5. $\boldsymbol{c}_{f_j u_j} = \mathrm{com}_{ck}(\boldsymbol{f}_j \circ \boldsymbol{u}_j; \boldsymbol{\xi}_j)$ for all j where $\boldsymbol{u}_j = (u_j, u_j^2, \ldots, u_j^{2^d})$,

$$\boldsymbol{f}_j = (f_{j0}, \ldots, f_{jd-1}), \boldsymbol{\xi}_{jk-1} \leftarrow \mathbb{Z}_q^d$$

Send: $c_{u_{11}}, \ldots, c_{u_{Nd}}, \boldsymbol{c}_{f_1}, \ldots, \boldsymbol{c}_{f_N}, \boldsymbol{c}_\delta, \boldsymbol{c}_{f_1 u_1}, \ldots, \boldsymbol{c}_{f_N u_N}$

Challenge: $x_1, \ldots, x_N \leftarrow \mathbb{Z}_q^*$

Answer: Compute

1. $\overline{f}_{jk} = x_j u_j^{2^k} + f_{jk}$ and $\overline{r}_{jk} = x_j r_{jk} + s_{jk}$ for all j and k
2. $\overline{\xi}_{jk-1} = x_j r_{jk+1} - \overline{f}_{jk} r_{jk} + \xi_{jk-1}$ for all j, k
3. $$\overline{t} = x_1^{d+1}\ldots x_N^{d+1} t + \sum_{\substack{k_1,\ldots,k_N=0 \\ \setminus\{k_1=\ldots=k_N=d+1\} \\ \wedge l=\sum_{i=1}^{d+1} k_i (d+1)^i}}^{d+1} t_l x_1^{k_1}\ldots x_N^{k_N}$$

Send: $\overline{f}_{10}, \dots \overline{f}_{Nd}, \overline{r}_{10}, \dots, \overline{r}_{Nd}, \overline{\xi}_{10}, \dots, \overline{\xi}_{Nd-1}, \overline{t}$

Verification: Accept if and only if for all j, k

1. $c_{u_{11}}, \dots, c_{u_{Nd}} \in \mathbb{G},\ \boldsymbol{c}_{f_1}, \dots, \boldsymbol{c}_{f_N} \in \mathbb{G}^{d+1},\ \boldsymbol{c}_\delta \in \mathbb{G}^{(d+2)^N-1},\ \boldsymbol{c}_{f_1 u_1}, \dots, \boldsymbol{c}_{f_N u_N} \in \mathbb{G}^d$
2. $\overline{f}_{10}, \dots \overline{f}_{Nd}, \overline{r}_{10}, \dots, \overline{r}_{Nd}, \overline{\xi}_{10}, \dots, \overline{\xi}_{Nd-1}, \overline{t} \in \mathbb{Z}_q$
3. $c_{u_j^{2^k}}^{x_j} c_{f_{jk}} = \mathrm{com}_{ck}(\overline{f}_{jk}; \overline{r}_{jk}) \qquad c_{u_j^{2^{k+1}}}^{x_j} c_{u_j^{2^k}}^{-\overline{f}_{jk}} c_{f_{jk-1}u_j^{2^k}} = \mathrm{com}_{ck}(0; \overline{\xi}_j k)$
4. $$c_v^{x_1^{d+1} \dots x_N^{d+1}} \prod_{\substack{k_1,\dots,k_N=0 \\ \setminus\{k_1=\dots=k_N=d+1\} \\ \wedge l=\sum_{i=1}^{d+1} k_i (d+1)^i}}^{d+1} c_{\delta_l}^{x_1^{k_1} \dots x_N^{k_N}} = \mathrm{com}_{ck}\left(\overline{\delta}; \overline{t}\right)$$

   where
   $$\overline{\delta} = \sum_{i_{10},\dots,i_{Nd}=0}^{1} a_{i_{10}\dots i_{Nd}} \prod_{j=1}^{N} \prod_{k=0}^{d} \overline{f}_{jk}^{\,i_{jk}} x_j^{1-i_{jk}}$$

Theorem 26. *The protocol above is a public coin generalized $\Sigma-$protocol of openings $u_1, \dots, u_N$ and v such that $v = P(u_1, \dots, u_N)$.*

Proof. Perfect completeness follows by careful inspection.

We will now argue that we have perfect SHVZK. On challenge $x_1, \dots, x_N \in \mathbb{Z}_q^*$ the simulator picks $c_{u_{j1}}, \dots, c_{u_{jd}}, c_{\delta_1}, \dots, c_{\delta_\nu} \leftarrow \mathbb{G}$ and $\overline{f}_j \leftarrow Z_q^{d+1}, \overline{r}_j, \overline{\xi}_j \leftarrow \mathbb{Z}_q^d, \overline{t} \leftarrow \mathbb{Z}_q$ and computes for all j and k

$$c_{f_{jk}} = \mathrm{com}_{ck}(\overline{f}_{jk}; \overline{r}_{jk}) c_{u_j^{2^k}}^{-x_k} \qquad c_{f_{jk}u_j^{2^k}} = \mathrm{com}_{ck}(0; \overline{\xi}_{jk}) c_{u_j^{2^{k+1}}}^{-x_j} c_{u_j^{2^k}}^{\overline{f}_{jk}}$$

and

$$\overline{\delta} = \sum_{i_{10},\dots,i_{Nd}=0}^{1} a_{i_{10}\dots i_{Nd}} \prod_{j=1}^{N} \prod_{k=0}^{d} \overline{f}_{jk}^{i_{jk}} x_j^{1-i_{jk}},$$

$$c_{\delta_0} = \mathrm{com}_{ck}\left(\overline{v}; \overline{t}\right) c_v^{-x_1^{d+1} \dots x_N^{d+1}} \prod_{\substack{k_1,\dots,k_N=0 \\ \setminus\{k_1=\dots=k_N=d+1\} \\ \wedge l=\sum_{i=1}^{d+1} k_i (d+1)^i}}^{d+1} c_l^{-x_1^{k_1} \dots x_N^{k_N}}.$$

This is a perfect simulation. In a real argument $c_{u_{j1}}, \dots, c_{u_{jd}}, c_{\delta_1}, \dots, c_{\delta_\nu}$ are uniformly random perfectly hiding commitments and therefore uniformly random, which is the same case in the simulation for all $j = 1, \dots, N$. In a real argument $\overline{\boldsymbol{f}}_j \in \mathbb{Z}_q^{d+1}, \overline{\boldsymbol{r}}_j, \boldsymbol{\xi}_j \in \mathbb{Z}_q^d, \overline{t} \in \mathbb{Z}_q$ are uniformly random since $f_{j0}, \dots, f_{jd}, r_{j0}, \dots, r_{jd}, \boldsymbol{t}, \xi_{j0}, \dots, \xi_{jd-1}$ are picked at random. In the simulation $\overline{\boldsymbol{f}}_j \in \mathbb{Z}_q^{d+1}, \overline{\boldsymbol{r}}_j, \boldsymbol{\xi}_j \in \mathbb{Z}_q^d, \overline{t} \in \mathbb{Z}_q$ are picked at random; thus, they follow the same distribution. Finally, both in the simulation and in the real argument these variables uniquely determine the values of $\boldsymbol{c}_{f_j}, j = 1, \dots, n$ and $c_{f_0 u_1}, \dots, c_{f_{d-1} u_{N(d-1)}}$ through the verification equations. So, simulated and real argument have identical probability distributions and the argument is SHVZK.

Next, we have to show that we have generalized special soundness. Given $(d+1)^N$ accepting transcripts with different challenges x_i and given $\overline{f}_{jk}^{(1)}, \overline{r}_{jk}^{(1)}$ and $\overline{f}_{jk}^{(2)}, \overline{r}_{jk}^{(2)}$ in the first two answers to challenges $x_{11}, \dots x_{1N}$ and $x_{(21}, \dots, x_{2N}$ the extractor can take linear combinations of the verification

equations to get openings of the commitments $c_{u_j^{2^k}}$. More precisely, we have that the two answers satisfy

$$c_{u_{jk}}^{x_{1j}} c_{f_{jk}} = \mathrm{com}_{ck}(\overline{f}_{jk}^{(1)}; \overline{r}_{jk}^{(1)}) \qquad\qquad c_{u_{jk}}^{x_{2j}} c_{f_{jk}} = \mathrm{com}_{ck}(\overline{f}_{jk}^{(2)}; \overline{r}_{jk}^{(2)}).$$

Picking α_1, α_2 such that $\alpha_1 x_{1j} + \alpha_2 x_{2j} = 1$ and $\alpha_1 + \alpha_2 = 0$ gives us

$$c_{u_jk} = c_{u_{jk}}^{\alpha_1 x_{1)j} + \alpha_2 x_{2j}} c_{f_{jk}}^{\alpha_1 + \alpha_2} = \mathrm{com}_{ck}(\alpha_1 \overline{f}_{jk}^{(1)} + \alpha_2 \overline{f}_{jk}^{(2)}; \alpha_1 \overline{r}_{jk}^{(1)} + \alpha_2 \overline{r}_{jk}^{(2)}),$$

which is an opening of $c_{u_{jk}}$ for all j and k.

Other types of linear combinations of the verification equations give us openings of the other commitments $c_{f_{jk}}, c_{f_{jk-k}u_{jk}}$ and c_v the prover sends in the initial message. In the case of c_l we find the linear combination as follows. Let

$$M = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{11}^d x_{12}^{d+1} \dots x_{1N}^{d+1} \\ \vdots & & & \vdots & \\ 1 & x_{(\nu+1)1} & x_{(\nu+1)2} & \dots & x_{(\nu+1)1}^d x_{(\nu+1)2}^{d+1} \dots x_{(\nu+1)N}^{d+1} \end{pmatrix}.$$

With overwhelming probability every set of N vectors are linearly independent and therefore all rows in the matrix are independent and the matrix M is invertible. By taking linear combinations of the verification equations

$$c_v^{x_1^{d+1} \dots x_N^{d+1}} \prod_{\substack{k_1,\dots,k_N=0 \\ \setminus\{k_1=\dots=k_N=d+1\} \\ \wedge l=\sum_{i=1}^{d+1} k_i (d+1)^i}}^{d+1} c_{\delta_l}^{x_1^{k_1} \dots x_N^{k_N}} = \mathrm{com}_{ck}\left(\sum_{i_{i0},\dots,i_{Nd}=0}^{1} a_{i_{10}\dots i_{Nd}} \prod_{j=1}^{N} \prod_{k=0}^{d} \overline{f}_{ij}^{i_{jk}} x_j^{1-i_{jk}}; \overline{t} \right)$$

we get that

$$\begin{pmatrix} \delta_0 & t_0 \\ \vdots & \vdots \\ \delta_\nu & t_\nu \\ v & s \end{pmatrix} = M^{-1} \begin{pmatrix} \sum_{i_{i0},\dots,i_{Nd}=0}^{1} a_{i_{10}\dots i_{Nd}} \prod_{j=1}^{N} \prod_{k=0}^{d} \overline{f}_{ij}^{i_{jk}} x_{(1)j}^{1-i_{jk}} & \overline{t}^{(1)} \\ \vdots & \vdots \\ \sum_{i_{i0},\dots,i_{Nd}=0}^{1} a_{i_{10}\dots i_{Nd}} \prod_{j=1}^{N} \prod_{k=0}^{d} \overline{f}_{ij}^{i_{jk}} x_{(\nu+1)j}^{1-i_{jk}} & \overline{t}^{((\nu+1)} \end{pmatrix}$$

which gives us openings of $c_{\delta_0}, \dots, c_{\delta_\nu}$ and c_v.

We now have openings to all the commitments. Because they are binding, each of them must be computed as they are by an honest prover in the argument. The verification equations

$$c_{u_{j(k+1)}}^{x_j} c_{u_{jk}}^{-\overline{f}_{jk}} c_{f_{jk} u_j^{2^k}} = \mathrm{com}_{ck}(0; \overline{\xi}_{jk})$$

now give us that the committed values satisfy

$$x_j u_{j(k+1)} - (x_j u_{j(k)} + f_{jk}) u_{j(j)} + \phi_{(jk)} = 0$$

for $j = 1, \ldots N,\ k = 0, \ldots, d-1$ with $u_{j(k)}$ being the value inside $c_{u_{jk}}$ and $\phi_{(jk)}$ being the value inside $c_{f_{jk}u_j^{2^k}}$. Since each of the polynomial equalities is of degree 1 and holds for $(d+2)^N$ different challenges we see that $u_{j(1)} = u_j^2, u_{j(2)} = u_j^{2^2}, \ldots, u_{j(d)} = u_j^{2^d}$ for all j.

Turning to the verification equation

$$c_v^{x_1^{d+1}\ldots x_N^{d+1}} \prod_{\substack{k_1,\ldots,k_N=0 \\ \setminus\{k_1=\ldots=k_N=d+1\} \\ \wedge l=\sum_{i=1}^{d+1} k_i(d+1)^i}}^{d+1} c_{\delta_l}^{x_1^{k_1}\ldots x_N^{k_N}} = \mathrm{com}_{ck}\left(\sum_{i_{10},\ldots,i_{Nd}=0}^{1} a_{i_{10}\ldots i_{Nd}} \prod_{j=1}^{N}\prod_{k=0}^{d} \overline{f}_{ij}^{i_{jk}} x_j^{1-i_{jk}}; \overline{t}\right)$$

we now have that this corresponds to the multivariate polynomial equation

$$\sum_{i_{i0},\ldots,i_{Nd}=0}^{1} a_{i_{10}\ldots i_{Nd}} \prod_{j=1}^{N}\prod_{k=0}^{d} (X_j u_j^{2^k} + f_{jk})^{i_{jk}} X_j^{1-i_{jk}}$$
$$= X_1^{d+1}\ldots X_N^{d+1} v + X_1^d X_2^{d+1}\ldots X_N^{d+1}\delta_\nu + \ldots + X_n\delta_1 + \delta_0.$$

With $(d+2)^n$ different values $x_{11}, \ldots, x_{(\nu+1)n}$ satisfying the equation, we conclude the two polynomials are identical. Looking at the coefficient for $X_1^{d+1}\ldots X_N^{d+1}$ we also conclude that the extracted openings of $c_{u_1}, \ldots, c_{u_N}$ and c_v satisfy $v = P(u_1, \ldots, u_N)$. Furthermore, the extracted openings are the same openings known by the prover. Otherwise the prover could use the extractor to break the commitment scheme. □

Efficiency: The communication consists of $(d+1)^N + 3Nd + N$ group elements and $3Nd + 2N$ field elements; therefore, the communication cost is $O(d^N) = O((\log D)^N)$.

The prover needs $2(d+1)^N + 6nN$ exponentiations in $\mathbb{G}$ to calculate the commitments and also has to calculate the δ_i's. The prover can calculate the degree d polynomials

$$\prod_{k=0}^{d} (X_j u_j^{2^k} + f_{jk})^{i_{jk}} X_j^{1-i_{jk}}$$

in a binary-tree fashion for all choices of $i_{jk} \in \{0,1\}$, which costs $O(NdD)$ multiplications in $\mathbb{Z}_q$. They can then calculate

$$\prod_{j=1}^{N}\prod_{k=0}^{d} (X_j u_j^{2^k} + f_{jk})^{i_{jk}} X_j^{1-i_{jk}}.$$

This costs another $O(d^N)$ multiplications for each polynomial; thus, in total $O(D^N d^N)$ multiplications. Lastly, the prover has to handle the $a_{i_{10}\ldots i_{Nd}}$, to do this efficient they multiple the values $a_{i_{jk}}$ on the inner polynomials in X_1, i.e.

$$\prod_{k=0}^{d} (X_1 u_1^{2^k} + f_{1k})^{i_{1k}} X_1^{1-i_{1k}},$$

this cost another $(D+1)^N(d+2)$ multiplications in $\mathbb{Z}_q$. Therefore, the total cost of the prover is $O(N \log D^N)$ exponentiations in $\mathbb{G}$ and $O(D^N d^N)$ multiplications in $\mathbb{Z}_q$.

The verifier has to compute $(d+1)^N + 3Nd$ exponentiations in $\mathbb{G}$. To calculate $\overline{\delta}$ the verifier

need to carry out $(D+1)^N Nd$ multiplications in $\mathbb{Z}_q$. Beside of this the verifier has to calculate $N(d+1)^N + 5Nd$ multiplications. So the cost for the verifier is $O(d^N)$ exponentiations in $\mathbb{G}$ and $O(NdD^N)$ multiplications in $\mathbb{Z}_q$.

In the calculation above we have ignored small constants and just focused on the dominant terms. Similar to the polynomial argument multi-exponentiation techniques such as the sliding window technique, Section 4.3, optimize the argument further. To reduce the computational burden even more randomized verification and other tricks [BGR98, Gro10] can be applied; hence, the estimates we have given above are quite conservative.

6.4 Batch Polynomial Argument

This section is joint work with Jens Groth.

Consider the case where we have a batch of polynomials $P^{(1)}(X), \ldots, P^{(L)}(X)$ of the form

$$P^{(\ell)}(X) = \sum_{i=0}^{D} a_i^{(\ell)} X^i = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d}^{(\ell)} \prod_{j=0}^{d} (X^{2^j})^{i_j},$$

commitments to evaluations $v_1 = P^{(1)}(u_1), \ldots, v_L = P^{(L)}(u_L)$ in committed values $u_1, \ldots, u_L$ and we want to show that indeed $v_i = P^{(i)}(u_i)$ holds for all i. A parallel repetition of the polynomial argument from Section 5.3 would give a communication complexity of $O(L \log D)$ group and field elements. We will now show that the Hadamard product argument from Section 6.2 can be used to reduce the cost to $O(\sqrt{L} \log D)$ when we have $L = mn$ and $m \approx n \approx \sqrt{L}$.

Like in the Hadamard product argument, we define $l(X) = \prod_{i=1}^{m}(X - \omega_i)$ and let $l_1(X), \ldots, l_m(X)$ be the Lagrange interpolation polynomials for $\Omega = \{\omega_1, \ldots, \omega_m\}$, which satisfy

$$l_i(X) = \begin{cases} 1 & \mod X - \omega_i \\ 0 & \mod \frac{l(X)}{X-\omega_i} \end{cases} \quad \text{for } i = 1, \ldots, m\,.$$

The idea is to batch-verify many polynomials simultaneously to reduce the communication cost. We will arrange the polynomials and the committed values in an $m \times n$ matrix, where $L = mn$. This gives us polynomials $P^{(i,k)}$ and the committed values $u_{i,k}$ and $v_{i,k}$ for $i = 1, \ldots, m$ and $k = 1, \ldots, n$. We will for each k verify the m polynomial evaluations $P^{(i,k)}(u_{i,k}) = v_{i,k}$ simultaneously.

In the argument we pick similar to the Hadamard argument random $f_j \leftarrow \mathbb{Z}_q^n$ and construct

$$\overline{\boldsymbol{u}}_{2^j} = l(X)\boldsymbol{f}_j + \sum_{i=1}^{m} l_i(X)\boldsymbol{u}_i^{2^j}.$$

Looking at the k'th entries $\overline{\boldsymbol{u}}_{2^j,k}$ of these vectors we have for each $i = 1, \ldots, m$ that $\overline{\boldsymbol{u}}_{2^j,k} \equiv u_{i,k}^{2^j} \bmod X - \omega_i$. Inserting this in the polynomial $P^{(i,k)}$ gives us

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d}^{(i,k)} \prod_{j=0}^{d} (\overline{\boldsymbol{u}}_{2^j,k})^{i_j} \equiv P^{(i,k)}(u_{i,k}) \equiv v_{i,k} \bmod X - \omega_i.$$

To verify many polynomials at once, the prover will demonstrate to the verifier that

$$\sum_{i=1}^{m} l_i(X) \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d}^{(i,k)} \prod_{j=0}^{d} (\overline{\boldsymbol{u}}_{2^j,k})^{i_j} \equiv \sum_{i=1}^{m} l_i(X) v_{i,k} \bmod l(X).$$

Statement: $\{\mathbb{G}, p, q\}$, ck, $c_{u_{1,2^0}}, c_{v_1}, \ldots, c_{u_{m,2^0}}, c_{v_m} \in \mathbb{G}$ and $L = mn$ polynomials $P^{(1,1)}, \ldots, P^{(m,n)}$ of the form

$$P^{(i,k)}(X) = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d}^{(i,k)} \prod_{j=0}^{d} (X^{2^j})^{i_j}.$$

Prover's witness: $\boldsymbol{u}_1, \boldsymbol{v}_1, \ldots, \boldsymbol{u}_m, \boldsymbol{v}_m \in \mathbb{Z}_q^n$ and $r_{1,2^0}, s_1, \ldots, r_{m,2^0}, s_m \in \mathbb{Z}_q$ such that for $i = 1, \ldots, m$ and $k = 1, \ldots, n$

$$c_{u_{i,2^0}} = \mathrm{com}_{ck}(\boldsymbol{u}_i; r_{i,2^0}) \qquad c_{v_i} = \mathrm{com}_{ck}(\boldsymbol{v}_i; s_i)$$

$$v_{i,k} = P^{(i,k)}(u_{i,k}) \quad \text{where} \quad \boldsymbol{u}_i = (u_{i,1}, \ldots, u_{i,n}) \qquad \boldsymbol{v}_i = (v_{i,1}, \ldots, v_{i,n})$$

Initial message: Compute

1. $c_{f_0} = \mathrm{com}_{ck}(\boldsymbol{f}_0; r_{f_0}), \ldots, c_{f_d} = \mathrm{com}_{ck}(\boldsymbol{f}_d; r_{f_d})$ where $\boldsymbol{f}_j \leftarrow \mathbb{Z}_q^n$ and $r_{f_j} \leftarrow \mathbb{Z}_q$
2. $c_{u_{1,2^1}} = \mathrm{com}_{ck}(\boldsymbol{u}_1^{2^1}; r_{1,2^1}), \ldots, c_{u_{m,2^d}} = \mathrm{com}_{ck}(\boldsymbol{u}_m^{2^d}; r_{m,2^d})$ where $r_{i,2^j} \leftarrow \mathbb{Z}_q$
3. $c_{v_0} = \mathrm{com}_{ck}(\boldsymbol{v}_0; s_0)$ where $\boldsymbol{v}_0 \leftarrow \mathbb{Z}_q^n$ and $s_0 \leftarrow \mathbb{Z}_q$
4. $\boldsymbol{\Delta}_0, \ldots, \boldsymbol{\Delta}_{dm-1} \in \mathbb{Z}_q^n$ such that $\overline{\boldsymbol{\Delta}}(X) = \sum_{i=0}^{dm-1} \boldsymbol{\Delta}_i X^i$ for $k = 1, \ldots, n$ satisfies

$$\sum_{i=1}^{m} l_i(X) \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d}^{(i,k)} \prod_{j=0}^{d} (\overline{\boldsymbol{u}}_{2^j,k}(X))^{i_j} - \overline{\boldsymbol{v}}_k(X) = l(X)\overline{\boldsymbol{\Delta}}_k(X)$$

   where we for $j = 0, \ldots, d$ define

$$\overline{\boldsymbol{u}}_{2^j}(X) = l(X)\boldsymbol{f}_j + \sum_{i=1}^{m} l_i(X)\boldsymbol{u}_i^{2^j} \qquad \overline{\boldsymbol{v}} = l(X)\boldsymbol{v}_0 + \sum_{i=1}^{m} l_i(X)\boldsymbol{v}_i$$

5. $c_{\Delta_0} = \mathrm{com}_{ck}(\boldsymbol{\Delta}_0; t_0), \ldots, c_{\Delta_{dm-1}} = \mathrm{com}_{ck}(\boldsymbol{\Delta}_{dm-1}; t_{dm-1})$ where $t_i \leftarrow \mathbb{Z}_q$

Send: $c_{f_0}, \ldots, c_{f_d}, c_{u_{1,2^1}}, \ldots, c_{u_{m,2^d}}, c_{v_0}, c_{\Delta_0}, \ldots, c_{\Delta_{dm-1}}$

Give in parallel d product arguments from Section 5.4 for $j = 0, \ldots, d-1$ with statements $(c_{u_{i,2^j}}, c_{u_{i,2^j}}, c_{u_{i,2^{j+1}}})_{i=1}^m$ using witnesses $\left(\boldsymbol{u}_{i,2^j}, r_{i,2^j}, \boldsymbol{u}_{i,2^j}, r_{i,2^j}, \boldsymbol{u}_{i,2^{j+1}}, r_{i,2^{j+1}}\right)_{i=1}^m$.

Challenge: $x \leftarrow \mathbb{Z}_q^* \setminus \Omega$

Answer: Compute for $j = 0, \ldots, d$

$$\overline{\boldsymbol{u}}_{2^j} = l(x)\boldsymbol{f}_j + \sum_{i=1}^{m} l_i(x)\boldsymbol{u}_{i,2^j} \qquad \overline{r}_{2^j} = l(x)r_{f_j} + \sum_{i=1}^{m} l_i(x) r_{i,2^j}$$

$$\overline{\boldsymbol{v}} = l(x)\boldsymbol{v}_0 + \sum_{i=1}^{m} l_i(x)\boldsymbol{v}_i \qquad \overline{s} = l(x)s_0 + \sum_{i=1}^{m} l_i(x)s_i \qquad \overline{t} = \sum_{i=0}^{dm-1} l_i(x)t_i$$

Send: $\overline{\boldsymbol{u}}_{2^0}, \overline{r}_{2^0}, \ldots, \overline{\boldsymbol{u}}_{2^d}, \overline{r}_{2^d}, \overline{\boldsymbol{v}}, \overline{s}, \overline{t}$

Verification: Compute $\overline{\boldsymbol{\Delta}} \in \mathbb{Z}_q^n$ such that for $k = 1, \ldots, n$

$$\sum_{i=1}^{m} l_i(x) \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d}^{(i,k)} \prod_{j=0}^{d} (\overline{\boldsymbol{u}}_{2^j,k})^{i_j} - \boldsymbol{v}_k = l(x)\overline{\boldsymbol{\Delta}}_k.$$

Accept if and only if for all $j = 0, \ldots, d$

1. $c_{f_j}^{l(x)} \prod_{i=1}^{m} c_{u_{i,2^j}}^{l_i(x)} = \mathrm{com}_{ck}(\overline{\boldsymbol{u}}_{2^j}; \overline{r_{2^j}})$
2. $c_{v_0}^{l(x)} \prod_{i=1}^{m} c_{v_i}^{l_i(x)} = \mathrm{com}_{ck}(\overline{\boldsymbol{v}}; \overline{s})$
3. $\prod_{i=0}^{dm-1} c_{\Delta_i}^{x^i} = \mathrm{com}_{ck}(\overline{\boldsymbol{\Delta}}; \overline{t})$
4. The d product arguments are accepting.

Theorem 27. *The protocol above is a public coin generalized $\Sigma-$protocol of committed values $u_{i,k}$ and $v_{i,k}$ satisfying $v_{i,k} = P^{(i,k)}(u_{i,k})$ for given polynomials $P^{(i,k)} \in \mathbb{Z}_q[x]$.*

Proof. Perfect completeness follows from the perfect completeness of the underlying product arguments.

For perfect SHVZK consider the simulator that is given a challenge $x \in \mathbb{Z}_q^*$ for which $l(x) \neq 0$. It picks $\overline{\boldsymbol{u}}_{2^0}, \ldots, \overline{\boldsymbol{u}}_{2^d} \leftarrow \mathbb{Z}_q^n$ and $\overline{r}_{2^0}, \ldots, \overline{r}_{2^d}, \overline{\boldsymbol{v}}, \overline{s}, \overline{t} \leftarrow \mathbb{Z}_q$ and $c_{u_{1,2^d}}, \ldots, c_{u_{m,2^d}}, c_{v_1}, \ldots, c_{v_m}$, $c_{\Delta_1}, \ldots, c_{\Delta_{dm-1}} \leftarrow \mathbb{G}$. Then it computes for $j = 0, \ldots, d$

$$c_{f_j} = \left(\prod_{i=1}^{m} c_{u_{i,2^j}}^{-l_i(x)} \mathrm{com}_{ck}(\overline{\boldsymbol{u}}_{2^j}; \overline{r_{2^j}}) \right)^{l(x)^{-1}} \qquad c_{v_0} = \left(\prod_{i=1}^{m} c_{v_i}^{-l_i(x)} \mathrm{com}_{ck}(\overline{\boldsymbol{v}}; \overline{s}) \right)^{l(x)^{-1}}$$

$$c_{\Delta_0} = \prod_{i=1}^{dm-1} c_{\Delta_i}^{-x^i} \mathrm{com}_{ck}(\overline{\boldsymbol{\Delta}}; \overline{t})$$

and runs the SHVZK simulator for the d product arguments.

To see this is a perfect simulation note that also in the real argument $\overline{\boldsymbol{u}}_{2^0}, \ldots, \overline{\boldsymbol{u}}_{2^d} \in \mathbb{Z}_q^n$ and $\overline{r}_{2^0}, \ldots, \overline{r}_{2^d}, \overline{\boldsymbol{v}}, \overline{s}, \overline{t} \in \mathbb{Z}_q$ and $c_{u_{1,2^1}}, \ldots, c_{u_{m,2^d}}, c_{v_1}, \ldots, c_{v_m}, c_{\Delta_1}, \ldots, c_{\Delta_{dm-1}} \in \mathbb{G}$ are uniformly random. Both in a real argument and a simulated argument, the commitments $c_{f_0}, \ldots, c_{f_d}, c_{v_0}, c_{\Delta_0}$ are uniquely determined by the verification equations once the other values are given. Therefore, the real argument and the simulated argument have identical probability distributions. Perfect SHVZK now follows from the perfect SHVZK of the d product arguments.

Finally, we will show that we have perfect special soundness. Given $(d+2)m$ accepting transcripts with different challenges x_i such that $l(x) \neq 0$, the extractor E works as follows.

The first m transcripts with challenges $x_1, \ldots, x_m$ for all $j = 0, \ldots, d$ satisfy the verification equations

$$c_{f_j}^{l(x)} \prod_{i=1}^{m} c_{u_{i,2^j}}^{l_i(x)} = \mathrm{com}_{ck}(\overline{\boldsymbol{u}}_{2^j}; \overline{r}_{2^j}) \qquad c_{v_0}^{l(x)} \prod_{i=1}^{m} c_{v_i}^{l_i(x)} = \mathrm{com}_{ck}(\overline{\boldsymbol{v}}; \overline{s}).$$

Note that for all $i = 1, \ldots, m$ we have $l_i(\omega_j) = \delta_{ij}$. Let $\alpha_i^{(1)}, \ldots, \alpha_i^{(m)}$ be the Lagrange interpolation coefficient for interpolation to polynomial evaluation in the point r_j such that

$$\sum_{k=1}^{m} \alpha_j^{(k)} l_i(x_k) = l_i(r_j) \text{ and } \sum_{k=1}^{m} \alpha_j^{(k)} l(x_k) = l(r_j).$$

This gives us

$$c_{v_i} = \prod_{k=1}^{m} \left(c_{v_0}^{l(x_k)} \prod_{i=1}^{m} c_{v_i}^{l_i(x_k)} \right)^{\alpha_k} = \mathrm{com}_{ck} \left(\sum_{k=1}^{m} \alpha_k \overline{\boldsymbol{v}}^{(k)}; \sum_{k=1}^{m} \alpha_k \overline{s}^{(k)} \right),$$

which provides an opening of c_{v_i}. By taking similar types of linear combinations of verification equations, we obtain openings of all the commitments $c_{f_0}, \ldots, c_{f_d}, c_{u_{1,2^1}}, \ldots, c_{u_{m,2^d}}, c_{v_0}, c_{\Delta_0}, \ldots, c_{\Delta_{dm-1}}$. Since the commitments are binding this means that the challenge x now uniquely determines the answers as

$$\overline{\boldsymbol{u}}_{2^j} = l(x)\boldsymbol{f}_j + \sum_{i=1}^{m} l_i(x)\boldsymbol{u}_{i,2^j} \qquad \overline{r}_{2^j} = l(x) r_{f_j} + \sum_{i=1}^{m} l_i(x) r_{i,2^j}$$

$$\overline{\boldsymbol{v}} = l(x)\boldsymbol{v}_0 + \sum_{i=1}^{m} l_i(x)\boldsymbol{v}_i \qquad \overline{s} = l(x)s_0 + \sum_{i=1}^{m} l_i(x)s_i \qquad \overline{t} = \sum_{i=0}^{dm-1} l_i(x)t_i$$

Furthermore, the d product arguments tell us $\boldsymbol{u}_{i,2^j} = \boldsymbol{u}_i^{2^j}$.

We now have $(d+2)m$ different points x, where the following polynomial equation of degree $(d+2)m - 1$ holds

$$\sum_{i=1}^{m} l_i(X) \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0 \ldots i_d}^{(i,k)} \prod_{j=0}^{d} (\overline{\boldsymbol{u}}_{2^j,k}(X))^{i_j} - \boldsymbol{v}_k(X) = l(X)\overline{\boldsymbol{\Delta}}_k(X).$$

This implies that the two polynomials are identical. With the two polynomials being identical, we deduce that the extracted openings for each $i = 1, \ldots, m$ and each $k = 1, \ldots, n$ satisfy $v_{i,k} = P^{(i,k)}(u_{i,k})$. Furthermore, the extracted openings are exactly the witnesses of the prover. If this is not the case, the extractor of the argument could be used by the prover to find a second opening of at least one of the commitments with non-negligible probability and the commitment scheme is broken. □

Efficiency: The communication cost is dominated by $2dm$ group elements, dn field elements and d product arguments for a total cost of $O(dm + dn) = O(\sqrt{L} \log D)$ group and field elements, assuming $m = n = \sqrt{L}$.

The verifier's computation is $2dm + dn$ exponentiations in $\mathbb{G}$ and $O(mnD) = O(LD)$ multiplications plus the cost of verifying d product arguments, for a total cost of $O(dm + dn) = O(\sqrt{L} \log D)$ exponentiations in $\mathbb{G}$ and $O(LD)$ multiplications in $\mathbb{Z}_q$.

The prover's computation is $2mnd$ exponentiations to compute the commitments, plus the cost of d product arguments, plus the cost of computing $\boldsymbol{\Delta}(X)$. This gives a total cost of $O(mnd) = O(L \log D)$ exponentiations in $\mathbb{G}$ and $O(Dmnd) = O(DL \log D) = \tilde{O}(DL)$ multiplications in $\mathbb{Z}_q$.

Let us illustrate how to get quasilinear computation complexity for the prover when we choose the roots of $l(X)$ as m-th roots of unity. The most expensive part in the prover's computation for very large D is to calculate the coefficients of $\boldsymbol{\Delta}(X)$. This is a degree $dm-1$ polynomial, so our strategy will be to evaluate it in dm different points and then use Lagrange-interpolation to get the coefficients. Recall, we defined $\boldsymbol{\Delta}(X)$ as the polynomial that for $k=1,\ldots,n$ satisfies

$$\sum_{i=1}^{m} l_i(X) \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d}^{(i,k)} \prod_{j=0}^{d} (\overline{\boldsymbol{u}}_{2^j,k}(X))^{i_j} - \overline{\boldsymbol{v}}_k(X) = l(X)\overline{\boldsymbol{\Delta}}_k(X)$$

where we for $j=0,\ldots,d$ defined

$$\overline{\boldsymbol{u}}_{2^j}(X) = l(X)\boldsymbol{f}_j + \sum_{i=1}^{m} l_i(X)\boldsymbol{u}_i^{2^j} \qquad \overline{\boldsymbol{v}} = l(X)\boldsymbol{v}_0 + \sum_{i=1}^{m} l_i(x)\boldsymbol{v}_i$$

It is easy to compute $\overline{\boldsymbol{v}}(X)$ and to evaluate $l_1(X),\ldots,l_m(X),l(X)$ in dm different points. Given those evaluations, we can compute dm evaluations of $\overline{\boldsymbol{u}}_{2^j}(X) = l(X)\boldsymbol{f}_j + \sum_{i=1}^{m} l_i(X)\boldsymbol{u}_i^{2^j}$ using $O(m^2d^2n)$ multiplications in a naïve implementation or in $O(mdn\log(md))$ using FFT techniques.

Having these values we can compute all the products $\prod_{j=0}^{d}(\overline{\boldsymbol{u}}_{2^j,k}(X))^{i_j}$ for all $i_0,\ldots,i_d$ in a binary-tree fashion using $Dmnd$ multiplications. Finally, we can rewrite the sum

$$\sum_{i=1}^{m} l_i(X) \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d}^{(i,k)} \prod_{j=0}^{d} (\overline{\boldsymbol{u}}_{2^j,k}(X))^{i_j} = \sum_{i_0,\ldots,i_d=0}^{1} \left(\sum_{i=1}^{m} l_i(X) a_{i_0\ldots i_d}^{(i,k)}\right) \prod_{j=0}^{d} (\overline{\boldsymbol{u}}_{2^j,k}(X))^{i_j}.$$

The inner component $\sum_{i=1}^{m} l_i(X)a_{i_0\ldots i_d}^{(i,k)}$ is a degree $m-1$ polynomial. It is easy to evaluate the polynomial in the roots of $l(X)$ using only m multiplications since $l_i(\omega_k)=\delta_{ik}$. If these roots are roots of unity, we furthermore get that an inverse FFT can be computed in time $O(m\log m)$ to get the coefficients of the polynomial. And in time $\tilde{O}(dm)$ we can now evaluate the polynomial in dm different points. We now have all the components to evaluate $\overline{\boldsymbol{\Delta}}(X)$ for dm different points at a cost of $O(Dmnd)=\tilde{O}(DL)$ multiplications. Using Lagrange interpolation, we can now compute the coefficients $\boldsymbol{\Delta}_0,\ldots,\boldsymbol{\Delta}_{dm-1}$.

Again, we only counted the dominant part of the calculations and ignored small constants. All values are conservative and can optimized even further by applying multi-exponentiation techniques, such as Brickell et al.'s algorithm from Section 4.3.3 or other optimization tricks [BGR98, Gro10].

6.5 Polynomial Evaluation Argument Based on Brands et al.'s [BDD07] techniques

We will describe for completeness how to use Brands et al.'s [BDD07] techniques to construct a polynomial argument with square root complexity. The original work states a non-membership proof, but it is easy to transform into a polynomial argument. To prove non-membership of a $u \in \mathbb{Z}_q$ to a list $\mathcal{L}=\{\lambda_1,\ldots,\lambda_D\}$, where without loss of generality $D=\mathfrak{d}^2$. Brands et al. construct $\mathfrak{d}=\sqrt{D}$ polyno-

mials

$$P_j(X) = \prod_{i=1}^{\mathfrak{d}} (X - \lambda_{(j-1)\mathfrak{d}+i})$$

and prove for each $j = 1, \ldots, \mathfrak{d}$ that $P_j(X)$ evaluated in u is not equal to 0. One part of the argument is to prove that $P_j(u) = v_j$ is indeed the polynomial evaluated on u. The prover also commits themselves to the values $u, u^2, u^3 \ldots, u^{\mathfrak{d}}$ and proves that these commitments contain the successive powers of u. We will use both protocols as building block of the polynomial argument.

In the case of the polynomial argument both parties know a polynomial

$$P(X) = a_D X^D + \ldots + a_1 X + a_0$$

and the prover wants to show that for committed values u, v it holds $P(u) = v$. To achieve the square root complexity we split the polynomial $P(X)$ in $\mathfrak{d}$ polynomials $P_j(X)$. In more detail we will rewrite

$$P(X) = P_d(X) X^{D-\mathfrak{d}} + \ldots + P_2(X) X^{\mathfrak{d}} + P_1(X),$$

where each polynomial is of degree $\mathfrak{d} = \sqrt{D}$.

The prover can now calculate $P_j(X) = v_j$ for all $j = 1, \ldots, \mathfrak{d}$ and use Brands et al.'s techniques to prove these equations. The prover also commits themselves to the values $u^{i\mathfrak{d}}$ in $c_{i\mathfrak{d}}, i = 2, \ldots, \mathfrak{d} - 1$, and $v_j u^{(j-1)\mathfrak{d}}$ in $c_{v_j^*}$ for $i = 2, \ldots, \mathfrak{d} - 1$. The prover then shows that the values $u^{i\mathfrak{d}}, i = 2, \ldots, \mathfrak{d} - 1$ are indeed successive powers of $u^{\mathfrak{d}}$ using the techniques given by Brands et al.. Lastly, by using simple product arguments 5.1 the prover also demonstrate that all the $P_j(u) u^{(j-1)\mathfrak{d}}$ are constructed correctly.

Finally, the verifier can check $\prod_{j=1}^{\mathfrak{d}} c_{v_j^*} = c_v$ which convince them that c_v contains the commitment of $P(X)$ evaluated on u.

Statement: $\mathbb{G},\ q,\ ck,$

$$P(X) = \sum_{i=0}^{D} a_i X^i = P_d(X) X^{D-\mathfrak{d}} + \ldots + P_2(X) X^{\mathfrak{d}} + P_1(X) \in \mathbb{Z}_q[X],$$

$P_j(X) \in \mathbb{Z}_q[X]$ are degree $\mathfrak{d}$ polynomials and $c_u, c_v \in \mathbb{G}$.

Prover's witness: $u, v, r_1, s \in \mathbb{Z}_q$ such that

$$v = P(u) \qquad c_{u_1} = \mathrm{com}_{ck}(u; r_1) \qquad c_v = \mathrm{com}_{ck}(v; s).$$

Initial message: Compute

1. $c_{u_i} = \mathrm{com}_{ck}(u^i; r_i)$ where $r_i \leftarrow \mathbb{Z}_q, i = 2, \ldots, \mathfrak{d}$
2. $c_{u_{i\mathfrak{d}}} = \mathrm{com}_{ck}(u^{i\mathfrak{d}}; r_{i\mathfrak{d}})$ where $r_{i\mathfrak{d}} \leftarrow \mathbb{Z}_q, i = 2, \ldots, \mathfrak{d} - 1$
3. $v_j = P_j(u)$ and $w_j = a_{j,\mathfrak{d}} r_{\mathfrak{d}} + \ldots a_{j,2} r_2 + a_{j,1} r_1, \quad j = 1, \ldots \mathfrak{d}$
4. $c_{v_i} = \mathrm{com}_{ck}(v_i; w_i), \quad i = 1, \ldots, \mathfrak{d}$

5. $c_{v_i^*} = \mathrm{com}_{ck}(v_i u^{(i-1)\mathfrak{d}}; s_i), \quad i = 2, \ldots, \mathfrak{d}$ where $s_i \leftarrow \mathbb{Z}_q, \quad s_\mathfrak{d} = s - \sum_{i=2}^{\mathfrak{d}-1} s_i - w_1$
6. $f_1 = \mathrm{com}_{ck}(r_u; t_1) \quad f_i = c_{u^{i-1}}^{r_u} \mathrm{com}_{ck}(0; t_i), i = 2, \ldots, \mathfrak{d}$ where $r_u \leftarrow \mathbb{Z}_q, t_i \leftarrow \mathbb{Z}_q$
7. $f_{i\mathfrak{d}} = c_{u_{(i-1)\mathfrak{d}}}^{r_{u_\mathfrak{d}}} \mathrm{com}_{ck}(0; t_{i\mathfrak{d}})$ where $r_{u_\mathfrak{d}}, t_{i\mathfrak{d}} \leftarrow \mathbb{Z}_q, i = 2, \ldots, \mathfrak{d} - 1$
8. $c_{d_i} = \mathrm{com}_{ck}(d_i; \rho_i) \quad c_e = \mathrm{com}_{ck}(e_2, \sigma_i)$ where $d_i, \rho_i, e_i, \sigma_i \leftarrow \mathbb{Z}_q$ for $i = 2, \ldots, \mathfrak{d}$
   $c_{f_i} = c_{u^{(i-1)\mathfrak{d}}}^{d_i} \mathrm{com}_{ck}(0; \rho_i)$ for $i = 2, \ldots, \mathfrak{d}$

Send: $c_{u_2}, \ldots, c_{u_\mathfrak{d}}, c_{u_{2\mathfrak{d}}}, \ldots, c_{u_{(\mathfrak{d}-1)\mathfrak{d}}}, c_{v_1}, \ldots, c_{v_\mathfrak{d}}, c_{v_2^*}, \ldots, c_{v_\mathfrak{d}^*}, f_1, \ldots, f_d, f_{2\mathfrak{d}}, \ldots, f_{(\mathfrak{d}-1)\mathfrak{d}}$, $c_{d_2}, \ldots, c_{d_\mathfrak{d}}, c_{e_1}, \ldots, c_{e_\mathfrak{d}}, c_{f_1}, \ldots, c_{f_\mathfrak{d}}$

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Compute

1. $\overline{u} = xu + r_u \qquad \overline{u}_\mathfrak{d} = xu^\mathfrak{d} + r_{u_\mathfrak{d}}$
2. $\overline{r}_1 = xr_1 + t_1, \qquad \overline{r}_i = x(r_i - ur_{i-1}) + t_i \quad$ for $i = 2, \ldots, \mathfrak{d}$
3. $\overline{r}_{i\mathfrak{d}} = x(r_{i\mathfrak{d}} - u^\mathfrak{d} r_{(i-1)\mathfrak{d}}) + t_i, \quad$ for $\quad i = 2, \ldots, \mathfrak{d} - 1$
4. $\overline{a}_i = d_i + xv_i \qquad \overline{r}_i = \rho_i + xw_i \qquad \overline{v}_i = e_i + xu^{(i-1)\mathfrak{d}} \qquad \overline{s}_i = \sigma_i + xr_{i\mathfrak{d}}$
   $\overline{t}_i = \rho_i - x(r_{i\mathfrak{d}} v_i - s_i)$ for $i = 2, \ldots, \mathfrak{d}$

Send: $\overline{u}, \overline{u}_\mathfrak{d}, \overline{r_1}, \ldots, \overline{r_\mathfrak{d}}, \overline{r}_{2\mathfrak{d}}, \ldots, \overline{r}_{(\mathfrak{d}-1)\mathfrak{d}}, \overline{a}_2, \ldots, \overline{a}_\mathfrak{d}, \overline{r}_1, \ldots, \overline{r}_\mathfrak{d}, \overline{b}_2, \ldots, \overline{b}_\mathfrak{d}, \overline{s}_2, \ldots, \overline{s}_\mathfrak{d}, \overline{t}_2, \ldots, \overline{t}_\mathfrak{d}$

Verification: Accept if and only if

1. $c_{u_1}, \ldots, c_{u_\mathfrak{d}}, c_{v_1}, \ldots, c_{v_\mathfrak{d}}, f_1, \ldots, f_\mathfrak{d}, c_{u_{2\mathfrak{d}}}, \ldots, c_{u_{(\mathfrak{d}-1)\mathfrak{d}}}, f_{2\mathfrak{d}}, \ldots, f_{(\mathfrak{d}-1)\delta} \in \mathbb{G}$,
   $c_{v_2^*}, \ldots, c_{v_\mathfrak{d}^*}, c_{\alpha_2}, \ldots, c_{\alpha_\mathfrak{d}}, c_{\beta_2}, \ldots, c_{\beta_\mathfrak{d}} \in \mathbb{G}$
2. $\overline{u}, \overline{u}_\mathfrak{d}, \overline{r}_1, \ldots, \overline{r}_\mathfrak{d}, \overline{r}_{2\mathfrak{d}}, \ldots, \overline{r}_{(\mathfrak{d}-1)\mathfrak{d}}, \overline{a}_2, \ldots, \overline{a}_{(\mathfrak{d})\mathfrak{d}}, \overline{r}_2, \ldots, \overline{r}_\mathfrak{d} \in \mathbb{Z}_q$,
   $\overline{b}_2, \ldots, \overline{b}_\mathfrak{d}, \overline{s}_2, \ldots, \overline{s}_\mathfrak{d}, \overline{t}_2, \ldots, \overline{t}_\mathfrak{d} \in \mathbb{Z}_q$
3. $\prod_{j=1}^{\mathfrak{d}} c_{v_j}^{\sigma_j} = G^{\sum_{j=1}^{\mathfrak{d}} a_{j,0}\sigma_j} \prod_{i=1}^{\mathfrak{d}} c_{u_{\mathfrak{d}_i}}^{\sum_{j=1}^{\mathfrak{d}} a_{j,i}\sigma_j}$ where $\sigma_j \leftarrow \mathbb{Z}_q \quad$, for $\quad j = 1, \ldots, \mathfrak{d}$
4. $\mathrm{com}_{ck}(\overline{u}; \overline{r}_1) = f_1 c_{u_1}^x \qquad c_{u_{i-1}}^{\overline{u}} \mathrm{com}_{ck}(0; \overline{r}_i) = f_i c_{u_i}^x$ for $i = 2, \ldots, \mathfrak{d}$
   $c_{u_{(i-1)\mathfrak{d}}}^{\overline{u}_\mathfrak{d}} \mathrm{com}_{ck}(0; \overline{r}_{i\mathfrak{d}}) = f_{i\mathfrak{d}} c_{u_{i\mathfrak{d}}}^x$ for $i = 2, \ldots, \mathfrak{d} - 1$
5. $c_{v_i}^x c_{d_i} = \mathrm{com}_{ck}(\overline{a}_i; \overline{r}_i) \quad c_{u_{(i-1)\mathfrak{d}}}^x c_{e_i} = \mathrm{com}_{ck}(\overline{b}_i; \overline{s}_i)$
   $c_{v_i^*}^x c_{f_i} = c_{u_{(i-1)\mathfrak{d}}}^{\overline{v_i}} \mathrm{com}_{ck}(0; \overline{t}_i), \quad$ for $i = 1, \ldots, \mathfrak{d}$
6. $c_{v_1} \prod_{i=2}^{\mathfrak{d}} c_{v_i^*} = c_v$

Theorem 28. *The protocol above is a public key perfect generalized $\Sigma-$protocol of committed values u, v satisfying $v = P(u)$ for given polynomial P(X).*

Proof. Perfect completeness follows from careful inspection of the verification equations.

Next, we have to show that the argument has perfect SHVZK. The simulator picks on challenge x random answers $\overline{u}, \overline{u}_\mathfrak{d}, \overline{r}_1, \ldots, \overline{r}_\mathfrak{d}, \overline{r}_{2\mathfrak{d}}, \ldots, \overline{r}_{(\mathfrak{d}-1)\mathfrak{d}}, \overline{a}_2, \ldots, \overline{a}_\mathfrak{d}, \overline{r}_1, \ldots, \overline{r}_\mathfrak{d}, \overline{b}_2, \ldots, \overline{b}_\mathfrak{d}, \overline{s}_2, \ldots, \overline{s}_\mathfrak{d}, \overline{t}_2, \ldots, \overline{t}_\mathfrak{d}$ and $c_{u_2}, \ldots, c_{u_\mathfrak{d}}, c_{2\mathfrak{d}}, \ldots, c_{(\mathfrak{d}-1)\mathfrak{d}}, c_{v_1}*, \ldots, c_{v_\mathfrak{d}}*$ as random commitments to 0. It then sets

$$f_1 = c_{u_1}^{-x} \mathrm{com}_{ck}(\overline{u}, \overline{r}_1), \qquad f_i = c_{u_{i-1}}^{\overline{u}} \mathrm{com}_{ck}(0; \overline{r}) c_{u_i}^{-x}, \quad \text{for } i = 2, \ldots \mathfrak{d}$$

$$f_{i\mathfrak{d}} = c_{u_{(i-1)\mathfrak{d}}}^{\overline{u}_{\mathfrak{d}}} \mathrm{com}_{ck}(0; \overline{r}_{i\mathfrak{d}}) c_{u_{i\mathfrak{d}}}^{-x} \quad \text{for } i = 2, \ldots, \mathfrak{d}-1$$

$$c_{d_i} = c_{v_i}^{x} \mathrm{com}_{ck}(\overline{v}_i; \overline{w}_i) \quad c_{e_i} = c_{v_i^*}^{x} \mathrm{com}_{ck}(\overline{v}_i^*; \overline{s_i}$$

$$c_{f_i} = c_{v_i^*}^{-x} c_{u_{(i-1)\mathfrak{d}}}^{\overline{v}_i} \mathrm{com}_{ck}(0; \overline{s_i}), \quad \text{for } i = 1, \ldots, \mathfrak{d}.$$

The simulator also picks $\sigma_i \leftarrow \mathbb{Z}_q$ and sets

$$c_{v_1} = \left[G^{\sum_{j=1}^{\mathfrak{d}} a_{j,0}\sigma_j} \prod_{i=1}^{\mathfrak{d}} c_{u_{\mathfrak{d}_i}}^{\sum_{j=1}^{\mathfrak{d}} a_{j,i}\sigma_j} \prod_{j=2}^{\mathfrak{d}} c_{v_j}^{-\sigma_j} \right]^{\sigma_1^{-1}} \text{ and } c_v = c_{v_1} \prod_{i=2}^{d} c_{v_i^*}.$$

In the real argument the answers are uniformly random distributed. In the simulation the values are picked at random and therefore they follow the same distribution. The commitment scheme is hiding; therefore, the commitments of the simulation and the real argument follow the same distribution. So, the argument is perfect SHVZK.

Finally, we have to see that we have perfect generalized special soundness. Given two accepting transcripts with different challenges x_1, x_2 the extractor E can take linear combinations of the verification equations to get openings of all commitments. The extracted witnesses satisfies the statement. As the commitment scheme is binding we can conclude that the commitments c_{v_i*} contains openings $v_i u^{(i-1)\mathfrak{d}}$ and from $c_v = c_{v_1} \prod_{i=2}^{\mathfrak{d}} c_{v_i^*}$ that $P(u) = v$. Therefore, the extracted witnesses satisfy the statement. Moreover, the extracted openings are exactly the witnesses of the prover. Otherwise, the extractor of the argument could be used by the prover to find a second opening of at least one of the commitments with non-negligible probability and the commitment scheme is broken. □

Efficiency: During the protocol $9\mathfrak{d}$ group elements and $7\mathfrak{d}$ fields elements are sent between the parties.

The prover has to calculate $18\mathfrak{d}$ exponentiations in $\mathbb{G}$ and $D + 19\mathfrak{d}$ multiplications, whereas the verifier has to calculate $18\mathfrak{d}$ exponentiations and $D + 19\mathfrak{d}$ multiplications in $\mathbb{G}$.

In the original publication [BDD07] Brands et al. use in step three of the verification $\mathfrak{d}$ equations which leads to a computation cost of $O(D)$, but they report themselves of a way to batch these equations into a single one, which cost only $O(\mathfrak{d})$ exponentiations. Naturally we chosen the more optimized version in our description.

The sliding window algorithm, Section 4.3.2, can be used to reduce the computational cost of the prover and verifier further. Other optimization techniques [BGR98, Gro10] could be applied as well; thus, the values above are conservative.

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 149 \rangle \subset \mathbb{Z}_{179}^*$, which has prime order $q = 89$. The statement consists of $\{\mathbb{G}, p, q\} = \{\langle 149 \rangle, 179, 89\}$, $ck = \{149, 129\}$ and the polynomials

$$\begin{aligned} P(X) &= X^8 + 40X^7 + 53X^6 + 7X^5 + 68X^4 + 32X^3 + 17X^2 + 68X + 70 \\ &= (X^2 + 40)X^6 + (53X^3 + 7X^2 + 68X)X^3 + (32X^3 + 17X^2 + 68X + 70) \\ &= P_3(X)X^6 + P_2(X)X^3 + P_1(X), \end{aligned}$$

of $\mathfrak{d} = \lceil\sqrt{8}\rceil = 3$ and of the commitments $c_{u_1} = 49$ and $c_v = 47$.

The prover knows witnesses $u = 47, v = P(u) = 10$ and $r_1 = 57, s = 77$ such that

$$c_{u_1} = \mathrm{com}_{ck}(u; r_1) = 49 \qquad c_v = \mathrm{com}_{ck}(v; s) = 47.$$

To convince the verifier of this claim the prover commits themselves to u^2, u^3 in

$$c_{u_2} = \mathrm{com}_{ck}(u^2; r_2) = 29, \quad c_{u_3} = \mathrm{com}_{ck}(u^3; r_3) = 108$$

with random picked $r_2 = 80, r_3 = 25$. Picking $r_{2\mathfrak{d}} = 52$ the prover can calculate the commitments to

$$c_{u_{2\mathfrak{d}}} = \mathrm{com}_{ck}(u^{2\mathfrak{d}}; r_{2\mathfrak{d}}) = \mathrm{com}_{ck}(u^4; r_{2\mathfrak{d}}) = 82.$$

Then the prover calculates

$$v_1 = P_1(u) = 63 \qquad v_2 = P_2(u) = 88 \qquad v_3 = P_3(u) = 24,$$

and $w_i = a_{i,\mathfrak{d}} r_{\mathfrak{d}} + \ldots a_{i,2} r_2 + a_{i,1} r_1$ for $i = 1, 2, 3$

$$w_1 = 11 \qquad w_2 = 3 \qquad w_3 = 20,$$

and lastly the prover computes $c_{v_i} = \mathrm{com}_{ck}(v_i; w_i)$

$$c_{v_1} = 88 \qquad c_{v_2} = 80 \qquad c_{v_3} = 51.$$

In the next step the prover sets $s_2 = 1$ and $s_3 = 4$, and calculates $c_{v_i *} = \mathrm{com}_{ck}(v_i u^{(i-1)\mathfrak{d}}; s_i)$

$$c_{v_2 *} = 5 \qquad c_{v_3 *} = 177.$$

Now the prover picks $r_u = 44, t_1 = 45, t_2 = 57, t_3 = 77$ and computes

$$f_1 = 172 \qquad f_2 = 20 \qquad f_3 = 81,$$

and for random $r_{u_\delta} = 26, t_{2\mathfrak{d}} = 81$ they compute $f_{2\mathfrak{d}} = 155$.

For the underlying simple product argument, the prover picks $d_2 = 34, d_3 = 88, \rho_2 = 36, \rho_3 = 16$, $e_3 = 9, e_3 = 71, \sigma_2 = 5, \sigma_3 = 17$ and calculates

$$c_{d_2} = 89 \quad c_{d_3} = 4 \quad c_{e_2} = 87 \quad c_{d_3} = 4 \quad c_{f_2} = 139 \quad c_{f_3} = 153.$$

The prover sends all the commitments

$$c_{u_2} = 29 \quad c_{u_3} = 108 \quad c_{u_{2\mathfrak{d}}} = 82$$

$$c_{v_1} = 88 \quad c_{v_2} = 80 \quad c_{v_3} = 51 \quad c_{v_2*} = 5 \quad c_{v_3*} = 177$$

$$f_1 = 172 \quad f_2 = 20 \quad f_3 = 81 \quad f_{2d} = 155$$

$$c_{d_2} = 89 \quad c_{d_3} = 4 \quad c_{e_2} = 87 \quad c_{e_3} = 4 \quad c_{f_1} = 139 \quad c_{f_2} = 153,$$

to the verifier, who picks a challenges $x = 26$.

To answer the challenge the prover calculates

$$\overline{u} = xu + r_u = 78 \qquad \overline{u}_{\mathfrak{d}} = 22 \qquad \overline{r}_1 = 8 \qquad \overline{r}_2 = 52 \qquad \overline{r}_3 = 54 \qquad \overline{r}_{2\mathfrak{d}} = 20$$

$$\overline{a}_2 = d_2 + xv_2 = 8 \quad \overline{a}_3 = d_3 + xv_3 = 0 \quad \overline{r}_2 = \rho_2 + xw_2 = 25 \quad \overline{w}_3 = \rho_3 + xw_3 = 2$$

$$\overline{b}_2 = e_2 + u^{\mathfrak{d}} = 37 \quad \overline{b}_3 = e_3 + u^{2\mathfrak{d}} = 19 \quad \overline{s}_2 = \sigma_i + xr_{\mathfrak{d}} = 32 \quad \overline{s}_3 = \sigma_i + xr_{2\mathfrak{d}} = 34$$

$$\overline{t}_2 = \rho_2 - x(r_{\mathfrak{d}}v_2 - s_2) = 0 \quad \overline{t_3} = \rho_2 - x(r_{2\mathfrak{d}}v_3 - s_3) = 68$$

and sends these values back to the verifier.

The verifier first checks if all answers are valid and the commitments are really in the group $\mathbb{G}$. He then picks random $\sigma_1 = 64, \sigma_2 = 81, \sigma_3 = 73$ and calculates the sums

$$\sum_{i=1}^{3} a_{i,0}\sigma_i = 30 \qquad \sum_{i=1}^{3} a_{i,1}\sigma_i = 53 \qquad \sum_{i=1}^{3} a_{i,2}\sigma_i = 37 \qquad \sum_{i=1}^{3} a_{i,2}\sigma_i = 33.$$

and checks

$$\prod_{j=1}^{\mathfrak{d}} c_{v_j}^{\sigma_j} = 172 = G^{\sum_{j=1}^{\mathfrak{d}} a_{j,0}\sigma_j} \prod_{i=1}^{\mathfrak{d}} c_{u^{\mathfrak{d}}i}^{\sum_{j=1}^{\mathfrak{d}} a_{j,i}\sigma_j} \quad (\checkmark)$$

He also checks

$$f_1 = 171 = c_{u_1}^{-x} G^{\overline{u}+e\overline{r}_1} \quad (\checkmark) \qquad f_{2d} = 155 = c_{u_3}^{\overline{u}_\delta} H^{\overline{r_{2\mathfrak{d}}}} c_{u_2\mathfrak{d}}^{-x} \quad (\checkmark)$$

$$f_2 = 20 = c_{u_1}^{\overline{u}} H^{\overline{r_2}} c_{u_2}^{-x} \quad (\checkmark) \qquad f_3 = 81 = c_{u_2}^{\overline{u}} H^{\overline{r_3}} c_{u_3}^{-x} \quad (\checkmark)$$

$$c_{u_2}^{x} c_{d_2} = 177 = \mathrm{com}_{ck}(\overline{a}_2; \overline{r}_2) \quad (\checkmark) \qquad c_{u_3}^{x} c_{d_3} = 173 = \mathrm{com}_{ck}(\overline{a}_3; \overline{r}_3) \quad (\checkmark)$$

$$c_{u_3}^{x} c_{e_2} = 13 = \mathrm{com}_{ck}(\overline{b}_2; \overline{s}_2) \quad (\checkmark) c_{u_{(2\mathfrak{d}}}^{x} c_{e_3} = 95 = \mathrm{com}_{ck}(\overline{b}_3; \overline{s}_3) \quad (\checkmark)$$

$$c_{v_2^*}^{x} c_{f_2} = 52 = c_{u_3}^{\overline{a_2}} \mathrm{com}_{ck}(0, \overline{t}_2) \quad (\checkmark) \qquad c_{v_3^*}^{x} c_{f_3} = 116 = c_{u_{2\mathfrak{d}}}^{\overline{a_3}} \mathrm{com}_{ck}(0, \overline{t}_3) \quad (\checkmark)$$

$$c_v = 45 = c_{v_1} c_{v_2*} c_{v_3*} \quad (\checkmark).$$

Now the verifier is convinced that c_v contains a commitment of $v = P(u)$.

6.5.1 Implementation and practical Results

We implemented Brands et al.'s argument to obtain some experimental results and to analyze the real life behavior of the protocol. We used different modular subgroups with different security levels. Security levels were estimated following [Len04] and can be found in Section 4.1.1. We chose two 160-bit subgroups modulo a 1 248 and a 1 536-bit prime, we also chosen three different 256-bit subgroups modulo a 1 536-bit, a 2 432-bit, and a 3 248-bit prime, and the last two groups we use are 384-bit subgroups modulo a 3 248-bit and a 7 936-bit prime.

Some of these groups are not standard and are not used in the research community, the reason that we picked these groups is to explore the merit of different parameters, for example subgroup size vs group size.

For the protocol we build a conservative version which contains the level of optimization we used to calculate the computation cost of the parties, that means we only optimized processes which contain a lot of multiplications. We also implemented two levels of optimization, firstly we only used the sliding window technique and secondly we looked at the merit of Lim-Lee's technique and Brickells et al.'s technique, for polynomials with degree $< 1\,000$ we used Lim-Lee's algorithm otherwise sticked with Brickells et al.'s technique. To obtain the results we run the protocols for each set of parameters 100 times and calculated the mean. The experiments were carried out on a Mac Book Pro with 2.54 GHz CPU and 4 GB RAM.

	Conservative	SW	Optimized	Ratio	
D	Prover			SW/Cons.	Opt./Cons.
10	38 ms	24 ms	24 ms	0.64	0.64
100	118 ms	75 ms	75 ms	0.63	0.63
1 000	409 ms	257 ms	257 ms	0.63	0.63
5 000	915 ms	584 ms	582 ms	0.64	0.64
10 000	1 297 ms	828 ms	826 ms	0.64	0.64
50 000	2 946 ms	1 887 ms	1 882 ms	0.64	0.64
100 000	4 204 ms	2 699 ms	2 691 ms	0.64	0.64
500 000	9 643 ms	6 292 ms	6 273 ms	0.65	0.65
1 000 000	13 916 ms	9 169 ms	9 153 ms	0.66	0.66
	Verifier				
10	38 ms	33 ms	30 ms	0.86	0.80
100	111 ms	95 ms	89 ms	0.85	0.80
1 000	376 ms	322 ms	299 ms	0.86	0.80
5 000	847 ms	725 ms	675 ms	0.86	0.80
10 000	1 207 ms	1 027 ms	954 ms	0.85	0.79
50 000	2 714 ms	2 322 ms	2 160 ms	0.86	0.80
100 000	3 853 ms	3 300 ms	3 072 ms	0.86	0.80
500 000	8 726 ms	7 484 ms	6 980 ms	0.86	0.80
1 000 000	12 646 ms	10 790 ms	10 018 ms	0.85	0.79

Table 6.11: Comparison of Brands et al.'s polynomial argument for different degree D and different levels of optimization on a 256-bit subgroup modulo a 1 536-bit prime.

The results for all three optimization levels for a 256-bit subgroup modulo a 1 536-bit prime can be found in Table 6.11, together with the ratio of the optimized versions against the conservative one.

D	Conservative	SW	Optimized	Ratio	
	Prover			SW/Cons.	Opt./Cons
10	206 ms	118 ms	118 ms	0.57	0.57
100	655 ms	370 ms	370 ms	0.57	0.57
1 000	2 251 ms	1 275 ms	1 270 ms	0.57	0.57
5 000	5 084 ms	2 882 ms	2 887 ms	0.57	0.57
10 000	7 196 ms	4 080 ms	4 087 ms	0.57	0.57
50 000	16 241 ms	9 235 ms	9 240 ms	0.57	0.57
100 000	23 050 ms	13 114 ms	13 132 ms	0.57	0.57
500 000	51 956 ms	29 733 ms	29 756 ms	0.57	0.57
1 000 000	73 835 ms	42 428 ms	42 486 ms	0.57	0.58
	Verifier				
10	205 ms	173 ms	158 ms	0.84	0.77
100	623 ms	518 ms	464 ms	0.83	0.74
1 000	2 125 ms	1 277 ms	1 568 ms	0.83	0.74
5 000	4 787 ms	3 961 ms	3 532 ms	0.83	0.74
10 000	6 755 ms	5 595 ms	4 991 ms	0.83	0.74
50 000	15 239 ms	12 643 ms	11 241 ms	0.83	0.74
100 000	21 911 ms	17 911 ms	15 941 ms	0.82	0.74
500 000	48 525 ms	40 211 ms	35 795 ms	0.83	0.74
1 000 000	68 818 ms	57 031 ms	50 818 ms	0.83	0.74

Table 6.12: Comparison of Brands et al.'s polynomial argument for different degree D and different levels of optimization on a 384-bit subgroup modulo a 3 248-bit prime.

In Table 6.12 we have the same date for a 384-bit subgroup modulo a 3 248-bit prime. We see that the influence of the sliding window technique on the provers performance gets higher for increasing subgroup and group parameters; but inside a setting the merit is the same for all degrees D.

For the verifier the effect of the sliding window technique is less, but again this is the same for all degree D. One reason for this reduced influence is that the verifier only calculates $8\sqrt{D}$ exponentiations, which can be optimized by this techniques, the other $8\sqrt{D}$ exponentiations are single exponentiations in the first version. This last part was optimized in the second optimization level and we see that Lim-Lee's as well Brickell et al's techniques speed up the verifier further.

The linear number of multiplications do not become dominant over the cost of the exponentiations; however, they have a small noticeable influence on the run-time, see Figure 6.3 for the prover and Figure 6.4 for the verifier. On the first glare it looks like both parties' run-times follow a square root function, but for bigger degree D the cost of the multiplication can be seen.

Table 6.13 states the optimized results of a 256-bit subgroup modulo a 1 536-bit prime, a 2 432-bit prime, and a 3 248-bit prime. Doubling the size of the modulo from 1 536-bit to 3 248-bit has a big effect on the run-time, the time gets nearly quadrupled for big D. A similar effect can be seen for $|p| = 2\,432$, in this case p grows by around 1.5 but the run-time doubles itself. One possible explanation for this behavior is that the cost of exponentiations gets higher for bigger prime $|p|$ and this cost dominates the run-time. Equal results can be seen for fixed 160-bit subgroup modulo a 1 248 and a 1 536-bit prime, see Table 6.14.

The effect of increasing the subgroup size for fixed primes can be found in Table 6.15 for $|p| = 1\,536$

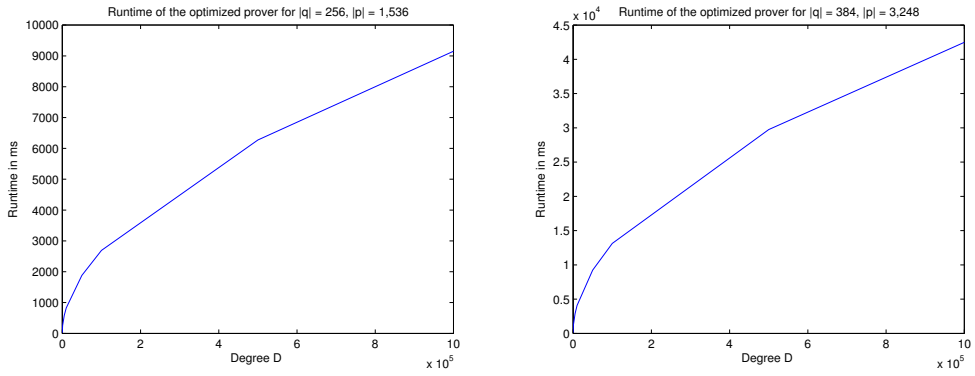


Figure 6.3: Run-time of Brands et al.'s polynomial argument prover plotted against the degree D for $|q| = 256$ and $|p| = 1\,536$, and $|q| = 384$ and $|p| = 3\,248$

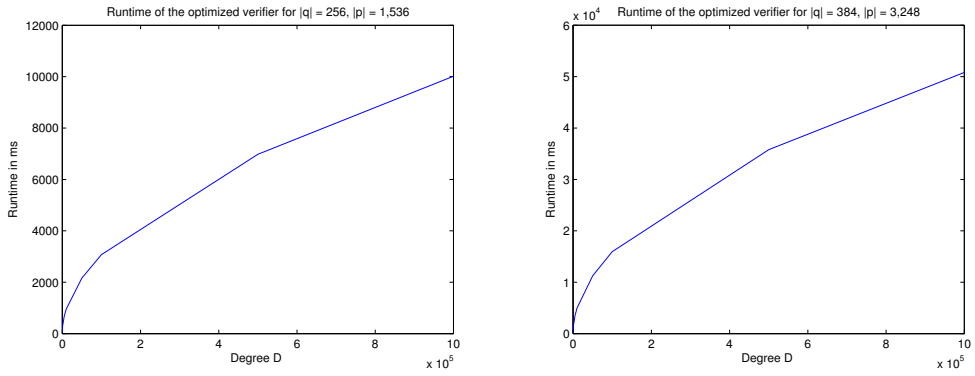


Figure 6.4: Run-time of Brands et al.'s polynomial argument verifier plotted against the degree D for $|q| = 256$ and $|p| = 1\,536$, and $|q| = 384$ and $|p| = 3\,248$

$\|q\| = 256$	$\|p\| = 1\,536$		$\|p\| = 2\,432$		$\|p\| = 3\,248$	
D	Prover	Verifier	Prover	Verifier	Prover	Verifier
10	24 ms	30 ms	52 ms	70 ms	81 ms	110 ms
100	75 ms	89 ms	162 ms	204 ms	254 ms	323 ms
1 000	257 ms	299 ms	560 ms	691 ms	875 ms	1 091 ms
5 000	582 ms	675 ms	1 267 ms	1 556 ms	1 978 ms	2 455 ms
10 000	826 ms	954 ms	1 795 ms	2 201 ms	2 800 ms	3 470 ms
50 000	1 882 ms	2 160 ms	4 065 ms	4 959 ms	6 329 ms	7 816 ms
100 000	2 691 ms	3 072 ms	5 783 ms	7 038 ms	8 995 ms	11 090 ms
500 000	6 273 ms	6 980 ms	13 184 ms	15 842 ms	20 375 ms	24 909 ms
1 000 000	9 153 ms	10 018 ms	18 749 ms	22 613 ms	29 088 ms	35 393 ms

Table 6.13: Comparison of Brands et al.'s optimized polynomial argument for different degree D a on 256-bit subgroups modulo a 1 536-bit, a 2 432-bit, and a 3 248-bit prime.

and $|q| = 160, 256$ and in Table 6.16 we can find the results for a 256-bit subgroup and a 384-bit subgroup modulo a 3 248-bit prime. We see that increasing the subgroup size by one and a half time, for example from 160-bit to 256-bit, has a similar increase on the run-time. That means by increasing the security level by a certain factor we have to expect a similar increase on the run-time.

In conclusion we can say that Brands et al.'s polynomial argument is practical for moderate degrees

$\|q\| = 160$	$\|p\| = 1\,248$		$\|p\| = 1\,536$	
D	Prover	Verifier	Prover	Verifier
10	12 ms	15 ms	16 ms	19 ms
100	38 ms	42 ms	48 ms	56 ms
1 000	130 ms	144 ms	167 ms	88 ms
5 000	294 ms	322 ms	379 ms	424 ms
10 000	421 ms	457 ms	539 ms	600 ms
50 000	967 ms	1 035 ms	1 232 ms	1 360 ms
100 000	1 395 ms	1 476 ms	1 770 ms	1 933 ms
500 000	3 336 ms	3 402 ms	4 172 ms	4 413 ms
1 000 000	4 923 ms	4 951 ms	6 133 ms	6 389 ms

Table 6.14: Comparison of Brands et al.'s optimized polynomial argument for different degree D a on 160-bit subgroups modulo a 1 248-bit, and a 1 526-bit prime.

$\|p\| = 1\,536$	$\|q\| = 160$		$\|q\| = 256$	
D	Prover	Verifier	Prover	Verifier
10	16 ms	19 ms	24 ms	30 ms
100	48 ms	56 ms	75 ms	89 ms
1 000	167 ms	188 ms	257 ms	299 ms
5 000	379 ms	424 ms	582 ms	675 ms
10 000	539 ms	600 ms	826 ms	954 ms
50 000	1 232 ms	1 360 ms	1 882 ms	2 160 ms
100 000	1 770 ms	1 933 ms	2 691 ms	3 072 ms
500 000	4 172 ms	4 413 ms	6 273 ms	6 980 ms
1 000 000	6 133 ms	6 389 ms	9 153 ms	10 018 ms

Table 6.15: Comparison of Brands et al.'s optimized polynomial argument for different degree D a on 160-bit and a 256-bit subgroup modulo a 1 526-bit prime.

$\|p\| = 3\,248$	$\|q\| = 256$		$\|q\| = 384$	
D	Prover	Verifier	Prover	Verifier
10	81 ms	110 ms	118 ms	158 ms
100	254 ms	323 ms	370 ms	464 ms
1 000	875 ms	1 091 ms	1 272 ms	1 568 ms
5 000	1 978 ms	2 455 ms	2 887 ms	3 532 ms
10 000	2 800 ms	3 470 ms	4 087 ms	4 991 ms
50 000	6 329 ms	7 816 ms	9 240 ms	11 241 ms
100 000	8 995 ms	11 090 ms	13 132 ms	15 941 ms
500 000	20 375 ms	24 909 ms	29 756 ms	35 795 ms
1 000 000	29 088 ms	34 393 ms	42 486 ms	50 818 ms

Table 6.16: Comparison of Brands et al.'s optimized polynomial argument for different degree D a on a 256-bit and a 384 subgroups modulo a 3,248-bit prime.

D. But since the square root complexity influences the run-time for big D heavily, the argument is only feasible for big degree D's if the security parameter is small. For long time security the impact is to big even taking in account that for better machines the values speed up and more sophisticated programming techniques can also speed up the program further.

Lastly, we have to analyze the data size, Table 6.17 states the data size for each round for a 256-bit subgroup and a 1 536-bit prime p. Round 2 consists only of 1 field element and can be neglected. We

expect a group element from $\mathbb{G}$ to be 6 times bigger that a field element and $8\mathfrak{d}$ elements are sent in round 1, versus $5\mathfrak{d}$ field element in round 5 and we expect the argument size of round 1 to be around 9.5 times bigger than round 3. The date supports this assumption.

D	Round 1	Round 2	Round 3	Statement
10	12 KB	0.08 KB	2 KB	1 KB
100	37 KB	0.08 KB	5 KB	8 KB
1 000	126 KB	0.08 KB	17 KB	77 KB
5 000	285 KB	0.08 KB	38 KB	381 KB
10 000	403 KB	0.08 KB	53 KB	761 KB
50 000	908 KB	0.08 KB	18 KB	3.9 MB
100 000	1 318 KB	0.08 KB	168 KB	7.8 MB
500 000	2 948 KB	0.08 KB	376 KB	38.9 MB
1 000 000	4 167 KB	0.08 KB	545 KB	77.8 MB

Table 6.17: Size of each round and of the statement for different degree D for a 256-bit subgroup modulo a 1 526-bit prime.

$\lvert q\rvert$	160	160	160	256	256	256	384	384
$\lvert p\rvert$	1 248	1 536	3 248	1 536	2 432	3 248	3 248	7 984
10	11 KB	13 KB	26 KB	14 KB	20 KB	27 KB	28 KB	63 KB
100	33 KB	40 KB	81 KB	42 KB	63 KB	82 KB	85 KB	196 KB
1 000	113 KB	136 KB	277 KB	142 KB	216 KB	283 KB	291 KB	676 KB
5 000	255 KB	136 KB	625 KB	322 KB	488 KB	639 KB	658 KB	1.5 MB
10 000	360 KB	436 KB	884 KB	456 KB	690 KB	904 KB	931 KB	2.1 MB
50 000	812 KB	982 KB	1.9 MB	1.0 MB	1.5 MB	2.0 MB	2.0 MB	4.7 MB
100 000	1.1 MB	1.4 MB	2.8 MB	1.4 MB	2.2 MB	2.8 MB	2.9 MB	6.7 MB
500 000	2.5 MB	3.0 MB	6.2 MB	3.2 MB	4.8 MB	6.3 MB	6.5 MB	15.0 MB
1 000 000	3.6 MB	4.3 MB	8.7 MB	4.5 MB	6.9 MB	8.9 MB	9.2 MB	21.2 MB

Table 6.18: Size whole argument for different degree D for all choices of groups $\mathbb{G}$.

In general, if a group element is n times bigger than a field element we expect round 1 to be $\frac{9}{7}n \approx 1.3n$ times bigger than round 3. Thus, for normal choices of q and p the size of round 3 has more or less no influence on the complete argument size. We also see that in the argument size for $D \geq 5000$ is smaller than the statement, see Table 6.17. The communication between prover and verifier consists of group and filed elements, whereas the statement consists only of field elements; hence, it takes a while for the square root complexity to kick in, i.e. that the argument size becomes smaller than the statement.

In Table 6.18 we can find the results of the argument size for different degrees D and group parameters. We see that only for small D the argument size consists of a few kilo bytes, whereas for medium and big degree D the size grows quickly to a few megabytes. In most settings transferring data of a few megabytes quickly is not a problem; however, there might be applications which only have slow speed connection and in this setting Brands et al.'s polynomial evaluation argument is not feasible.

6.6 Comparison

6.6.1 Polynomial Evaluation Argument

Theoretical: The first approaches [CS97, Bra97] to prove for committed $u, v \in \mathbb{Z}_q$ that $p(u) = v$ for a given polynomial $p(X)$ with order D split in two parts: first they construct commitments $c_1, \ldots, c_D$ to values $u, u^2, u^3, \ldots, u^D$ and then they use the homomorphic property of the commitment scheme to get $p(u)$ as a linear combination of $u, u^2, \ldots, u^D$. This approach requires sending D commitments and the use of D multiplication arguments to show that the commitments $c_1, \ldots, c_D$ have been correctly constructed and indeed contain the correct powers of u. The cost can be reduced to $O(\sqrt{D})$ as suggested in Brands et al. [BDD07] by splitting the polynomial in $\sqrt{D}$ polynomials of degree $\sqrt{D}$ each.

Another approach is to arrange the polynomials in an arithmetic circuit. Cramer and Damgård [CD98] arithmetic circuit technique leads to linear cost, as they prove each gate separately. More sophisticated commitment scheme, as the general Pedersen commitment, allowed Groth [Gro09] to prove correctness of many gates at the same time and therefore to prove the correctness of the circuit with $O(\sqrt{D})$ communication cost.

Our protocol also has a two part structure, but we only need $\log D$ commitments $c_2, c_4, c_8, c_{16}, \ldots, c_D$ and $\log D$ multiplication arguments to prove they have been correctly formed and contain $u, u^2, u^4, u^8, u^{16}, \ldots, u^D$. By using a more sophisticated combination of these values in combination with the homomorphic properties of the commitment scheme, we then get the desired argument for $v = P(u)$. This reduces our communication to only $O(\log D)$ group elements, which is a huge improvement over the former work.

SHVZK argument	Rounds	Time $\mathcal{P}$ Expos	Time $\mathcal{V}$ Expos	Size Elements
[Bra97, CS97]	3	$O(D)$	$O(D)$	$O(D)\ \mathbb{G} + O(D)\ \mathbb{Z}_q$
[BDD07]	3	$O(\sqrt{D})$	$O(\sqrt{D})$	$O(\sqrt{D})\ \mathbb{G} + O(\sqrt{D})\ \mathbb{Z}_q$
[Gro09]	7	$O(D)$	$O(\sqrt{D})$	$\sqrt{D}\ \mathbb{G} + \sqrt{D}\ \mathbb{Z}_q$
This work	3	$O(\log D)$	$O(\log D)$	$O(\log D)\ \mathbb{G} + O(\log D)\ \mathbb{Z}_q$

Table 6.19: Comparison of our polynomial argument with former work

Table 6.19 states the asymptotic communication cost and computation cost of the polynomial arguments based on the discrete logarithm assumption. The polynomial evaluation argument from Brands et al. [BDD07] achieves the best complexity, so we will in the following paragraphs give a more detailed theoretical and practical comparison. In Table 6.20 we give a more detailed theoretical analysis that also counts the number of multiplications.

SHVZK argument	Rounds	Time $\mathcal{P}$ Expos	Time $\mathcal{P}$ Multip.	Time $\mathcal{V}$ Expos	Time $\mathcal{V}$ Multip.	Size Elements
[BDD07]	3	$16\sqrt{D}$	$18\sqrt{D}$	$D + 16\sqrt{D}$	$D + 10\sqrt{D}$	$8\sqrt{D}\ \mathbb{G} + 5\sqrt{D}\ \mathbb{Z}_q$
This work	3	$8 \log D$	$2D \log D$	$7 \log D$	$2D$	$4 \log D\ \mathbb{G} + 3 \log D \mathbb{Z}_q$

Table 6.20: Detailed comparison of our polynomial argument with Brands et al. [BDD07] argument

Brands et al.'s argument has square root communication complexity, but has to send $8\sqrt{D}\,\mathbb{G} + 5\sqrt{D}\,\mathbb{Z}_q$ elements in total. This yields a much bigger argument size than the $4\log D\,\mathbb{G} + 3\log D\mathbb{Z}_q$ elements which are sent by our protocol. So, we expect our argument to be smaller for small degree D and also expect that the increase of the size of the parameters enforces this effect.

Based on this table we would expect our verifier to run faster than Brands et al.'s as our asymptotic computation cost is much smaller, both verifiers have the same asymptotic number of multiplications but the number of exponentiations is much smaller for our verifier.

Just looking at the numbers of exponentiations needed by the prover can be a little deceptive though since in our polynomial evaluation argument we need $O(D \log D)$ multiplications in $\mathbb{Z}_q$ to compute the δ_j's and for large D this cost becomes prohibitive. Our performance gain for the prover is therefore largest in the range where D is large enough for $\log D$ to be significantly smaller than $\sqrt{D}$ yet not so large that the cost of $D \log D$ multiplications in $\mathbb{Z}_q$ becomes dominant.

For the run-time of the whole protocol however we expect that our argument performs faster in most cases, since the total asymptotic cost is smaller than Brands et el.'s computation cost. In the case where D is very big and the group size are small we assume that the influence of the $2D \log D$ multiplications is so big that our argument runs slower.

Practical: For the comparison for our polynomial evaluation argument we have used a 256-bit subgroup modulo a 1 536-bits p, and a 256-bit subgroup modulo a 3 248-bit prime, and a 384-bit subgroup modulo a 7 984-bit prime. This gives us enough data to compare the protocols for different levels of security.

We assume that the polynomial $P(X)$ is pre-computed and measured the run-time for polynomials with the various degrees D between 10 and 1 000 000. The result of our experiment can be found in Table 6.21, 6.22, and 6.23.

Elements in list D	Prover Brands et al.	This work	Verifier Brands et al.	This work	Communication Brands et al.	Communication This work
10	24 ms	13 ms	30 ms	17 ms	14 KB	8 KB
100	75 ms	24 ms	89 ms	30 ms	42 KB	15 KB
1 000	275 ms	41 ms	299 ms	45 ms	142 KB	21 KB
5 000	582 ms	106 ms	675 ms	67 ms	322 KB	27 KB
10 000	826 ms	182 ms	954 ms	81 ms	456 KB	29 KB
50 000	1 882 ms	714 ms	2 160 ms	143 ms	1.0 MB	33 KB
100 000	2 691 ms	1 420 ms	3 072 ms	217 ms	1.4 MB	35 KB
500 000	6 273 ms	7 393 ms	6 980 ms	689 ms	3.2 MB	40 KB
1 000 000	9 153 ms	15 573 ms	10 018 ms	1 313 ms	4.5 MB	42 KB

Table 6.21: Comparison of our polynomial argument with Brands et al. [BDD07]. All experiments used a 256-bit subgroup modulo a 1 536-bit prime.

As expected our communication cost compares very favorably against Brands et al.'s cost. For small q, p and small degree D our argument is only slightly smaller than Brands et al.'s argument. But this advantage grows for medium degree D and for big degrees D our communication is significantly smaller. For bigger q, p we have a much smaller communication cost for small degree D and increasing the size of the parameters enforce this effect.

As foreseen our verifier runs much faster than Brands et al.'s verifier for all combinations of degree sizes D and group parameters. For moderate size D it is also the case that our prover is more efficient. However, for $|q| = 256, |p| = 1\,536$ and for very large D the cost to calculate the δ_i's becomes dominant for our prover and here Brands et al.'s prover is faster. The experiments conducted for large security

Elements in list D	Prover Brands et al.	This work	Verifier Brands et al.	This work	Communication Brands et al.	Communication This work
10	81 ms	45 ms	110 ms	61 ms	27 KB	16 KB
100	254 ms	82 ms	323 ms	111 ms	82 KB	29 KB
1 000	875 ms	128 ms	1 091 ms	161 ms	283 KB	42 KB
5 000	1 978 ms	234 ms	2 455 ms	219 ms	693 KB	55 KB
10 000	2 800 ms	340 ms	3 470 ms	246 ms	904 KB	59 KB
50 000	6 329 ms	1 072 ms	7 816 ms	340 ms	2.0 MB	67 KB
100 000	8 995 ms	2 112 ms	11 090 ms	439 ms	2.8 MB	71 KB
500 000	20 375 ms	7 745 ms	24 909 ms	935 ms	6.3 MB	80 KB
1 000 000	29 088 ms	16 149 ms	35 393 ms	1 585 ms	8.9 KB	82 KB

Table 6.22: Comparison of our polynomial argument with Brands et al. [BDD07]. All experiments used a 256-bit subgroup modulo a 3 248-bit prime.

Elements in list D	Prover Brands et al.	This work	Verifier Brands et al.	This work	Communication Brands et al.	Communication This work
10	508 ms	289 ms	664 ms	401 ms	63 KB	37 KB
100	1 591 ms	506 ms	1 938 ms	702 ms	196 KB	65 KB
1 000	5 483 ms	734 ms	6 563 ms	1 012 ms	676 KB	87 KB
5 000	12 373 ms	1 021 ms	14 779 ms	1 336 ms	1.5 MB	127 KB
10 000	17 537 ms	1 181 ms	120 864 ms	1 451 ms	2.1 MB	137 KB
50 000	39 814 ms	1 976 ms	47 054 ms	1 736 ms	4.7 MB	156 KB
100 000	56 010 ms	2 978 ms	66 580 ms	1 936 ms	6.7 MB	166 KB
500 000	125 775 ms	10 644 ms	149 159 ms	2 757 ms	15.0 MB	186 KB
1 000 000	177 855 ms	20 829 ms	210 890 ms	3 689 ms	21.1 MB	196 KB

Table 6.23: Comparison of our polynomial argument with Brands et al. [BDD07]. All experiments used a 384-bit subgroup modulo a 7 984-bit prime.

parameters show that this effect occurs as well in these cases but for even bigger D. Thus, our argument is faster for all reasonable degree sizes.

Our experiments also show that the total run-time for our protocol is faster than the combined one for Brands et al.'s prover and verifier. The only exception for this could be a combination of small q, p and very large D.

In conclusion we can say that our argument has a clear performance advantage over Brands et al.'s protocol and as a result an even bigger advantage over all former arguments which have linear complexity.

6.6.2 Multivariate Polynomial Argument

To prove for secret $u_1, u_2, \ldots, u_N, v$ that

$$P(u_1, \ldots, u_N) = v$$

for polynomial $P(X_1, \ldots, X_N) \in \mathbb{Z}_q[X_1, \ldots, X_N]$ [Bra97, CS97] committed first to all possible values $u_1^{i_1} \ldots u_N^{i_N}, i_1, \ldots i_N = 1, \ldots, D$. Next, they used the homomorphic property of the commitment scheme to show that the correct linear combination of these values is equal to the commitment of v. This approach is very expensive as the prover has to commit themselves to $O(D^N)$ values. Applying general circuit techniques [CD98, Gro09] in the discrete logarithmic setting suffer of the same bottleneck, the prover has to commit themselves to all the coefficients.

In Table 6.24 we compare the communication cost and computation cost of these former approaches and our solution from Section 6.3. Based on the table and the practical results for $N = 1$ we expect our communication cost to be much smaller than the communication cost of the former work. This advantage should get bigger the bigger N and D gets.

All former protocols have a computation cost of $O(D^N)$ for both parties, whereas our prover and verifier have only to calculate $O((\log D)^N)$ exponentiations. We expect therefore our protocol to be much lighter and faster than the former one. However, our prover has to calculate a huge number of multiplications on top of the exponentiations and the same is true for our verifier. These multiplications become dominant for big D and slow our protocol down.

The data of the single variate argument indicates that for big group parameters, the cost of $O(\sqrt{D})$ exponentiations is still higher than the cost of the dominant multiplication. In this case we compare our parties against parties with linear computation cost; thus, we expect our parties still to be faster and lighter than the former one.

SHVZK argument	Rounds	Time $\mathcal{P}$ Expos	Time $\mathcal{V}$ Expos	Size Elements
[Bra97, CS97]	3	$O(D^N)$	$O(D^N)$	$O(D^N)\ \mathbb{G} + O(D^N)\ \mathbb{Z}_q$
[CD98]	3	$O(D^N)$	$O(D^N)$	$O(D^N)\ \mathbb{G} + O(D^N)\ \mathbb{Z}_q$
[Gro09]	7	$O(D^N)$	$O(D^{N/2})$	$O(D^{N/2})\ \mathbb{G} + O(D^{N/2})\ \mathbb{Z}_q$
This work	3	$O((\log D)^N)$	$O((\log D)^N)$	$O((\log D)^N)\ \mathbb{G} + O((\log D)^N)\ \mathbb{Z}_q$

Table 6.24: Comparison of our multivariate polynomial argument with former work

6.6.3 Batch Polynomial Argument

To prove evaluation of L polynomials at the same time requires for all former techniques [Bra97, CS97, BDD07, CD98, Gro09] to repeat a single argument L times. This requires L times the original cost. Our Batch polynomial argument from Section 6.4 can do better, a detailed comparison can be found in Table 6.25.

SHVZK argument	Rounds	Time $\mathcal{P}$ Expos	Time $\mathcal{V}$ Expos	Size Elements
[Bra97, CS97, CD98]	3	$O(LD)$	$O(LD)$	$O(LD)\ \mathbb{G} + O(LD)\ \mathbb{Z}_q$
[Gro09]	7	$O(LD)$	$O(LD)$	$O(L\sqrt{D})\ \mathbb{G} + O(L\sqrt{D})\ \mathbb{Z}_q$
[BDD07]	3	$O(L\sqrt{D})$	$O(L\sqrt{D})$	$O(L\sqrt{D})\ \mathbb{G} + O(L\sqrt{D})\ \mathbb{Z}_q$
This work	3	$O(L \log D)$	$O(\sqrt{L} \log D))$	$O(\sqrt{L} \log D)\ \mathbb{G} + O(\sqrt{L} \log D)\ \mathbb{Z}_q$

Table 6.25: Comparison of our batch polynomial argument with former work

Brands et al.'s [BDD07] argument has for $L = 1$ the smallest communication cost so far, the data in Section 6.6.1 shows that our communication cost in this case is much lighter. For $L > 1$ this cost grows linear in Brands et al.'s case whereas our cost grows only logarithmic in L. For this reason we expect that the communication cost of our batch polynomial is much smaller than the cost of all former protocols.

A similar argument holds for the verifier. In the case $L = 1$ our verifier is much faster than Brands et al.'s verifier and again our cost grows less for $L > 1$ than Brands et al.'s cost. Thus, our verifier seems to be faster than all former ones.

The cost of our prover grows linearly in L, similar to all other protocols. However, in the case $L = 1$ our prover runs faster than Brands et al.'s prover for nearly all cases and therefore we expect the same for the cases $L > 1$.

In general we can say that our new batch polynomial argument leads to a much lighter and faster arguments compared to all former work.

Chapter 7

Zero-Knowledge Non-Membership And Membership Arguments

The theoretical part of Section 7.2 and some of the practical result from the same section, most parts of Section 7.4 and 7.6 are published at Eurocrypt 2013[BG13] together with Jens Groth. Section 7.5 summarize Brands et al.'s [BDD07] blacklist argument, and also gives a discussion on practical results based on our implementation.

7.1 Introduction

Given a public list $\mathcal{L} = \{\lambda_1, \ldots, \lambda_D\}$ and a committed u, we want to prove that u is not contained in $\mathcal{L}$ such non-membership proofs have many useful applications. As a concrete application of our polynomial evaluation argument we will look at blacklisting anonymous users, which is a notoriously difficult cryptographic problem. Anonymising networks such as Tor [Tor13] allow a user to hide their IP address and are used by a number of groups including undercover police agents, abuse victims and citizens living under dictatorships. During the Arab Spring, for instance, the Tor network experienced a spike in users from Libya and Egypt [Din11]. However, anonymous access to services can also lead to abuse. Wikipedia, for instance, allows anonymous postings, but blocks the IP address of misbehaving users. This crude solution means that if one user of Tor abuses Wikipedia, all users whose traffic comes out of the same Tor relay with this IP-address are blocked. So one misbehaving user causes many innocent users to be punished. Johnson et al. [JKTS07] suggested the Nymble system to deal with this problem. In this alternative solution IP-addresses are not blocked but instead each user anonymously proves that they has not been blacklisted. Using the polynomial evaluation argument we construct a non-membership proof, which enables a user to efficiently prove that they have not been blacklisted.

One downside of Nymble-like systems is, that a user has to obtain a personal token. Depending on the construction the user has to remember the token and log in every time, or the token is only valid with one copy of a browser. Both possibilities discourage people from using Nymble-like systems, as they want to have everything as easy as possible. Unfortunately, using our non-membership proof in real life would result in similar problems, a user still needs a personal token.

Another application of non-membership proofs is to show that a member of a group signature

scheme is not revoked. Depending on the setup of the group signature scheme a user could also show that they has the right to sign using a membership proof.

We can use our polynomial evaluation argument to construct efficient membership proofs. Membership proofs are useful when operating a whitelist access control system, or in applications such as e-voting or e-auctions where users want to prove that their votes are valid or their bids belong to a set of approved values.

We construct zero-knowledge arguments for membership and non-membership with logarithmic communication complexity, which are based on our polynomial evaluation argument, section 6.2. We have strived to keep the arguments as simple as possible: they are a 3-move public coin arguments, the setup consists of just a few group elements in a prime-order group, and security relies on the discrete logarithm assumption.

7.1.1 Techniques

Given a Pedersen commitment c_u and a public list of values $\mathcal{L} = \{\lambda_1, \ldots, \lambda_D\}$ we give a zero-knowledge argument of knowledge that c_u is a commitment to $u \in \mathcal{L}$ or $u \notin \mathcal{L}$. Following Brands et al.'s [BDD07] approach we compute the polynomial

$$P(X) = \prod_{i=1}^{D}(X - \lambda_i)$$

and commit to $P(u) = v$. In the case of non-membership $u \notin \mathcal{L}$ and $P(u) \neq 0$; thus, we will show that the commitment to v does not contain 0. In the reversed case $u \in \mathcal{L}$ we know $P(u) = 0$ and we will show that the commitment of v does contain $v = 0$.

The communication complexity of our basic non-membership proof is $O(\log D)$ group elements. This is a significant reduction compared to previous schemes with complexity $O(D)$ or $O(\sqrt{D})$. The prover's computation is a logarithmic number of exponentiations and a quasi-linear number of multiplications in $\mathbb{Z}_q$. The verifier's computation is a logarithmic number of exponentiations and a linear number of multiplications in $\mathbb{Z}_q$.

The asymptotic complexity of our membership argument is the same as for the non-membership protocol. Thus, the argument consists of $O(\log D)$ group elements, and both verifier and prover have to compute $O(\log D)$ exponentiations.

We also consider the case from the single sign-on system of Brands et al. [BDD07] where we have commitments to L values and have L sets for which we want to prove non-membership. Their solution had a communication cost of $O(L\sqrt{D})$, which we are able to reduce to $O(\sqrt{L}\log D)$.

We have implemented the zero-knowledge argument for non-membership to demonstrate the practicality of our protocol, we also coded Brand et al.'s blacklist argument. Performance results indicate that our argument size is much smaller than previous arguments. Furthermore, our verifier's computation is considerably faster. Our prover is faster for small to big blacklists, but for very big blacklists the prover by Brands et al. is faster.

7.1.2 Former Work

Proofs for set membership and non-membership for a committed $u \in \mathcal{L}$ or $u \notin \mathcal{L}$ where $\mathcal{L} = \{\lambda_1, \ldots, \lambda_D\}$ have been studied before. The most straightforward approach is to prove in a single case by single case manner the conjunction $\lambda_1 \neq u \wedge \ldots \wedge \lambda_D \neq u$ in the case of non-membership or the disjunction $\lambda_1 = u \vee \ldots \vee \lambda_D = u$ in the case of membership. In the context of revoking members of group signature schemes Bresson and Stern [BS01] proposed such a solution based on the strong RSA assumption. Peng and Bao [PB10] gave a general discrete logarithm based zero-knowledge arguments of non-membership with linear complexity. Brands et al. [BDD07] improved the communication complexity to $O(\sqrt{D})$ group elements and later Peng [Pen11] gave a solution for a non-membership proof with the same complexity using techniques similar to Brands et al. [BDD07].

Accumulators [BdM93, BP97, CL02, Ngu05, CKS09, TX03, WWP07] provide another mechanism for giving membership proofs. An accumulator is a succinct aggregate of a set of values where it is possible to issue membership proofs for each accumulated value. A party in possession of such a membership proof, typically a few group elements, can then demonstrate that the value is included in the set. Non-membership accumulators have been proposed as well [LLX07, YYC$^+$12]. Accumulators, however, rely on a trusted party to maintain the accumulator and if corrupt this trusted party can issue arbitrary membership proofs. Furthermore, accumulators rely on cryptographic assumptions that have been studied less than the discrete logarithm problem, for instance the strong RSA assumption or the q-SDH assumption in a pairing-based setting. These assumptions also mean that group elements have to be large and once this has been factored in the accumulator-based solutions end up having larger communication cost than our membership and non-membership proofs for groups over elliptic or hyper-elliptic curves. The construction of Camacho et al. [CHKO12] does not rely on a trusted party and only assumes the existence of hash functions; however, witnesses in their setting depend logarithmically on the number of accumulated elements.

In Song's non-membership proof [Son01] the prover publishes a constant number of elements and the verifier checks these elements against a blacklist by carrying out a few operations for each blacklist element, several systems along these lines have been proposed [AST02, BS04, NF05, NF06, ZL06]. The operations consist either of exponentiations or pairings, so this scheme places a heavy computational burden on the verifier.

Camenisch et al. [CCS08] gave a membership proof where the set elements are signed by a trusted third party. Now membership can be proven with a constant number of group elements by demonstrating that the value has been signed. Related ideas have recently been used by Libert et al. [LPY12] in the context of revoking group signatures, where a trusted third party signs representatives of sets that cover the whitelist of non-revoked users and the user gives a zero-knowledge proof of belonging to this set [NNL01].

All these solutions suffer from similar drawbacks that accumulator-based solutions have though. They require trust in a third party to be honest when blacklisting members or signing messages, and to get efficient proofs the signatures are built from strong assumptions such as the strong RSA assumption

or pairing-based assumptions.

A different approach is taken by Nymble-like systems [JKTS07, TKCS11, HHG10, HG11, LH11], which also rely on a trusted third party. The user obtains a pseudonym, a "nymble", from the trusted third party which is only valid for a certain time frame with one server. The blacklist consists of nymbles by misbehaving users and in [JKTS07, TKCS11, HHG10] the server simply checks if the nymble of a connecting user is contained in the blacklist. To weaken the trust in the trusted third party Lofgren and Hopper [LH11] use accumulators together with the Nymble setup, while Henry and Goldberg [HG11] rely on the techniques of Brands et al. [BDD07] for the user to give a zero-knowledge argument for the non-membership of the blacklist.

7.2 Non-Membership Argument Based on our Polynomial Evaluation Argument

We will now describe a non-membership argument with logarithmic communication complexity for a committed value not belonging to a set $\mathcal{L} = \{\lambda_1, \ldots, \lambda_D\}$.

Similar to Brands et al.'s approach we define a polynomial $P(X) = \prod_{i=1}^{D}(X - \lambda_i)$ with the elements in the set as roots. With this choice of polynomial we have $u \in \mathcal{L}$ if and only if $P(u) = 0$. The prover has a commitment c_u and will demonstrate that the committed value u does not belong to $\mathcal{L}$ by showing $P(u) \neq 0$.

The prover computes $v = P(u)$, makes a commitment to v and can now give a SHVZK argument for $v = P(u)$ using the techniques from Section 6.2. To prove non-membership they just needs to prove $v \neq 0$. To do this the prover commits to $w = v^{-1}$ and uses an inverse argument as described in Section 5.1 to show $vw = 1$, which will convince the verifier that $v \neq 0$. The main cost in this argument is the polynomial evaluation argument, the inverse argument only costs a couple of group elements.

Statement: $\{\mathbb{G}, p, q\},\ ck,\ \mathcal{L} = \{\lambda_1, \ldots, \lambda_D\} \subset \mathbb{Z}_q$, $P(X) = \prod_{i=1}^{D}(X - \lambda_i)$, and $c_u \in \mathbb{G}$

Prover's witness: $u, r \in \mathbb{Z}_q$ such that $c_u = \mathrm{com}_{ck}(u; r)$ and $u \notin \mathcal{L}$

Argument: Compute

1. $v = P(u)$
2. $c_v = \mathrm{com}_{ck}(v; s)$ where $s \leftarrow \mathbb{Z}_q$
3. Engage in parallel in a SHVZK inverse argument described in Section 5.2 to show $v \neq 0$
4. In parallel engage in the SHVZK polynomial evaluation argument from Section 6.2 to show $v = P(u)$.

Send c_v

Verification: The verifier accepts $u \notin \mathcal{L}$ if and only if

1. $c_v, \in \mathbb{G}$

2. The two SHVZK arguments are valid.

Theorem 29. *The protocol above is a public coin perfect generalized $\Sigma-$protocol of $u \in \mathbb{Z}_q$ such that $u \notin \mathcal{L}$.*

Proof. Perfect completeness follows from the perfect completeness of the two SHVZK arguments.

The SHVZK simulator picks $c_v \leftarrow \mathbb{G}$ at random and runs the SHVZK simulators for the two underlying SHVZK arguments. Since the commitment scheme is perfectly hiding and the underlying SHVZK arguments are perfect SHVZK this gives us perfect SHVZK.

The protocol has perfect generalized special soundness. The extractor E runs the extractors for the two underlying SHVZK arguments to get openings u, v satisfying $v \neq 0$ and $P(u) = v$. The second condition tells us $P(u) \neq 0$, so u is a not a root of the polynomial and therefore not in the list. □

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 172 \rangle \subset \mathbb{Z}^*_{179}$, which has prime order $q = 89$. The statement consists of a blacklist $\mathcal{L} = \{9, 26, 49, 48\}$, the corresponding polynomial

$$P(X) = X^4 + 46X^3 + 18X^2 + 2X + 81,$$

$c_{u_0} = 142$ and $\{\mathbb{G}, p, q\} = \{\langle 172 \rangle, 179, 89\},\ ck = \{74, 172\}$.

The prover knows witnesses $u = 84, r_0 = 15$, such that $c_{u_0} = \mathrm{com}_{ck}(u; r_0)$ and $u \notin \mathcal{L}$ and wants to convince the verifier that u is not included in the blacklist. The prover first calculates $v = P(u) = 24 \in \mathbb{Z}_{89}$, picks random $s = 34$, and calculate the commitment to v as

$$c_v = \mathrm{com}_{ck}(v; s) = 85.$$

The prover then engages in a integer inverse argument to show $v \neq 0$ and picking $s_i = 88, t_i = 26$ they compute

$$c_b = 83.$$

Both parties also engage in a polynomial argument. For this argument the prover commits themselves to u^2, u^4 in

$$c_{u_2} = \mathrm{com}_{ck}(u^2; r_1) = 17 \qquad c_{u_2} = \mathrm{com}_{ck}(u^4; r_2) = 46$$

where $r_1 = 80, r_2 = 1$. Next, the prover picks $f_0 = 43, f_1 = 25, f_2 = 44, s_0 = 16, s_1 = 52, s_2 = 47$ and calculates

$$c_{f_0} = 139 \qquad c_{f_1} = 142 \qquad c_{f_2} = 82.$$

Then, the prover calculates the δ_i's for $i = 0, 1, 2$ as described in Section 6.2, which gives

$$\delta_0 = 0, \delta_1 = 55, \delta_2 = 47.$$

The prover picks $t_0 = 57, t_1 = 77, t_2 = 26$ and compute the commitment to the δ_i's:

$$c_{\delta_0} = 64 \qquad c_{\delta_1} = 36 \qquad c_{\delta_2} = 100.$$

Finally the prover picks $\xi_0 = 26, \xi_1 = 64$ and calculates $fu_0 = 52, fu_1 = 2$, and

$$c_{fu_0} = 110 \qquad c_{fu_1} = 117,$$

and sends all the commitments

$$c_v = \mathrm{com}_{ck}(v; s) = 85 \quad c_d = 83 \quad c_e = 36 \quad c_f = 29$$

$$c_{\delta_0} = 64 \qquad c_{\delta_1} = 36 \qquad c_{\delta_2} = 100$$

$$c_{f_0} = 139 \quad c_{f_1} = 142 \quad c_{f_2} = 82 \quad c_{fu_0} = 110 \quad c_{fu_1} = 117$$

to the verifier.

The verifier challenges the prover with $x = 81$; so the prover calculates the answers,

$$\overline{a} = 46 \qquad \overline{r} = 30$$

$$\overline{f}_0 = 83 \qquad \overline{f}_1 = 3 \qquad \overline{f}_2 = 28 \qquad \overline{r}_0 = 74 \qquad \overline{r}_1 = 35 \qquad \overline{r}_2 = 39$$

$$\overline{t} = 73 \qquad \overline{\xi}_0 = 10 \qquad \overline{\xi}_1 = 83$$

and sends them back.

The verifier checks now if the commitments are in $\mathbb{G}$ and therefore valid, and also if the answers are all in $\mathbb{Z}_q$. Then the verifier checks if the inverse argument is valid, that means

$$c_b = 83 = c_a^{\overline{a}} \mathrm{com}_{ck}(x; \overline{r}) \quad (\checkmark)$$

To check the underlying polynomial argument, the verifier first calculates $\overline{\delta} = 70$ using a binary tree and tests

$$c_{u_0}^x c_{f_0} = 135 = \mathrm{com}_{ck}(\overline{f}_0; \overline{r}_0) \quad (\checkmark) \qquad c_{u_1}^x c_{f_1} = 82 = \mathrm{com}_{ck}(\overline{f}_1; \overline{r}_1) \quad (\checkmark)$$

$$c_{u_2}^x c_{f_2} = 46 = \mathrm{com}_{ck}(\overline{f}_2; \overline{r}_2) \quad (\checkmark)$$

$$c_{u_1}^x c_0^{-\overline{f}_0} c_{fu_0} = 81 = \mathrm{com}_{ck}(0; \overline{\xi}_0) \quad (\checkmark) \qquad c_{u_2}^x c_1^{-\overline{f}_1} c_{fu_1} = 86 = \mathrm{com}_{ck}(0; \overline{\xi}_1) \quad (\checkmark)$$

$$c_v^{x^3} c_{\delta_2}^{x^2} c_{\delta_1}^x c_{\delta_0} = 46 = \mathrm{com}_{ck}(\overline{\delta}; \overline{t}) \quad (\checkmark).$$

7.2.1 Implementation and Practical Results

We implemented the protocol to test the real life performance and to obtain some experimental results. We chose the same groups as in the case of out polynomial argument, that means a 160-bit subgroup modulo a 1 248-bit, a 1 536-bit, and a 3 248-bit prime, and subgroups with order $|p| = 256$ modulo a 1 536-bit prime, a 2 432-bit prime, and a 3 248-bit prime. The groups have different levels of security, the exact values can be found in Section 4.1.1, and help to analyze the influence of different parameters, for example group size.

The non-membership proof is a direct application of our polynomial argument with the same asymptotic cost and we expect a similar running time for this reason. To validate this assumption, we implemented two different versions. The first one is a conservative un-optimized version and the second one is optimized using the sliding window algorithm.

Table 7.1 states the run-time of the un-optimized version and the optimized version for a 256-bit prime modulo a 1 536-bit prime. We see that the influence of the optimization for the verifier is very small. The reason for this is that the cost of the multiplication is dominant over the very small number of exponentiations. For the prover the influence of the optimization is bigger for small D but this effect is canceled out for bigger D, as the cost of the multiplications becomes dominant.

Next, we have to analyze what happens if the order stays fixed and the moduli value increase. Table 7.2 and 7.3 give the results for the optimized version for fixed order with 160-bit and 256-bit. We see that the run-time increases slightly for bigger moduli values, but this is negligible compared to the increase of the moduli. The reason for this is that the run-time is dominated by the multiplications and this cost only depends on the subgroup size. This result means that we can increase the security of our protocol slightly by increasing the modulus value of the underlying group without compromising performance.

Table 7.4 and 7.5 state the result of the reverse case of fixed group size of 1 536-bit and 3 248-bit. Again the run-time of the protocol gets higher for bigger subgroup size, but this increase is very small, for bigger D the ratio between the different results is around one. Taking also into account that for small D the run-time is only a few milliseconds independent of the group size, we can increase the subgroup

	Prover			Verifier		
D	Conservative	Optimized	Ratio	Conservative	Optimized	Ratio
10	29 ms	18 ms	0.62	24 ms	20 ms	0.85
100	46 ms	29 ms	0.63	39 ms	34 ms	0.86
1 000	70 ms	47 ms	0.68	56 ms	49 ms	0 .88
5 000	140 ms	109 ms	0 .78	77 ms	68 ms	0 .89
10 000	220 ms	189 ms	0 .86	90 ms	83 ms	0 .92
50 000	776 ms	754 ms	0 .97	152 ms	148 ms	0 .97
100 000	1 538 ms	1 495 ms	0 .97	228 ms	216 ms	0 .95
500 000	7 396 ms	7 369 ms	1.00	702 ms	693 ms	0 .99
1 000 000	20 423 ms	15 384 ms	1.00	1 469 ms	1 310 ms	0 .99

Table 7.1: Run-time in ms of the blacklist argument on a group $\mathbb{G}$ with 256-bit order modulo a 1 536-bit prime for degree D between 10 and 1 000 000 for the conservative and the optimized version and the ratio between the two versions.

$\|q\| = 160$	$\|p\| = 1\,248$		$\|p\| = 1\,536$	
D	Prover	Verifier	Prover	Verifier
10	9 ms	10 ms	12 ms	13 ms
100	15 ms	16 ms	9 ms	22 ms
1 000	26 ms	23 ms	31 ms	31 ms
5 000	80 ms	35 ms	91 ms	47 ms
10 000	150 ms	44 ms	159 ms	56 ms
50 000	670 ms	97 ms	678 ms	109 ms
100 000	1 382 ms	160 ms	1 391 ms	174 ms
500 000	6 776 ms	562 ms	6 900 ms	581 ms
1 000 000	14 370 ms	1 095 ms	14 321 ms	1 180 ms

Table 7.2: Comparison of the blacklist argument for different degree D for optimized version on different groups with fixed subgroup size of 160-bit.

$\|q\| = 256$	$\|p\| = 1\,536$		$\|p\| = 2\,432$		$\|p\| = 3\,248$	
	Prover	Verifier	Prover	Verifier	Prover	Verifier
10	18 ms	20 ms	39 ms	48 ms	63 ms	76 ms
100	29 ms	34 ms	62 ms	79 ms	99 ms	125 ms
1 000	47 ms	49 ms	91 ms	113 ms	144 ms	176 ms
5 000	109 ms	68 ms	164 ms	161 ms	253 ms	235 ms
10 000	189 ms	83 ms	240 ms	194 ms	361 ms	261 ms
50 000	754 ms	148 ms	773 ms	411 ms	1 114 ms	358 ms
100 000	1 495 ms	216 ms	1 500 ms	689 ms	2 130 ms	455 ms
500 000	7 369 ms	693 ms	7 445 ms	893 ms	7 827 ms	1 019 ms
1 000 000	15 384 ms	1 310 ms	15 524 ms	1 404 ms	16 137 ms	1 572 ms

Table 7.3: Comparison of the blacklist argument for different degree D for optimized version on different groups with fixed subgroup size of 256-bit.

$\|p\| = 1\,536$	$\|q\| = 160$		$\|q\| = 256$	
D	Prover	Verifier	Prover	Verifier
10	12 ms	13 ms	18 ms	20 ms
100	19 ms	22 ms	29 ms	34 ms
1 000	31 ms	31 ms	47 ms	49 ms
5 000	91 ms	47 ms	109 ms	68 ms
10 000	159 ms	56 ms	189 ms	83 ms
50 000	678 ms	109 ms	754 ms	148 ms
100 000	1 391 ms	174 ms	1 495 ms	216 ms
500 000	6 900 ms	581 ms	7 369 ms	693 ms
1 000 000	14 321 ms	1 180 ms	15 384 ms	1 310 ms

Table 7.4: Comparison of the blacklist argument for different degree D for optimized version on different subgroups with moduli of 1 536-bit.

size and therefore the security level without compromising the performance.

All these results confirm our assumption that the protocol behaves like the polynomial evaluation argument, section 6.2, and we can therefore say that this argument is practical for all levels of security beside the case that D gets very big.

Table 7.6 gives the size of the complete argument for different groups $\mathbb{G}$ and different blacklist sizes D. We see that in all cases the argument size consists only of a few kilobytes, independent of the

$\|p\| = 3\,248$	$\|q\| = 160$		$\|q\| = 256$		$\|q\| = 384$	
D	Prover	Verifier	Prover	Verifier	Prover	Verifier
10	41 ms	48 ms	63 ms	76 ms	92 ms	111 ms
100	63 ms	79 ms	99 ms	125 ms	148 ms	187 ms
1 000	94 ms	114 ms	144 ms	176 ms	222 ms	276 ms
5 000	169 ms	153 ms	253 ms	235 ms	344 ms	367 ms
10 000	246 ms	171 ms	361 ms	261 ms	447 ms	394 ms
50 000	776 ms	238 ms	1 114 ms	358 ms	1 186 ms	521 ms
100 000	1 495 ms	305 ms	2 130 ms	455 ms	2 144 ms	640 ms
500 000	6 900 ms	736 ms	7 827 ms	1 019 ms	9 845 ms	1 383 ms
1 000 000	14 402 ms	1 268 ms	16 137 ms	1 572 ms	20 257 ms	2 171 ms

Table 7.5: Comparison of the blacklist argument for different degree D for optimized version on different subgroups with moduli of 3 248-bit.

$\|q\|$	160	160	160	256	256	256	384	384
$\|p\|$	1 248	1 536	3 248	1 536	2 432	3 248	3 248	7 984
10	8 KB	10 KB	20 KB	11 KB	16 KB	21 KB	21 KB	50 KB
100	13 KB	16 KB	33 KB	17 KB	25 KB	33 KB	34KB	80 KB
1 000	18 KB	22 KB	45 KB	23 KB	35 KB	46 KB	47 KB	109 KB
5 000	23 KB	29 KB	56 KB	28 KB	44 KB	58KB	60 KB	139 KB
10 000	25 KB	30 KB	59 KB	31 KB	48 KB	61 KB	64 KB	149 KB
50 000	28 KB	34 KB	69 KB	36 KB	54 KB	71KB	73 KB	169 KB
100 000	29 KB	36 KB	73 KB	38 KB	57 KB	75 KB	77 KB	178 KB
500 000	33 KB	40 KB	81 KB	42 KB	63 KB	83 KB	85KB	198 KB
1 000 000	34 KB	41 KB	84 KB	44 KB	65 KB	85 KB	88 KB	302 6KB

Table 7.6: Size of whole blacklist argument for different degree D for all choices of groups $\mathbb{G}$.

group size or blacklist size. This means that our new argument can be used in application with low speed connection or in settings where many users try at the same time to access a server and the server has to verify many protocols at the same time, which places a high burden on the system.

7.3 Non-Membership Argument to a Secret List

In this section we will describe how to show that a committed value u does not belong to a committed set $\mathcal{L} = \{\lambda_1, \ldots, \lambda_D\}$.

Similar to the non-membership argument in Section refnmp we define a polynomial

$$P(X) = \prod_{i=1}^{D}(X - \lambda_i) = \sum_{i=0}^{D} a_i X$$

with all $\lambda_i \in \mathcal{L}$ as roots. With this choice of polynomial we have $u \in \mathcal{L}$ if and only if $P(u) = 0$. The prover has a commitment c_u and will demonstrate that the committed value u does not belong to $\mathcal{L}$ by showing $P(u) \neq 0$.

Since the verifier does not know the list $\mathcal{L}$, he cannot know the polynomial. Therefore, the first step of the prover is convincing the verifier form the correctness of the polynomial, i.e. that is has the λ_is as root.To do that the prover has to commit themselves to the coefficients a_i of the polynomial $P(X)$ and demonstrate that for random challenge x it holds $P(x) = \prod_{i=1}^{D}(\lambda_i - x)$. This can be done using

a single value product argument, Section 5.3.4. Thus, both parties calculate $c_{b_i} = c_{\lambda_i}\mathrm{com}_{ck}(-x, 0)$ for $1 \leq i \leq D$ and $c_b = \prod_{i=1}^{D} c_{a_i}^{x^i}$ themselves and in the argument to show

$$\prod_{i=1}^{D}(\lambda_i - x) = \sum_{i=0}^{D} a_i X.$$

The prover computes $v = P(u)$, makes a commitment to v and can now give a SHVZK argument for $v = P(u)$ using the techniques similar to Section 6.2. The only difference is that the verifier cannot calculate

$$\overline{\delta} = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j}$$

themselves, as he do not know the coefficients $a_i = a_{i_0\ldots i_d}$ of the polynomial $P(X)$.

More precisely, by padding with zero-coefficients we can without loss of generality assume $D = 2^{d+1} - 1$. It is useful to write i in binary, i.e. $i = i_0 \ldots i_d$ where $i_j \in \{0, 1\}$. We can then rewrite the term X^i as

$$X^i = X^{\sum_{j=0}^{d} i_j 2^j} = \prod_{j=0}^{d} \left(X^{2^j}\right)^{i_j}.$$

Substituting this in the polynomial we get

$$P(X) = \sum_{i=0}^{D} a_i X^i = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} \left(X^{2^j}\right)^{i_j}.$$

The prover will calculate commitments to $u^2, u^4, \ldots, u^{2^d}$ and prove that when inserted into the rewritten polynomial we have

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} \left(u^{2^j}\right)^{i_j} = v.$$

Since $d = \lfloor \log D \rfloor$ the prover only makes a logarithmic number of commitments, which will help to keep the communication cost low. Standard techniques can be used to give arguments of knowledge that the commitments $c_{u_1}, \ldots, c_{u_d}$ to $u^{2^1}, \ldots, u^{2^d}$ are well-formed and indeed contain the correct powers of u.

To show the committed powers of u in $c_0, c_1, \ldots, c_d$ evaluate to the committed v the prover picks random values $f_0, \ldots, f_d \leftarrow \mathbb{Z}_p$ and defines a new polynomial

$$Q(X) = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j} = X^{d+1}P(u) + X^d\delta_d + \ldots + X\delta_1 + \delta_0.$$

The idea behind this choice of $Q(X)$ is that for each i_j either an Xu^{2^j} factor is included or an X factor is included, hence $P(u)$ is the coefficient of X^{d+1}. Each f_j on the other hand is not multiplied by X and will therefore only affect the lower degree coefficients $\delta_0, \ldots, \delta_d$ of $Q(X)$.

The prover will now demonstrate that the coefficient of X^{d+1} in the secret $Q(X)$ is the same as v in a way that cancels out the $\delta_0, \ldots, \delta_d$ coefficients. The prover sends the verifier commitments $c_{f_0}, \ldots, c_{f_d}$

to $f_0, \ldots, f_d$ and commitments $c_{\delta_0}, \ldots, c_{\delta_d}$ to $\delta_0, \ldots, \delta_d$. Afterwards, the verifier will pick a random challenge $x \leftarrow \mathbb{Z}_q^*$. The prover will now open suitable products of the commitments in a way such that the verifier can check that the committed values u, v satisfy

$$Q(x) = x^{d+1}v + x^d\delta_d + \ldots + \delta_0.$$

More precisely, after receiving the challenge x the prover opens each product $c_j^x c_{f_j}$ to $\overline{f_j} = xu^{2^j} + f_j$.

Lastly, the prover opens

$$c_v^{x^{d+1}} \prod_{j=0}^{d} c_{\delta_j}^{x^j} \quad \text{to} \quad \boldsymbol{c_a^f},$$

to show that $P(u) = v$. The vector $\boldsymbol{f}$ is definded as

$$\boldsymbol{f} = (x^{d+1}, \overline{f}_0 x^d, \overline{f}_1 x^d, \ldots, \prod_{j=1}^{d} \overline{f}_j)$$

with the i-th entry in the vector equals $\prod_{j=1}^{d} \overline{f}_j^{\,i_j} x^{1-i_j}$ for $i = i_0 \ldots i_d$

Note that the verifier can calculate $\boldsymbol{f}$ by themselves and therefore only accepts the opening if

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j} = x^{d+1}v + x^d\delta_d + \ldots + x\delta_1 + \delta_0.$$

This has negligible probability of being true unless $P(u) = v$.

Finally, to prove non-membership the prover needs to demonstrate $v \neq 0$. This is done using a inverse argument as described in Section 5.2.

Statement: $\{\mathbb{G}, p, q\},\ ck,\ c_u \in \mathbb{G}$ and $\boldsymbol{c}_\lambda \in \mathbb{G}^D$

Prover's witness: $\mathcal{L} = \{\lambda_1, \ldots, \lambda_D\} \subset \mathbb{Z}_q \quad \boldsymbol{s} \in \mathbb{Z}_q^D$ such that $\boldsymbol{c}_\lambda = \mathrm{com}_{ck}(L; \boldsymbol{s}), \quad P(X) = \prod_{i=1}^{D}(X - \lambda_i)$ and $u, r \in \mathbb{Z}_q$ such that $c_u = \mathrm{com}_{ck}(u; r)$ and $u \notin \mathcal{L}$

Initial message: Compute

1. $\boldsymbol{c}_a = \mathrm{com}_{ck}(\boldsymbol{a}; \boldsymbol{\tau})$ where $\boldsymbol{\tau} \leftarrow \mathbb{Z}_q^{D+1}$
2. $v = P(u)$
3. $c_v = \mathrm{com}_{ck}(v; s)$ where $s \leftarrow \mathbb{Z}_q$
4. $c_{u_1} = \mathrm{com}_{ck}(u^{2^1}; r_1) \quad \ldots \quad c_{u_d} = \mathrm{com}_{ck}(u^{2^d}; r_d) \quad$ where $r_1, \ldots, r_d \leftarrow \mathbb{Z}_q$
5. $c_{f_0} = \mathrm{com}_{ck}(f_0; s_0) \quad \ldots \quad c_{f_d} = \mathrm{com}_{ck}(f_d; s_d) \quad$ where $f_0, s_0, \ldots, f_d, s_d \leftarrow \mathbb{Z}_q$
6. $\delta_0 \quad \ldots \quad \delta_d \in \mathbb{Z}_q$ such that

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j} = X^{d+1}v + \sum_{j=0}^{d} X^j\delta_j$$

7. $c_{\delta_0} = \mathrm{com}_{ck}(\delta_0; t_0), \ldots, c_{\delta_d} = \mathrm{com}_{ck}(\delta_d; t_d)$ where $t_0, \ldots, t_d \leftarrow \mathbb{Z}_q$

8. $c_{fu_0} = \mathrm{com}_{ck}(f_0 u^{2^0}; \xi_0) \quad \ldots \quad c_{fu_{d-1}} = \mathrm{com}_{ck}(f_{d-1} u^{2^{d-1}}; \xi_{d-1})$
   where $\xi_0, \ldots, \xi_{d-1} \leftarrow \mathbb{Z}_q$

Send: $\boldsymbol{c}_a, c_v, \boldsymbol{c}_u, \boldsymbol{c}_f, \boldsymbol{c}_\delta, \boldsymbol{c}_{fu}$

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Compute

1. $\overline{f}_j = xu^{2^j} + f_j \qquad \overline{r}_j = xr_j + s_j$ for all j
2. $\overline{\xi}_j = xr_{j+1} - \overline{f}_j r_j + \xi_j -$ for all j
3. $\overline{t} = x^{d+1}s + \sum_{i=0}^{d} t_i x^i - \sum_{i=0}^{D} \tau_i f_i$ where

$$f_i = \prod_{j=1}^{d} \overline{f}_j^{\,i_j} x^{1-i_j} \text{ for } i = i_0 \ldots i_d.$$

4. Engage in parallel in a SHVZK inverse argument described in Section 5.2 to show $v \neq 0$
5. Engage in parallel in a SHVZK single value product argument, Section 5.3.4, to show

$$\prod_{i=1}^{D} (\lambda_i - x) = \sum_{i=0}^{D} a_i X^i$$

Send: $\overline{f}_0, \overline{r}_0, \ldots, \overline{f}_d, \overline{r}_d, \overline{t}, \overline{\xi}_0, \ldots, \overline{\xi}_{d-1}$

Verification: The verifier accepts $u \notin \mathcal{L}$ if and only if for all j

1. $\boldsymbol{c}_a \in \mathbb{G}^D, \quad \boldsymbol{c}_u, \boldsymbol{c}_f, \boldsymbol{c}_\delta \in \mathbb{G}^{d+1}, \quad \boldsymbol{c}_{fu} \in \mathbb{G}^d, \quad c_v \in \mathbb{G}$
2. $\overline{f}_0, \overline{r}_0, \ldots, \overline{f}_d, \overline{r}_d, \overline{t}, \overline{\xi}_0, \ldots, \overline{\xi}_{d-1} \in \mathbb{Z}_q$
3. $c_{u_j}^x c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j; \overline{r}_j)$
4. $c_{u_{j+1}}^x c_{u_j}^{-\overline{f}_j} c_{fu_j} = \mathrm{com}_{ck}(0; \overline{\xi}_j)$
5. $c_v^{x^{d+1}} \prod_{i=0}^{d} c_{\delta_i}^{x^i} = \boldsymbol{c}_a^{\boldsymbol{f}} \mathrm{com}_{ck}(0; \overline{t}) \quad$ with

$$f_i = \prod_{j=1}^{d} \overline{f}_j^{\,i_j} x^{1-i_j} \text{ for } i = i_0 \ldots i_d.$$

6. The other two SHVZK arguments are valid.

Theorem 30. *The protocol above is a public coin perfect generalized $\Sigma-$protocol of commited $u \in \mathbb{Z}_q$ such that u is not contained in a secrete list $\mathcal{L}$.*

Proof. Perfect completeness follows by careful inspection and from the perfect completeness of the two underlying SHVZK arguments.

We will now argue that we have perfect SHVZK. Upon being challenged with $x \in \mathbb{Z}_q^*$ the simulator picks $c_{u_1}, \ldots, c_{u_d}, c_{\delta_1}, \ldots, c_{\delta_d} \leftarrow \mathbb{G}$ and $\overline{f}_0, \overline{r}_0, \ldots, \overline{f}_d, \overline{r}_d, \overline{t}, \overline{\xi}_0, \ldots, \overline{\xi}_{d-1} \leftarrow \mathbb{Z}_q$ and sets for all j

$$c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j; \overline{r}_j) c_{u_j}^{-x} \qquad c_{fu_j} = \mathrm{com}_{ck}(0; \overline{\xi}_j) c_{u_{j+1}}^{-x} c_{u_j}^{\overline{f}_j}.$$

It calculates $f_i = \prod_{j=1}^{d} \overline{f}_j^{\,i_j} x^{1-i_j}$ for $i = i_0 \ldots i_d$ and sets

$$c_{\delta_0} = c_v^{-x^{d+1}} \prod_{i=0}^{d} c_{\delta_i}^{-x^i} \boldsymbol{c}_a^{\boldsymbol{f}} \mathrm{com}_{ck}(0; \overline{t}).$$

This is a perfect simulation. In a real argument $c_{u_1}, \ldots, c_{u_d}, c_{\delta_1}, \ldots, c_{\delta_d}$ are uniformly random perfectly hiding commitments as in the simulation. In a real argument $\overline{f}_0, \overline{r}_0, \ldots, \overline{f}_d, \overline{r}_d, \overline{t}, \overline{\xi}_0, \ldots, \overline{\xi}_{d-1} \in \mathbb{Z}_q$ are also uniformly random because of the random choice of $f_0, r_0, \ldots, f_d, r_d, t_0, \xi_0, \ldots, \xi_{d-1}$. Finally, both in the simulation and in the real argument these choices through the verification equations uniquely determine the values of $c_{f_0}, \ldots, c_{f_d}, c_{\delta_0}$ and $c_{fu_0}, \ldots, c_{fu_{d-1}}$. This means simulated and real arguments have identical probability distributions.

The SHVZK simulator picks also $c_v \leftarrow \mathbb{G}$ and $\boldsymbol{c}_a \leftarrow \mathbb{G}^D$ at random and runs the SHVZK simulators for the two underlying SHVZK arguments. Since the commitment scheme is perfectly hiding and the underlying SHVZK arguments are perfect SHVZK this gives us perfect SHVZK.

Finally, we have to show that we have perfect generalized special soundness. Given $d+2$ accepting transcripts with different challenges x_i the extractor E runs the extractors for the two underlying SHVZK arguments to get openings u, v satisfying $v \neq 0$ and $P(u) = v$. The extractor E also returns openings $a_0, \ldots, a_D, \lambda_1, \ldots, \lambda_D$ satisfying $P(X) = \prod_{i=1}^{D}(X - \lambda_i) = \sum_{i=0}^{D} a_i X$ by the binding property of the commitment scheme.

Lastly, the extractor E can extract witnesses $u^i, f_i, s_i, r_i, \delta_i$. Given $\overline{f}_j^{(1)}, \overline{r}_j^{(1)}$ and $\overline{f}_j^{(2)}, \overline{r}_j^{(2)}$ in the first two answers to challenges x_1 and x_2 the extractor can take linear combinations of the verification equations to get openings of the commitments c_{u_j}. More precisely, we have that the two answers satisfy

$$c_{u_j}^{x_1} c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j^1; \overline{r}_j^1) \qquad c_{u_j}^{x_2} c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j^2; \overline{r}_j^2).$$

Picking α_1, α_2 such that $\alpha_1 x_1 + \alpha_2 x_2 = 1$ and $\alpha_1 + \alpha_2 = 0$ gives us

$$c_{u_j} = c_{u_j}^{\alpha_1 x_1 + \alpha_2 x_2} c_{f_j}^{\alpha_1 + \alpha_2} = \mathrm{com}_{ck}(\alpha_1 \overline{f}_j^{(1)} + \alpha_2 \overline{f}_j^{(2)}; \alpha_1 \overline{r}_j^{(1)} + \alpha_2 \overline{r}_j^{(2)}),$$

which is an opening of c_{u_j}.

Other types of linear combinations of the verification equations give us openings of the other commitments c_{f_j}, c_{fu_j}, c_v and c_{δ_i} the prover sends in the initial message. In the case of c_{δ_i} we find the linear

combination as follows. Let

$$M = \begin{pmatrix} 1 & x_1 & \dots & x_1^{d+1} \\ \vdots & & & \vdots \\ 1 & x_{d+2} & \dots & x_{d+2}^{d+1} \end{pmatrix}.$$

Since it is a Vandermonde matrix with different $x_1, \dots, x_{d+2}$ it is invertible. By taking linear combinations of the verification equations

$$c_v^{x^{d+1}} \prod_{i=0}^{d} c_{\delta_i}^{x^i} = \boldsymbol{c_a^f} \mathrm{com}_{ck}(0; \overline{t}) = \mathrm{com}_{ck}\left(\sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} \overline{f}_j^{i_j} x^{1-i_j}; x^{d+1} s + \sum_{i=0}^{d} t_i x^i \right)$$

for different challenges $x_1, \dots, x_{d+2}$ we find that

$$\begin{pmatrix} \delta_0 & t_0 \\ \vdots & \vdots \\ \delta_d & t_d \\ v & s \end{pmatrix} = M^{-1} \begin{pmatrix} \sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} (\overline{f}_j^{(1)})^{i_j} x_1^{1-i_j} & x_{(1)}^{d+1} s + \sum_{i=0}^{d} t_i x_{(1)}^i \\ \vdots & \vdots \\ \sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} (\overline{f}_j^{(d+2)})^{i_j} x_{d+2}^{1-i_j} & x_{(d+2)}^{d+1} s + \sum_{i=0}^{d} t_i x_{(d+2)}^i \end{pmatrix}$$

which gives us openings of $c_{\delta_0}, \dots, c_{\delta_d}, c_v$.

We now have openings to all the commitments. Because the commitments are binding, each answer must be computed as they are by an honest prover in the argument. Therefore, the verification equations

$$c_j^x c_{f_j} = \mathrm{com}_{ck}(\overline{f}_j, \overline{r}_j)$$

give us $\overline{f}_j = x u_j + f_j$, where u_j is the extracted value in c_j and f_j is the extracted value in c_{f_j}. The verification equations

$$c_{u_{j+1}}^x c_{u_j}^{-\overline{f}_j} c_{f u_j} = \mathrm{com}_{ck}(0; \overline{\xi}_j)$$

now give us that the committed values satisfy

$$x u_{j+1} - (x u_j + f_j) u_j + \phi_j = 0$$

for $j = 0, \dots, d-1$ with u_j being the value inside c_j and ϕ_j being the value inside $c_{f u_j}$. Since each of the polynomial equalities is of degree 1 and holds for $d+2$ different challenges we see that $u_{j+1} = u_j u_j$. Since $u_0 = u$ this gives us $u_1 = u^{2^1}, u_2 = u^{2^2}, \dots, u_d = u^{2^d}$.

The verification equation

$$c_v^{x^{d+1}} \prod_{i=0}^{d} c_{\delta_i}^{x^i} = \boldsymbol{c_a^f} = \mathrm{com}_{ck}\left(\sum_{i_0,\dots,i_d=0}^{1} a_{i_0 \dots i_d} \prod_{j=0}^{d} \overline{f}_j^{i_j} x^{1-i_j}; \overline{t} \right)$$

we now have that this corresponds to the degree $d+1$ polynomial equation

$$X^{d+1}v + X^d\delta_d + \ldots + X\delta_1 + \delta_0 = \sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j}.$$

With $d+2$ different values $x_1, \ldots, x_{d+2}$ satisfying the equation, we conclude the two polynomials are identical. Looking at the coefficient for X^{d+1} we can see that the extracted openings of c_{u_0} and c_v satisfy $P(u) = v$. Furthermore, the extracted openings u, v are the same openings known by the prover If this were not the case the prover could use the extractor to find another opening for at least one of the commitments, and the commitment scheme would be broken.

Since $v \neq 0$, as seen above, and the $P(u) \neq v$, u is a not a root of the polynomial and therefore not in the list. □

Efficiency: The communication consists of $D + 4d$ group elements and $2D + 3d$ field elements and is therefore linear in the size of the blacklist.

The prover needs $2D$ exponentiations to compute the commitments to the coefficients of $P(X)$ and the exponentiation of the underlying single value product argument, which cost another $3D$ exponentiations. The prover also has to calculate the δ_is. These values are defined to satisfy

$$\sum_{i_0,\ldots,i_d=0}^{1} a_{i_0\ldots i_d} \prod_{j=0}^{d} (Xu^{2^j} + f_j)^{i_j} X^{1-i_j} = X^{d+1}v + \sum_{j=0}^{d} X^j\delta_j.$$

The prover can calculate the degree d polynomials $\prod_{j=0}^{d}(Xu^{2^j} + f_j)^{i_j} X^{1-i_j}$ in a binary-tree fashion for all choices of $i_0 \ldots, i_d \in \{0,1\}$ at a cost of dD multiplications in $\mathbb{Z}_q$. Multiplying with the $a_{i_0\ldots i_d}$'s uses another dD multiplications. The total cost for the prover is therefore $5D$ exponentiations in $\mathbb{G}$ and $2dD$ multiplications in $\mathbb{Z}_q$.

The verifier can check the argument using D exponentiations in $\mathbb{G}$. He also needs to compute the values

$$\prod_{j=0}^{d} \overline{f}_j^{\,i_j} x^{1-i_j}, \text{for } i = i_0 \ldots i_d$$

which can be done in a binary tree fashion for all choices of $i_0, \ldots, i_d \in \{0,1\}$ using D multiplications in $\mathbb{Z}_q$.

We have ignored small constants in the calculations above and just focused on the dominant terms. Using the sliding window technique, from Section 4.3.2, we can reduce the computational burden of the exponentiations. Applying randomized verification and other tricks [BGR98, Gro10] it is possible to reduce the computation even further for the prover and verifier, so the estimates we have given above are quite conservative.

7.4 Membership and Multi Non-Membership Arguments

It is easy to modify our non-membership argument into a membership argument. If u is a member

of a list $\mathcal{L}$ then $P(u) = 0$. Therefore, to get a membership argument we let the prover give a SHVZK argument for c_v containing $v = 0$, instead of demonstrating that v is invertible and hence $v \neq 0$. Zero arguments are standard and only cost a couple of group elements in communication [Sch91].

If we have many committed values $u_1, \ldots, u_L$ and blacklists $\mathcal{L}_1, \ldots, \mathcal{L}_L$ we can adapt the non-membership protocol by committing to the different polynomial evaluations

$$v_1 = P^{(1)}(u_1), \ldots, v_L = P^{(L)}(u_L)$$

. We can now use the invertible argument from Section 5.2 and the polynomial evaluation argument from Section 6.4. This gives a non-membership argument for L blacklists with a communication complexity of $O(\sqrt{L}\log D)$ group and field elements.

To get a membership argument for multiple whitelists we can use the multiple polynomial evaluation argument together with an argument for $v_1 = 0, \ldots, v_L = 0$, which can be constructed using standard techniques [Gro09].

We can also combine whitelists and blacklists where we use the same polynomial evaluation argument to show $v_1 = P^{(1)}(u_1), \ldots, v_L = P^{(L)}(u_L)$. The inverse argument from Section 5.2 can now be used to show exclusion from the blacklists if $v_i \neq 0$. For $v_i = 0$ a selective zero argument can be used to show inclusion of v_i in the whitelists $\mathcal{L}_i$. This gives us a communication complexity of $O(\sqrt{L}\log D)$ for simultaneously showing memberships and non-memberships of a mixed set of L whitelists and blacklists.

7.5 Brands et al.'s [BDD07] Techniques

In this section we will briefly recap Brands et al.'s [BDD07] techniques to construct a non-membership argument for $u \in \mathbb{Z}_q$ and a list $\mathcal{L} = \{\lambda_1, \ldots, \lambda_D\}$ which has $O(\sqrt{D})$ communication cost.

To convince a verifier in zero-knowledge that a secret value $u \in \mathbb{Z}_q$ is not contained in a list $\mathcal{L} = \{\lambda_1, \ldots, \lambda_D\}$ both parties construct $\mathfrak{d} = \sqrt{D}$ polynomials

$$P_j(X) = \prod_{i=1}^{\mathfrak{d}} (X - \lambda_{(j-1)\mathfrak{d}+i}) = a_{j\mathfrak{d}}X^{\mathfrak{d}} + \cdots + a_{j1}X + a_{j0}$$

and the prover shows that for all $j = 1, \ldots, \mathfrak{d}$ that $P_j(u) \neq 0$.

To do this the prover first picks random values $r_i \leftarrow \mathbb{Z}_q, r = 1, \ldots, \mathfrak{d}$, commits themselves to $u, u^2, \ldots, u^{\mathfrak{d}}$ in $c_{u_i} = \mathrm{com}_{ck}(u^i; r_i)$, calculates $v_j = P_j(u)$ and $w_j = a_{jd}r_{\mathfrak{d}} + \ldots + a_{j2}r_2 + a_{j1}r_1$, and conceals these values inside v_j, w_j in $c_{v_j} = \mathrm{com}_{ck}(v_j; w_j)$. The prover sends the commitments to the verifier, who can check that

$$c_{v_j} = c_{u_{\mathfrak{d}}}^{a_{j\mathfrak{d}}} c_{u^{\mathfrak{d}-1}}^{a_{j(\mathfrak{d}-1)}} \ldots c_u^{a_{j1}} \mathrm{com}_{ck}(a_{j0}, 0).$$

If all commitments are constructed correctly the verifier always accept this step as

$$c_{u_\mathfrak{d}}^{a_{j\mathfrak{d}}} c_{u_{\mathfrak{d}-1}}^{a_{j\mathfrak{d}-1}} \dots c_u^{a_{j1}} \mathrm{com}_{ck}(a_{j0}; 0) = \mathrm{com}_{ck}(a_{j\mathfrak{d}} u^{\mathfrak{d}} + \dots + a_{j1} u + a_{j0}; a_{j,m} r_m + \dots + a_{j1}; r_1) = C_{v_j}.$$

This check convince the verifier that the values $P_j(u)$ are hidden inside c_{v_j}.

Parallel to this the prover has also to show that $c_{u_1}, \dots, c_{u_\mathfrak{d}}$ are commitments to the successive powers of u. This is done using standard techniques. To complete the argument the prover has to show that $P_j(u) \neq 0$ for all $j = 1, \dots \mathfrak{d}$, this is also done by standard techniques.

Statement: $\{\mathbb{G}, p, q\},\ ck,\ \mathcal{L} = \{\lambda_1, \dots, \lambda_D\} \subset \mathbb{Z}_q$, $P_j(X) = \prod_{i=1}^{\mathfrak{d}} (X - \lambda_{(j-1)\mathfrak{d}+i})$, and $c_u \in \mathbb{G}_q$

Prover's witness: $u, r_1 \in \mathbb{Z}_q$ such that $u \notin \mathcal{L}$ and $c_{u_1} = (u; r_1)$

Initial message: Compute

1. $c_{u_i} = G^{u^i} \mathrm{com}_{ck}(0; r_i)$ where $r_i \leftarrow \mathbb{Z}_q$ for $i = 2, \dots, \mathfrak{d}$
2. $v_j = P_j(u)$ for $j = 1, \dots, \mathfrak{d}$
3. $w_j = a_{j,\mathfrak{d}} r_\mathfrak{d} + \dots a_{j,2} r_2 + a_{j,1} r_1 \quad$ for $j = 1, \dots \mathfrak{d}$
4. $c_{v_j} = \mathrm{com}_{ck}(v_j; w_j)$
5. $f_1 = \mathrm{com}_{ck}(r_u; s_1) \quad$ where $r_u, s_1 \leftarrow \mathbb{Z}_q$
6. $f_i = c_{u_{i-1}}^{r_u} \mathrm{com}_{ck}(0; s_i) \quad$ where and $s_i \leftarrow \mathbb{Z}_q$ for $i = 2, \dots \mathfrak{d}$,
7. $f_{v_i} = c_{v_i}^{r_{v_i}} \mathrm{com}_{ck}(0; -r_{w_i}) \quad$ for $i = 1, \dots, \mathfrak{d}$ where $r_{v_i}, r_{w_i} \leftarrow \mathbb{Z}_q^*$

Send: $c_{u_2}, \dots, c_{u_\mathfrak{d}}, c_{v_1}, \dots, c_{v_\mathfrak{d}}, f_1, \dots, f_d, f_{v_1}, \dots, f_{v_\mathfrak{d}}$

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer Compute

1. $\overline{u} = xu + r_u$
2. $\overline{r}_1 = xr_1 + s_1 \qquad \overline{r}_i = x(r_i - ur_{i-1}) + s_i$ for $i = 2, \dots, \mathfrak{d}$
3. $\overline{v}_j = xv_i^{-1} + r_{v_i},$ for $i = 1, \dots \mathfrak{d}$
4. $\overline{w}_j = xw_k v_i^{-1} + r_{w_i}$ for $i = 1, \dots \mathfrak{d}$

Send: $\overline{u}, \overline{r}_1, \dots, \overline{r}_\mathfrak{d}, \overline{v_1}, \dots, \overline{v}_\mathfrak{d}, \overline{w}_1, \dots, \overline{w}_\mathfrak{d}$

Verification: Accept if and only if

1. $c_{u_2}, \dots, c_{u_\mathfrak{d}}, c_{v_1}, \dots, c_{v_\mathfrak{d}}, f_1, \dots, f_\mathfrak{d}, f_{v_1}, \dots, f_{v_\mathfrak{d}} \in \mathbb{G}$
2. $\overline{u}, \overline{r}_1, \dots, \overline{r}_\mathfrak{d}, \overline{v_1}, \dots, \overline{v}_\mathfrak{d}, \overline{w}_1, \dots, \overline{w}_\mathfrak{d} \in \mathbb{Z}_q$
3. $\prod_{j=1}^{\mathfrak{d}} c_{v_j}^{\sigma_j} = \mathrm{com}_{ck}(\sum_{j=1}^{\mathfrak{d}} a_{j,0}\sigma_j; 0) \prod_{i=1}^{\mathfrak{d}} c_{u_{\mathfrak{d}i}}^{\sum_{j=1}^{\mathfrak{d}} a a_{j,i}\sigma_j}$ where $\sigma_j \leftarrow \mathbb{Z}_q$, $j = 1, \dots, \mathfrak{d}$
4. $\mathrm{com}_{ck}(\overline{u}; \overline{r}_1) c_{u_1}^{-x} = f_1 \qquad c_{u_{i-1}}^{\overline{u}} \mathrm{com}_{ck}(0; \overline{r}_i) c_{u_i}^{-x} = f_i, \quad$ for $i = 2, \dots, \mathfrak{d}$
5. $c_{v_j}^{\overline{v}_j} \mathrm{com}_{ck}(-x; \overline{w}) = f_{v_j} \quad$ for $j = 1, \dots \mathfrak{d}$

Theorem 31. *The protocol above is a perfect public coin perfect generalized $\Sigma-$protocol for $u \in \mathbb{Z}_q$ such that $u \notin \mathcal{L}$.*

Proof. Perfect completeness can be seen by inspection of the verification equations.

To show that the argument is perfect SHVZK the simulator picks $c_{u_2}, \ldots, c_{u_\mathfrak{d}}, c_{v_2}, \ldots, c_{v_\mathfrak{d}}$ as random commitments to 0 and $\overline{u}, \overline{r}_1, \ldots, \overline{r}_\mathfrak{d}, \overline{v_1}, \ldots, \overline{v}_\mathfrak{d}, \overline{w}_1, \ldots, \overline{w}_\mathfrak{d} \leftarrow \mathbb{Z}_q$ at random. The simulator picks $\sigma_i \leftarrow \mathbb{Z}_q$ and computes

$$c_{v_1} = \left[\mathrm{com}_{ck}\Big(\sum_{j=1}^{\mathfrak{d}} a_{j,0}\sigma_j; 0 \Big) \prod_{i=1}^{\mathfrak{d}} c_{u_{d_i}}^{\sum_{j=1}^{\mathfrak{d}} a_{j,i}\sigma_j} \prod_{j=2}^{\mathfrak{d}} c_{v_j}^{-\sigma_j} \right]^{\sigma_1^{-1}}.$$

It also sets $f_1 = \mathrm{com}_{ck}(\overline{u}; \overline{r}_1) c_{u_1}^{-x}$, $f_i = c_{u_{i-1}}^{\overline{u}} \mathrm{com}_{ck}(0; \overline{r}_i) c_{u_i}^{-x}$, for $i = 2, \ldots, \mathfrak{d}$ and $f_{v_i} = c_{v_i}^{\overline{v}_i} \mathrm{com}_{ck}(-x; \overline{w})$ for $i = 1, \ldots \mathfrak{d}$.

The answers are uniformly random in the simulation similar to the real argument. Since the commitment scheme is unconditional hiding; thus all commitments in the simulation and the real argument follow the same distribution. Therefore, the argument is perfect SHVZK.

Lastly, we have to check that the protocol has perfect generalized special soundness. Given two accepting arguments with challenges $x_1 \neq x_2$ the extractor E can extract openings of all commitments. The found witnesses fulfill the statement, as the commitment scheme is binding. More precisely, as the commitment scheme is binding, the verification equation

$$c_{u_\mathfrak{d}}^{a_{j\mathfrak{d}}} c_{u_{\mathfrak{d}-1}}^{a_{j\mathfrak{d}-1}} \ldots c_u^{a_{j1}} \mathrm{com}_{ck}(a_{j0}; 0)$$

gives us openings satisfying

$$a_{j\mathfrak{d}} \tilde{u}_\mathfrak{d} + \ldots + a_{j1} \tilde{u}_1 + a_{j0} = v_j,$$

for all j, where $\tilde{u}_i$ are the openings of c_{u_i}. From the equations $\mathrm{com}_{ck}(\overline{u}; \overline{r}_1) c_{u_1}^{-x} = f_1$, $c_{u_{i-1}}^{\overline{u}} \mathrm{com}_{ck}(0; \overline{r}_i) c_{u_i}^{-x} = f_i$ if follows that c_{u_i} contain indeed the successive powers of u and therefore $v_j = P_j(u)$ for all j.

The equation $c_{v_j}^{\overline{v}_j} \mathrm{com}_{ck}(-x; \overline{w}) = f_{v_j}$ holds for two different challenges x. Since there is only a negligible chance that this equation holds unless v_j is invertible, so the extracted witnesses v_j are invertible and therefore $v_j \neq 0$ for all j.

We can conclude that, the extracted witnesses fulfill the statement and the witness u is not in the list $\mathcal{L}$. Furthermore, the extracted openings are exactly the witnesses of the prover. If this is not the case, the extractor of the argument could be used by the prover to find a second opening of at least one of the commitments with non-negligible probability and the commitment scheme is broken. □

Efficiency: During the protocol $4\mathfrak{d}$ group elements and $3\mathfrak{d}$ field elements are transferred between the prover and verifier; therefore, the communication cost of the argument is $O(\mathfrak{d}) = O(\sqrt{D})$.

The prover has to compute $8\mathfrak{d}$ exponentiations in $\mathbb{G}$ and total number of $2D + 8\mathfrak{d}$ multiplications. The verifier has to calculate $8\mathfrak{d}$ exponentiations and $2D + 3\mathfrak{d}$ multiplications. In the original description

of the protocol by Brands et al. the verifier needed to calculate $D + 6\mathfrak{d}$ exponentiation in $\mathbb{G}$ and $D + 3\mathfrak{d}$ multiplications. However, Brands et al. also report that it is possible to collapse the equations of step 3 in the verification to a single batch verification using techniques by Bellare et al. [BGR98], this technique is used in our description. Other randomization techniques could be used to reduce the computational burden of the verifier, see also [Gro10].

Example: Let be $\mathbb{G} = \langle G \rangle = \langle 149 \rangle \subset \mathbb{Z}^*_{179}$, with prime order $q = 89$. The statement consists of the blacklist $\mathcal{L} = \{9, 26, 49, 48\}$, the corresponding polynomials
$P_1(X) = X^2 + 54X + 54$ and $P_2(X) = X^2 + 81X + 38$, $\mathfrak{d} = \sqrt{4} = 2$, the commitment $c_{u_1} = 22$, and $\mathbb{G} = \{p, q\} = \{179, 89\}$, $ck = \{149, 76\}$.

The prover knows $u = 84, r_1 = 43$ such that $c_{u_1} = 22$. To convince the verifier that the witness does not belong to the blacklist, the prover picks $r_2 = 26$ at random, and commits themselves to

$$c_{u2} = \mathrm{com}_{ck}(u^2; r_2) = 36.$$

They also evaluate $P_1(X), P_2(X)$ in u to get $v_1 = P_1(u) = 78$ and $v_2 = P_2(X) = 14$, calculates $w_j = a_{j,d} r_\delta + \ldots a_{j,2} r_2 + a_{j,1} r_1$ for $j = 1, 2$,

$$w_1 = 24 \in \mathbb{Z}_{89} \qquad w_2 = 28 \in \mathbb{Z}_{89}$$

and commits in these values

$$c_{v1} = \mathrm{com}_{ck}(v_1; w_1) = 20 \qquad c_{v2} = \mathrm{com}_{ck}(v_2; w_2) = 144.$$

The prover also picks random values $r_u = 80, s_1 = 25, s_2 = 52$ and computes

$$f_1 = \mathrm{com}_{ck}(r_u; s_1) = 82 \qquad f_2 = c_u^{r_u} H^{s_1} = 25$$

and picking random $r_{v1} = 1, r_{v2} = 47, r_{w1} = 44, r_{w2} = 57$ the prover sets

$$f_{v1} = c_{v_1}^{r_{v1}} H^{-r_{w1}} = 68 \qquad f_{v2} = c_{v2}^{r_{v2}} H^{-r_{w2}} = 75.$$

Finally, the prover send the committed values

$$c_{u2} = 36 \quad c_{v1} = 20 \quad c_{v2} = 144 \quad f_1 = 82 \quad f_2 = 25 \quad f_{v1} = 68 \quad f_{v2} = 75$$

to the verifier.

The verifier picks challenge $x = 77$ and the prover answers with

$$\overline{u} = 51 \qquad \overline{r}_1 = 43 \qquad \overline{r}_1 = 39$$

$$\overline{v}_1 = 83 \qquad \overline{v}_2 = 8 \qquad \overline{w}_1 = 54 \qquad \overline{w}_2 = 33.$$

The verifier checks if all commitments are valid and if all answers are in $\mathbb{Z}_{89}$. He picks $\sigma_1 = 26, \sigma_2 = 81$ and checks

$$c_{v_1}^{\sigma_1} c_{v_2}^{\sigma_2} = 20^{26} 144^{81} = 100 = G^{\sum_{j=1}^{2} a_{j,0}\sigma_j} \prod_{i=1}^{2} c_{u_{d_i}}^{\sum_{j=1}^{2} a_{j,i}\sigma_j} \quad (\checkmark)$$

$$f_1 = 82 = G^{\overline{u}+e\overline{r}_1} c_{u1}^{x} \quad (\checkmark) \qquad f_2 = 25 = c_{u1}^{\overline{u}} H^{\overline{r}_2} c_{u2}^{-x} \quad (\checkmark)$$

$$f_{v1} = 68 = c_{v1}^{\overline{v}_1} G^{-(e\overline{w}_1+x)} \quad (\checkmark) \qquad f_{v2} = 75 = c_{v2}^{\overline{v}_2} G^{-(e\overline{w}_2+x)} \quad (\checkmark)$$

7.5.1 Implementation and practical results

To obtain some experimental results and analyze the real life behavior we implemented Brands et al.'s blacklist protocol. We collected data for different groups with different levels of security. We have chosen subgroups with 160-bit order modulo a 1 248-bit, a 1 536-bit, and a 3 248-bit prime. Three 256-bit subgroups modulo a 1 536, a 2 432, and a 3 248-bit prime, and lastly a 384-bit subgroups modulo a 3 248-bit prime. Some of these groups are not standard and are not used in real life. We have chosen these groups to learn more about the influence of the different parameters, for example subgroup size. The security levels of ours groups together with a detailed discussion why we have chosen them can be found in Section 4.1.1.

The polynomial argument in Section 6.5 is based on this blacklist argument, but in this setting the prover and verifier needs to calculate only half of the number of exponentiations; therefore, we expect faster performance.

We implemented different levels of optimization. First, we wanted to analyze the merit of batching the d test equations together into one equation. Thus, we coded first the un-batched version and an optimized version using the multi-exponentiation techniques from Section 4.3. Then, we implemented the protocol using the batched verification. Again we optimized this version using the multi-exponentiation techniques.

In Table 7.7 we can find the results of the un-optimized protocol without batching and with batching for a 256-bit subgroup modulo a 1 536-bit prime. We see that the performance of the verifier without batching is very slow compared to the batched verifier, batching alone helps to speed up the protocol by a factor up to 140 for blacklists with huge number of elements.

The next step is to analyze if the multi-exponentiation techniques give a better performance speed increase than the batching step. We find in Table 7.7 the data for the optimized un-batched version, and the optimized batched protocol. We see that optimizing the un-batched version increase the performance drastically, Brickels et al's technique gives a asymptotically speed up factor of $\log(\mathfrak{d})$ when calculating a multi-exponentiation of $\mathfrak{d}$ elements. We see that in our implementation the influence of the multi-exponentiation techniques gets bigger for big $\mathfrak{d} = \sqrt{D}$, but is less than $\log(\mathfrak{d})$. We also see that the run-time of the optimized un-batched version is still slower than the un-optimized version batched version. Applying multi-exponentiation techniques reduces this run-time even further.

	un-batched:		batched:	
Elements	Conservative			
in list D	Prover	Verifier	Prover	Verifier
10	23 ms	21 ms	23 ms	20 ms
100	61 ms	109 ms	58 ms	50 ms
1 000	199 ms	880 ms	191 ms	163 ms
5 000	430 ms	3 928 ms	442 ms	377 ms
10 000	596 ms	7 557 ms	615 ms	520 ms
50 000	1 366 ms	36 342 ms	1 373 ms	1 165 ms
100 000	1 935 ms	72 003 ms	1 952 ms	1 650 ms
500 000	4 572 ms	351 593 ms	4 562 ms	3 748 ms
1 000 000	6 742 ms	695 634 ms	6 735 ms	5 541 ms
	Optimized			
	Prover	Verifier	Prover	Verifier
10	16 ms	52 ms	17 ms	15 ms
100	41 ms	171 ms	42 ms	38 ms
1 000	134 ms	684 ms	133 ms	118 ms
5 000	304 ms	1 972 ms	299 ms	260 ms
10 000	444 ms	3 273 ms	423 ms	368 ms
50 000	980 ms	11 270 ms	975 ms	831 ms
100 000	1 408 ms	19 847 ms	1 408 ms	1 186 ms
500 000	3 319 ms	79 660 ms	3 414 ms	2 763 ms
1 000 000	4 863 ms	150 419 ms	5 101 ms	4 075 ms

Table 7.7: Run-time in ms of Brands et al.'s blacklist argument for a verifier without batching and with batching on a 256-bit subgroup modulo a 1 536-bit prime.

Therefore, we can say that substitution a high number of exponentiation with the same asymptotic number of multiplications and a fewer number of exponentiations has a great effect on the run-time. That means that in general batching many equations into one helps to reduce the computational burden drastically.

We also have to investigate the influence of the different multi-exponentiations techniques on the batched protocol. Table 7.8 and 7.9 state the data for the un-optimized version, a version only optimized with the sliding window technique, and the last version also optimized with Lim-Lee's respectively Brickels et al.'s technique for two different groups $\mathbb{G}$. We see that the merit of the sliding window technique is bigger for bigger moduli p, but for all degree D in one setting is more or less the same. This means that exponentiations are dominant over the multiplication. The influence on the prover is bigger than the influence on the verifier, but this was expected, as the verifier has to calculate the batched verification equation. We see that the cost of this equation can be reduced using the other multi-exponentiation techniques, again the merit is bigger the bigger the moduli p gets. One explanation is that in this case the cost for a single-exponentiation gets higher and therefore the optimization works better.

Table 7.10 states the results for subgroups with fixed order of 160-bit modulo a 1 248-bit prime and a 1 536-bit prime, whereas Table 7.11 states the results for fixed 256-bit prime modulo different primes. The first thing we see is that the performance in all cases is less than half of the performance of the polynomial argument, see Section 6.5, but this was expected. We also see that increasing the moduli values slows the run-time down of the protocol. This effect is super-linear with the increase of

	Conservative	SW	Optimized	Ratio	
D	Prover			SW/Cons.	Opt./Cons.
10	23 ms	16 ms	17 ms	0.68	0.72
100	58 ms	39 ms	42 ms	0.67	0.72
1 000	191 ms	126 ms	133 ms	0.66	0.70
5 000	442 ms	285 ms	299 ms	0.64	0.67
10 000	615 ms	407 ms	423 ms	0.66	0.69
50 000	1 373 ms	998 ms	975 ms	0.73	0.71
100 000	1 952 ms	1 343 ms	1 408 ms	0.69	0.72
500 000	4 562 ms	3 230 ms	3 414 ms	0.71	0.75
1 000 000	6 735 ms	5 091 ms	5 101 ms	0.76	0.76
	Verifier				
10	20 ms	18 ms	17 ms	0.91	0.79
100	50 ms	46 ms	38 ms	0.91	0.76
1 000	163 ms	147 ms	118 ms	0.90	0.72
5 000	377 ms	330 ms	260 ms	0.87	0.69
10 000	520 ms	470 ms	368 ms	0.90	0.71
50 000	1 165 ms	1 138 ms	831 ms	0.98	0.71
100 000	1 650 ms	1 515 ms	1 186 ms	0.92	0.72
500 000	3 748 ms	3 524 ms	2 763 ms	0.94	0.74
1 000 000	5 541 ms	5 116 ms	4 075 ms	0.92	0.74

Table 7.8: Comparison of Brands et al.'s blacklist argument for different blacklist sizes and different levels of optimization on a 256-bit subgroup modulo a 1 536-bit prime.

	Cons.	SW	Opt.	Ratio	
D	Prover			SW/Cons.	Opt./Cons.
10	88 ms	54 ms	55 ms	0.61	0.62
100	221 ms	135 ms	138 ms	0.61	0.63
1 000	718 ms	440 ms	439 ms	0.61	0.61
5 000	1 593 ms	985 ms	975 ms	0.62	0.61
10 000	2 246 ms	1 393 ms	1 377 ms	0.62	0.61
50 000	5 015 ms	3 114 ms	3 114 ms	0.62	0.62
100 000	7 076 ms	4 432 ms	4 436 ms	0.63	0.63
500 000	15 951 ms	10 064 ms	10 190 ms	0.63	0.64
1 000 000	22 962 ms	14 782 ms	14 694 ms	0.64	0.64
	Verifier				
10	74 ms	66 ms	54 ms	0.89	0.73
100	192 ms	172 ms	137 ms	0.90	0.71
1 000	620 ms	556 ms	425 ms	0.90	0.69
5 000	1 389 ms	1 249 ms	936 ms	0.90	0.67
10 000	1 946 ms	1 766 ms	1 316 ms	0.91	0.68
50 000	4 352 ms	3 924 ms	2 954 ms	0.90	0.68
100 000	6 139 ms	5 560 ms	4 189 ms	0.91	0.68
500 000	13 693 ms	12 409 ms	9 475 ms	0.91	0.69
1 000 000	19 627 ms	17 729 ms	13 558 ms	0.90	0.69

Table 7.9: Comparison of Brands et al.'s blacklist argument for different blacklist sizes and different levels of optimization on a 256-bit subgroup modulo a 3 248-bit prime.

the moduli, the bigger the prime p gets the more noticeable is the effect. One explanation for this is that for bigger moduli the cost for the exponentiations gets higher and the increase of this cost is not linear. Together with the dominance of the exponentiations' cost this effect occurs. That means, it is

$\lvert q\rvert = 160$	$\lvert p\rvert = 1\,248$		$\lvert p\rvert = 1\,536$	
D	Prover	Verifier	Prover	Verifier
10	9 ms	7 ms	11 ms	10 ms
100	21 ms	18 ms	27 ms	24 ms
1 000	67 ms	54 ms	85 ms	75 ms
5 000	150 ms	120 ms	192 ms	165 ms
10 000	214 ms	169 ms	272 ms	233 ms
50 000	502 ms	390 ms	634 ms	532 ms
100 000	737 ms	565 ms	923 ms	762 ms
500 000	1 887 ms	1 393 ms	2 301 ms	1 808 ms
1 000 000	2 918 ms	2 142 ms	3 505 ms	2 719 ms

Table 7.10: Comparison of Brands et al.'s blacklist argument for different blacklist sizes for a fixed 160-bit subgroup modulo a 1 248-bit prime and a 1 536-bit prime.

$\lvert q\rvert = 256$	$\lvert p\rvert = 1\,536$		$\lvert p\rvert = 2\,432$		$\lvert p\rvert = 3\,248$	
D	Prover	Verifier	Prover	Verifier	Prover	Verifier
10	17 ms	15 ms	36 ms	35 ms	55 ms	54 ms
100	42 ms	38 ms	89 ms	88 ms	138 ms	137 ms
1 000	133 ms	118 ms	283 ms	271 ms	439 ms	425 ms
5 000	299 ms	260 ms	629 ms	596 ms	975 ms	936 ms
10 000	423 ms	368 ms	888 ms	839 ms	1 377 ms	1 316 ms
50 000	975 ms	831 ms	2 016 ms	1 885 ms	3 114 ms	2 954 ms
100 000	1 408 ms	1 186 ms	2 882 ms	2 676 ms	4 436 ms	4 189 ms
500 000	3 414 ms	2 763 ms	6 706 ms	6 094 ms	10 190 ms	9 475 ms
1 000 000	5 101 ms	4 075 ms	9 751 ms	8 756 ms	14 694 ms	13 558 ms

Table 7.11: Comparison of Brands et al.'s blacklist argument for different blacklist sizes for a fixed 256-bit subgroup modulo a 1 536-bit, a 2 432-bit and a 3 248-bit prime.

not possible to enlarge security by keeping the same group order and choosing a bigger moduli, without compromising performance.

$\lvert p\rvert = 1\,536$	$\lvert q\rvert = 160$		$\lvert q\rvert = 256$	
D	Prover	Verifier	Prover	Verifier
10	11 ms	10 ms	17 ms	15 ms
100	27 ms	24 ms	42 ms	38 ms
1 000	85 ms	75 ms	133 ms	118 ms
5 000	192 ms	165 ms	299 ms	260 ms
10 000	272 ms	233 ms	423 ms	368 ms
50 000	634 ms	532 ms	975 ms	831 ms
100 000	923 ms	762 ms	1 408 ms	1 186 ms
500 000	2 301 ms	1 808 ms	3 414 ms	2 763 ms
1 000 000	3 505 ms	2 719 ms	5 101 ms	4 075 ms

Table 7.12: Comparison of Brands et al.'s blacklist argument for different blacklist sizes for different order sizes modulo a fixed prime p with 1 536-bit.

In Table 7.12 and Table 7.13 we can find the reverse case of fixed moduli size. Again we see that increasing the group order increases the run-time of the protocol. But in this case the increase of the time is more or less linear with the increase of the group order. A consequence of this is, that increasing the security level by a certain factor the run-time of the protocols slows down by the same factor. Thus, we

$\lvert p\rvert = 3\,248$	$\lvert q\rvert = 160$		$\lvert q\rvert = 256$		$\lvert q\rvert = 384$	
D	Prover	Verifier	Prover	Verifier	Prover	Verifier
10	35 ms	35 ms	55 ms	54 ms	79 ms	80 ms
100	87 ms	88 ms	138 ms	137 ms	179 ms	200 ms
1 000	279 ms	269 ms	439 ms	425 ms	632 ms	619 ms
5 000	621 ms	569 ms	975 ms	936 ms	1 407 ms	1 369 ms
10 000	877 ms	838 ms	1 377 ms	1 316 ms	1 986 ms	1 925 ms
50 000	1 987 ms	1 881 ms	3 114 ms	3 924 ms	4 508 ms	4 330 ms
100 000	2 837 ms	2 674 ms	4 436 ms	4 189 ms	6 409 ms	6 122 ms
500 000	6 582 ms	6 073 ms	10 190 ms	9 475 ms	14 785 ms	13 863 ms
1 000 000	9 556 ms	8 743 ms	14 694 ms	13 558 ms	21 381 ms	19 876 ms

Table 7.13: Comparison of Brands et al.'s blacklist argument for different blacklist sizes for different order sizes modulo a fixed prime p with 3 248-bit.

can conclude that higher security compromises speed.

We see that Brands et al.'s blacklist argument is practical for small blacklist sizes D for all levels of security, but for bigger D the practicality suffers for medium and higher security levels.

Next, we have to consider the argument size, Table 7.14 states the size for all three rounds separately for a 256-bit subgroup modulo a 1 536-bit prime. We see that the challenge in round adds nothing to the complete argument size. We also see that the size of round 3 is much smaller that round 1, and adds more or less no weight. More precisely, we send $4\mathfrak{d}$ group elements in round 1 and $3\mathfrak{d}$ field elements in round 3, if a group element is n times bigger than a field element, round 1 should be approximately $\frac{3n}{4}$ times bigger than round 3. In our case this factor should be 8, which is the case in reality.

We also see that Brands et al.'s communication cost is smaller than the statement for $D \leq 5\,000$, and the sublinear asymptotic is fulfilled for growing D.

D	Round 1	Round 2	Round 3	Statement
10	11 KB	0.08 KB	1 KB	1 KB
100	33 KB	0.08 KB	4 KB	8 KB
1 000	115 KB	0.08 KB	12 KB	72 KB
5 000	260 KB	0.08 KB	27 KB	381 KB
10 000	367 KB	0.08 KB	39 KB	761 KB
50 000	827 KB	0.08 KB	87 KB	3.9 MB
100 000	1.2 MB	0.08 KB	123 KB	7.8 MB
500 000	2.6 MB KB	0.08 KB	275 KB	38.9 MB
1 000 000	1.9 MB	0.8 KB	238 KB	77.8 MB

Table 7.14: Size of each round and of the statement for different degree D for a 256-bit subgroup modulo a 1 526-bit prime.

In Table 7.15 we find the size of the complete argument for different combinations of order and moduli values. We see that that for small D the size consists only of a few kilobytes but for bigger D the square root asymptotic kicks in and the size grows to a few megabytes. In web based applications, for example blacklist access to Wikipedia, this could lead to a problem as the traffic can get high if many people try to access at the same time.

In conclusion we can say, Brands et al.'s blacklist argument is practical in situations where the

$\|q\|$	160	160	256	256	256	384	384
$\|p\|$	1 248	1 536	1 536	2 432	3 248	3 248	7 984
10	7 KB	8 KB	9 KB	13 KB	17 KB	17 KB	40 KB
100	17 KB	20 KB	21 KB	32 KB	42 KB	43 KB	100 KB
1000	53 KB	64 KB	67 KB	102 KB	133 KB	137 KB	318 KB
5000	117 KB	142 KB	148 KB	225 KB	295 KB	303 KB	704 KB
10000	165 KB	200 KB	209 KB	317 KB	415 KB	427 KB	991 KB
50000	370 KB	448 KB	468 KB	710 KB	929 KB	955 KB	2.2 MB
100000	524 KB	634 KB	662 KB	1.0 MB	1.3 MB	1.4 MB KB	3.1 MB
500000	1.2 MB KB	1.5 MB	1.5 MB	2.2 MB	2.9 MB	3.0 MB	7.0 MB
1 000 000	1.6 MB	1.9 MB	2.0 MB	3.0 MB	4.0 MB	4.1 MB	9.5 MB

Table 7.15: Size whole blacklist argument for different degree D for all choices of groups $\mathbb{G}$.

blacklist size is relatively small and the verifier does not need to verify many protocols in parallel.

7.6 Comparison

SHVZK argument	Rounds	Time $\mathcal{P}$ Expos	Time $\mathcal{V}$ Expos	Size Elements
[BDD07]	3	$6\sqrt{D}$	$D + 5\sqrt{D}$	$5\sqrt{D}\,\mathbb{G} + 3\sqrt{D}\,\mathbb{Z}_p$
[PB10]	3	$8D$	$8D$	$8D\,\mathbb{G} + 6D\,\mathbb{Z}_p$
[Pen11]	3	$7\sqrt{D}$	$10\sqrt{D}$	$5\sqrt{D}\,\mathbb{G} + 5\sqrt{D}\,\mathbb{Z}_p$
This work	3	$8\log D$	$7\log D$	$4\log D\,\mathbb{G} + 3\log D\,\mathbb{Z}_p$

Table 7.16: Comparison of our non-membership argument with earlier work

Theoretical: In the setting of membership and non-membership arguments, the communication cost can be reduced to $O(\sqrt{D})$ as suggested in Brands et al. [BDD07] by splitting the polynomial into $\sqrt{D}$ polynomials of degree $\sqrt{D}$ each. Table 7.16 gives a theoretical performance comparison of our argument to other solutions of non-membership proofs. The best solution so far achieved only $O(\sqrt{D})$ communication cost, we only require $O(\log D)$ communication cost.

The verifier in [Pen11], which is the best so far, only needs $O(\sqrt{D})$ exponentiations but also has to solve a D dimensional system of linear equations on top of this. Our verifier needs only to calculate $7\log D$ exponentiations, which is cheap even if D gets very large.

Just looking at the numbers of exponentiations needed by the prover can be a little deceptive since in our polynomial evaluation argument we need $O(D\log D)$ multiplications in $\mathbb{Z}_q$ to compute the δ_j values and for very large D this cost becomes dominant. Our performance gain for the prover is largest in the medium range, where D is large enough for $\log D$ to be significantly smaller than $\sqrt{D}$ yet not so large that the cost of $D\log D$ multiplications in $\mathbb{Z}_p$ becomes dominant.

Practical: We implemented our non-membership argument for a single blacklist and we also implemented Brands et al.'s [BDD07] technique. To compare both approaches we used a 256-bit subgroup modulo a 1 536-bit prime and a 3 248-bit prime, and a 384-bit subgroup modulo a 7 936-bit prime, assumed that the polynomial $P(X)$ had been pre-computed and obtained the run-time for blacklists between 10 and 1 000 000 elements. This gives us enough data to compare the protocols for different levels

of security. The results can be found in Table 7.17, 7.18, and 7.19.

Elements	Prover		Verifier		Size	
in list D	Brands et al.	This work	Brands et al.	This work	Brands et al.	This work
10	17 ms	18 ms	15 ms	20 ms	9 KB	11 KB
100	42 ms	29 ms	38 ms	34 ms	21 KB	17 KB
1 000	133 ms	47 ms	118 ms	49 ms	67 KB	23 KB
5 000	299 ms	109 ms	260 ms	68 ms	148 KB	28 KB
10 000	423 ms	189 ms	368 ms	83 ms	209 KB	31 KB
50 000	975 ms	754 ms	831 ms	148 ms	468 KB	36 KB
100 000	1 408 ms	1 495 ms	1 186 ms	216 ms	662 KB	38 KB
500 000	3 414 ms	7 369 ms	2 763 ms	693 ms	1.5 MB	42 KB
1 000 000	5 101 ms	15 384 ms	4 075 ms	1 310 ms	4.1 MB	42 KB

Table 7.17: Comparison of our non-membership argument with Brands et al. [BDD07]. All experiments used a 256-bit subgroup modulo a 1 536-bit prime.

Elements	Prover		Verifier		Size	
in list D	Brands et al.	This work	Brands et al.	This work	Brands et al.	This work
10	55 ms	63 ms	54 ms	76 ms	17 KB	21 KB
100	138 ms	99 ms	137 ms	125 ms	42 KB	33 KB
1,000	439 ms	144 ms	425 ms	176 ms	133 KB	46 KB
5,000	975 ms	253 ms	936 ms	235 ms	295 KB	58 KB
10,000	1,377 ms	361 ms	1,316 ms	261 ms	415 KB	61 KB
50,000	3,114 ms	1,114 ms	2,954 ms	358 ms	929 KB	71 KB
100,000	4,436 ms	2 130 ms	4 189 ms	455 ms	1.3 MB	75 KB
500 000	10 190 ms	7 827 ms	9 475 ms	1 019 ms	2.9 MB KB	83 KB
1 000 000	14 694 ms	16 137 ms	13 558 ms	1 572 ms	4.0 MB	85 KB

Table 7.18: Comparison of our non-membership argument with Brands et al. [BDD07]. All experiments used a 256-bit subgroup modulo a 3 248-bit prime.

Elements	Prover		Verifier		Size	
in list D	Brands et al.	This work	Brands et al.	This work	Brands et al.	This work
10	325 ms	397 ms	332 ms	483 ms	40 KB	50 KB
100	817 ms	630 ms	841 ms	808 ms	100 KB	80 KB
1 000	2 615 ms	892 ms	2 600 ms	1 161 ms	318 KB	109 KB
5 000	5 818 ms	1 192 ms	5 743 ms	1 497 ms	704 KB	139 KB
10 000	8 201 ms	1 358 ms	8 060 ms	1 621 ms	991 KB	149 KB
50 000	18 409 ms	2 190 ms	18 032 ms	1 894 ms	2.2 MB	169 KB
100 000	26 070 ms	3 179 ms	25 547 ms	2 090 ms	3.1 MB	178 KB
500 000	58 649 ms	10 839 ms	57 131 ms	2 857 ms	7.0 MB	198 KB
1 000 000	83 300 ms	21 119 ms	80 775 ms	3 790 ms	9.5 MB	203 KB

Table 7.19: Comparison of our non-membership argument with Brands et al. [BDD07]. All experiments used a 384-bit subgroup modulo a 7 984-bit prime.

For very small blacklists $\mathcal{L}$ our argument is only slightly smaller than Brands et al.'s communication and for size $D > 100$ our protocol outperforms Brands et al.. This advantage grows and for medium to large lists our communication cost is noticeable smaller.

Brands et al.'s [BDD07] approach has a verifier that has square root cost, where as our verifier need

only to calculate a logarithmic number of exponentiations. As expected our verifier runs faster than their verifier apart from the cases with very small blacklists. The bigger the blacklist gets and the higher the security parameters are chosen the more noticeable is the different between the two verifiers.

Our prover is faster for moderate blacklist sizes, but we see that for big D the cost to calculate the δ_i's becomes more expensive and Brands et al.'s prover becomes faster. But it is the case that very large D is required before the prover of Brands et al. becomes better from a computational perspective, for moderate size D we have a clear performance advantage. Overal our argument is faster than the combined prover and verifier of Brands et al., the only exception for this are settings with a very small security parameter and a very large D.

Thus, we can conclude that our protocol gives better performance than Brands et al. in all reasonable settings.

Chapter 8

Zero-Knowledge Shuffle Argument

This chapter was joint work with Jens Groth and published at Eurocrypt 2012 [BG12]. We have added a deeper discussion on implementation results.

8.1 Introduction

In recent years the governments of various countries considered establishing e-voting for major elections. Practical and secure e-voting schemes are still under construction, for this reason various countries tested different approaches in local elections. For instance in England tests were carried out for various elections since 2000 and in Estonia since 2005 e-voting is used for general elections[1].

All the approaches used have in common that the construction should guarantee the correctness and secrecy of the vote. One major approach to construct e-voting schemes uses mix-nets [Cha81]. A mix-net is a multi-party protocol which can also be used for other applications which require anonymity, such as anonymous broadcast. It allows a group of senders to input a number of encrypted messages to the mix-net, which then outputs them in random order. It is common to construct mix-nets from shuffles. Other applications for shuffles are among others onion-routing and oblivious databases.

Informally, a shuffle of ciphertexts $C_1, \dots C_N$ is a set of ciphertexts $C'_1, \dots, C'_N$ with the same plaintexts in permuted order. In this chapter we will examine shuffle protocols constructed from homomorphic encryption schemes. That means for a given public key pk, messages M_1, M_2 and randomness ρ_1, ρ_2 the encryption function satisfies $\mathcal{E}_{pk}(M_1 M_2; \rho_1 + \rho_2) = \mathcal{E}_{pk}(M_1; \rho_1)\mathcal{E}_{pk}(M_2; \rho_2)$. Thus, we may construct a shuffle of $C_1, \dots, C_N$ by selecting a permutation $\pi \in \Sigma_N$ and randomizers $\rho_1, \dots \rho_N$, and calculating $C'_1 = C_{\pi(1)}\mathcal{E}_{pk}(1; \rho_1), \dots, C'_N = C_{\pi(N)}\mathcal{E}_{pk}(1; \rho_N)$.

A common construction of mix-nets is to let the mix-servers take turns in shuffling the ciphertexts. If the encryption scheme is IND-CPA secure the shuffle $C'_1, \dots, C'_N$ output by a mix-server does not reveal the permutation or the messages. But this also means that a malicious mix-server in the mix-net could substitute some of the ciphertexts without being detected. In a voting protocol, it could for instance replace all ciphertexts with encrypted votes for candidate X. Therefore, the goal is to construct an interactive zero-knowledge argument that makes it possible to verify that the shuffle was done correctly (soundness), but reveals nothing about the permutation or the randomizers used (zero-knowledge).

[1] `http://en.wikipedia.org/wiki/Electronic_voting_examples`

We propose a practical efficient honest verifier zero-knowledge argument for the correctness of a shuffle. Our argument is very efficient, in particular we drastically decrease the communication complexity compared to previous shuffle arguments. We cover the case of shuffles of ElGamal ciphertexts but it is possible to adapt our argument to other homomorphic cryptosystems as well.

Our argument has sublinear communication complexity. When shuffling N ciphertexts, arranged in an $m \times n$ matrix, our argument transmits $O(m + n)$ group elements giving a minimal communication complexity of $O(\sqrt{N})$ if we choose $m = n$. In comparison, Groth and Ishai's argument [GI08] communicates $\Omega(m^2 + n)$ group elements and all other state of the art shuffle arguments communicate $\Theta(N)$ elements.

The disadvantage of Groth and Ishai's argument compared to the schemes with linear communication complexity was that the prover's computational complexity was on the order of $O(Nm)$ exponentiations. It was therefore only possible to choose small m. In comparison, our prover's computational complexity is $O(N \log m)$ exponentiations for constant round arguments and $O(N)$ exponentiations if we allow a logarithmic number of rounds. In practice, we do not need to increase the round complexity until m gets quite large, so the improvement in the prover's computational speed is significant compared to Groth and Ishai's work and is comparable to the complexity seen in arguments with linear communication complexity. Moreover, the verifier is fast in our argument making the entire process very light from the verifier's point of view.

8.1.1 Techniques

Groth [Gro09] proposed efficient sublinear size arguments to be used in connection with linear algebra over a finite field. We combine these techniques with Groth and Ishai's sublinear size shuffle argument. The main problem in applying Groth's techniques to shuffling is that they were designed for use in finite fields and not for use with group elements or ciphertexts. It turns out though that the operations are mostly linear; therefore, it is possible to carry them out "in the exponent", somewhat similar to what is often done in threshold cryptography. Using this adaptation we are able to construct an efficient multi-exponentiation argument that a ciphertext C is the product of a set of known ciphertexts $C_1, \ldots, C_N$ raised to a set of hidden committed values $a_1, \ldots, a_N$. This is the main bottleneck in our shuffle argument; therefore, gives us a significant performance improvement.

Groth's sublinear size zero-knowledge arguments also suffered from a performance bottleneck in the prover's computation. At some juncture it is necessary to compute the sums of the diagonal strips in a product of two matrices. This problem is made even worse in our setting because when working with group elements we have to compute these sums in the exponents. By adapting techniques for polynomial multiplication such as Toom-Cook [Too00, Coo66] and the Fast Fourier Transform [CT65] we are able to reduce this computation.

8.1.2 Former Work

The idea of a shuffle was introduced by Chaum [Cha81] but he didn't give any method to guarantee the correctness. In fact this construction was broken by Pfitzmann and Pfitzmann [PP90]. Park et al. [PIK94] were the first to consider ElGamal encryption for mixing. Since then many suggestions have been made

how to build mix-nets or prove the correctness of a shuffle, many of the suggested approaches have been partially or fully broken [JJ01, JJR02], and the remaining schemes sometimes suffer from other drawbacks. The scheme of Desmedt and Kurosawa [DK00] assumed that only a small number of mix-servers are corrupt. Peng et al. [PBDV04] restricted the class of possible permutations and also required that a part of the senders are honest. None of these drawbacks are suffered by the shuffle scheme of Wikström [Wik02] and approaches based on zero-knowledge arguments. Since zero-knowledge arguments achieve better efficiency they will be the focus of our work.

An early contribution using zero-knowledge arguments were made by Sako and Killian [SK95] achieving an universal verifiable protocol. Michels and Horster [MH96] showed that in order to achieve this property each server leaks some information and this leak makes the whole scheme insecure and attackable. Even after Abe [Abe98] fixed the scheme the protocol is known as insecure. Another downside of this method was its high computation and communication cost.

Abe suggested in [Abe98] a version of a mix-net with lower computation cost. In 1999 Abe [Abe99] published a construction of a mix-net based on the permutation network of [Wak68] and later he fixed together with Hoshino [AH01] a security gap.

Furukawa and Sako took in [FS01] a completely different approach to construct and prove the correctness of a shuffle. Their approach was based on permutation matrices and was refined further by Furukawa [Fur05], and Groth and Lu [GL07]. Furukawa, Miyachi, Mori, Obana, and Sako [FMM$^+$02] presented an implementation of a shuffle argument based on permutation matrices and tested it on mix-nets handling 100 000 ElGamal ciphertexts. Recently, Furukawa and Sako [FMS10] have reported on another implementation based on elliptic curve groups.

Wikström [Wik09] also used the idea of permutation matrices and suggested a shuffle argument which splits in an offline and online phase. Furthermore, Terelius and Wikström [TW10] constructed conceptually simple shuffle arguments that allowed the restriction of the shuffles to certain classes of permutations. Both protocols are implemented in the Verificatum mix-net library [Wik10].

The contributions based on the permutation matrices and the followings by Neff were the first ones based on ElGamal encryption and have a complexity that depends linearly on the number of ciphertexts.

Neff [Nef01] published another way to construct a proof of shuffling of ciphertexts, his main idea is the invariance of polynomials $P(X) = \prod_{i=1}^{N}(m_i - X)$ under permutation of roots.

Using the same paradigm Groth [Gro10] published a version of a proof protocol for shuffling ciphertexts. Stamer [Sta05] reported on an implementation of this scheme. Later Groth and Ishai [GI08] proposed the first shuffle argument where the communication complexity is sublinear in the number of ciphertexts.

de Hoogh et al. [dHSSV09] solutions are based on similar ideas to [Nef01], but they only consider rotations instead of general permutations. One of their shuffle arguments is based on the discrete Fourier transform, but the approach is completely different to our use of the FFT. Furthermore, both proposed solutions have only linear complexity.

However, all of the protocols above have linear computation and communication cost, besides the

work of Groth and Ishai, which has sublinear communication cost at the cost of higher computation time.

But efficiency is a major concern in arguments for the correctness of a shuffle. In large elections it is realistic to end up shuffling millions of votes. This places considerable strain on the performance of the zero-knowledge argument both in terms of communication and computation complexity. We will construct an honest verifier zero-knowledge argument for correctness of a shuffle that is highly efficient in terms of communication as well as computation complexity.

8.2 Shuffle Argument

We will give an argument of knowledge of a permutation $\pi \in \Sigma_N$ and randomness $\{\rho_i\}_{i=1}^N$ such that for given ciphertexts $\{C_i\}_{i=1}^N, \{C'_i\}_{i=1}^N$ we have $C'_i = C_{\pi(i)}\mathcal{E}_{pk}(1;\rho_i)$. The shuffle argument combines a multi-exponentiation argument, which allows us to prove that the product of a set of ciphertexts raised to a set of committed exponents yields a particular ciphertext and a product argument, which allows us to prove that a set of committed values has a particular product. The multi-exponentiation argument is given in Section 8.3 and the product argument is given in Section 5.3. In this section, we will give an overview of the protocol and explain how a multi-exponentiation argument can be combined with a product argument, see Section 5.3, to yield an argument for the correctness of a shuffle.

The first step for the prover is to commit to the permutation. This is done by committing to $\pi(1), \ldots, \pi(N)$. The prover will now receive a challenge x and commit to $x^{\pi(1)}, \ldots, x^{\pi(N)}$. The prover will give an argument of knowledge of openings of the commitments to permutations of respectively $1, \ldots, N$ and $x^1, \ldots, x^N$ and demonstrate that the same permutation has been used in both cases. This means the prover has a commitment to $x^1, \ldots, x^N$ permuted in an order that was fixed before the prover saw x.

To check that the same permutation has been used in both commitments the verifier sends random challenges y and z. By using the homomorphic properties of the commitment scheme the prover can in a verifiable manner compute commitments to

$$d_1 - z = y\pi(1) + x^{\pi(1)} - z, \quad \ldots, \quad d_N - z = y\pi(N) + x^{\pi(N)} - z.$$

Using the product argument from Section 5.3 the prover shows that

$$\prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(yi + x^i - z).$$

Observe that we have two identical degree N polynomials in z since the only difference is that the roots have been permuted. The verifier does not know a priori that the two polynomials are identical but can by the Schwartz-Zippel lemma deduce that the prover has negligible chance over the choice of z of making a convincing argument unless indeed there is a permutation π such that

$$d_1 = y\pi(1) + x^{\pi(1)}, \quad \ldots, \quad d_N = y\pi(N) + x^{\pi(N)}.$$

Furthermore, there is negligible probability over the choice of y of this being true unless the first commitment contains $\pi(1), \ldots, \pi(N)$ and the second commitment contains $x^{\pi(1)}, \ldots, x^{\pi(N)}$.

The prover now has commitments to $x^{\pi(1)}, \ldots, x^{\pi(N)}$ and uses the multi-exponentiation argument from Section 8.3 to demonstrate that there exists a ρ such that

$$\prod_{i=1}^{N} C_i^{x^i} = \mathcal{E}_{pk}(1;\rho) \prod_{i=1}^{N} (C'_i)^{x^{\pi(i)}}.$$

The verifier does not see the committed values and therefore does not learn what the permutation is. However, from the homomorphic properties of the encryption scheme the verifier can deduce

$$\prod_{i=1}^{N} M_i^{x^i} = \prod_{i=1}^{N} (M'_i)^{x^{\pi(i)}}$$

for some permutation π that was chosen before the challenge x was sent to the prover. Taking discrete logarithms we have the polynomial identity

$$\sum_{i=1}^{N} \log(M_i) x^i = \sum_{i=1}^{N} \log(M'_{\pi^{-1}(i)}) x^i.$$

There is negligible probability over the choice of x of this equality holding true unless $M'_1 = M_{\pi(1)}, \ldots, M'_N = M_{\pi(N)}$. This shows that we have a correct shuffle.

Statement: $\{\mathbb{G}, p, q\},\ pk, ck,\ \boldsymbol{C}, \boldsymbol{C'} \in \mathbb{H}^N$ with $N = mn$.

Prover's witness: $\pi \in \Sigma_N$ and $\boldsymbol{\rho} \in \mathbb{Z}_q^N$ such that $\boldsymbol{C'} = \mathcal{E}_{pk}(\mathbf{1}; \boldsymbol{\rho}) \boldsymbol{C}_\pi$.

Initial message: Compute

1. $\boldsymbol{a} = \left\{\pi(i)\right\}_{i=1}^{N}$

2. $\boldsymbol{c}_A = \mathrm{com}_{ck}(\boldsymbol{a}; \boldsymbol{r})$, where $\boldsymbol{r} \leftarrow \mathbb{Z}_q^m$.

Send $\boldsymbol{c}_A$

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer Compute

1. $\boldsymbol{b} = \{x^{\pi(i)}\}_{i=1}^{N}$

2. $\boldsymbol{c}_B = \mathrm{com}_{ck}(\boldsymbol{b}; \boldsymbol{s})$, where $\boldsymbol{s} \in \mathbb{Z}_q^m$.

Send $\boldsymbol{c}_B$

Challenge: $y, z \leftarrow \mathbb{Z}_q^*$

Answer: 1. Define $\boldsymbol{c}_{-z} = \mathrm{com}_{ck}(-z, \ldots, -z; \mathbf{0})$ and $\boldsymbol{c}_D = \boldsymbol{c}_A^{\,y}\boldsymbol{c}_B$. Compute openings $\boldsymbol{d} = y\boldsymbol{a} + \boldsymbol{b}$ and $\boldsymbol{t} = y\boldsymbol{r} + \boldsymbol{s}$ of c_D, and engage in a product argument as described in Section 5.3 of openings $d_1 - z, \ldots, d_N - z$ and $\boldsymbol{t}$ such that

$$\boldsymbol{c}_D\boldsymbol{c}_{-z} = \mathrm{com}_{ck}(\boldsymbol{d} - \boldsymbol{z}; \boldsymbol{t}\,) \qquad \text{and} \qquad \prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(yi + x^i - z).$$

2. Compute $\rho = -\boldsymbol{\rho} \cdot \boldsymbol{b}$ and set $\boldsymbol{x} = (x, x^2, \ldots, x^N)^T$. Engage in a multi-exponentiation argument as described in Section 8.3 of $\boldsymbol{b}$, $\boldsymbol{s}$ and ρ such that

$$\boldsymbol{C}^{\boldsymbol{x}} = \mathcal{E}_{pk}(1; \rho)\boldsymbol{C}'^{\boldsymbol{b}} \qquad \text{and} \qquad \boldsymbol{c}_B = \mathrm{com}_{ck}(\boldsymbol{b}; \boldsymbol{s})$$

The two arguments can be run in parallel. Furthermore, the multi-exponentiation argument can be started already in round 3 after the computation of the commitments $\boldsymbol{c}_B$.

Verification: Compute $\prod_{i=1}^{N}(yi + x^i - z)$, $\boldsymbol{C}^{\boldsymbol{x}}$, and compute $\boldsymbol{c}_{-z}, \boldsymbol{c}_D$ as described above.

The verifier accepts if and only if

1. $\boldsymbol{c}_A, \boldsymbol{c}_B \in \mathbb{G}^m$
2. The product argument is valid.
3. The multi-exponentiation argument is valid.

Theorem 32. *The protocol is a public coin perfect generalized $\Sigma-$protocol of $\pi \in \Sigma_N$ and $\boldsymbol{\rho} \in \mathbb{Z}_q^N$ such that $\boldsymbol{C}' = \mathcal{E}_{pk}(\mathbf{1}; \boldsymbol{\rho})\boldsymbol{C}_\pi$.*

Proof. Let us first argue that we have perfect completeness. Note that $d_i = y\pi(i) + x^{\pi(i)}$ so we have

$$\prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(y\pi(i) + x^{\pi(i)} - z) = \prod_{i=1}^{N}(yi + x^i - z).$$

Also, we have with $C'_i = \mathcal{E}_{pk}(1; \rho_i), b_i = x^{\pi(i)}$ and $\rho = -\boldsymbol{\rho} \cdot \boldsymbol{b}$ that

$$\mathcal{E}_{pk}(1; \rho)\boldsymbol{C}'^{\boldsymbol{b}} = \mathcal{E}_{pk}(\mathbf{1}; -\boldsymbol{\rho})^{\boldsymbol{b}}\,(\mathcal{E}_{pk}(\mathbf{1}, \boldsymbol{\rho})\boldsymbol{C}_\pi)^{\boldsymbol{b}} = \boldsymbol{C}_\pi^{\boldsymbol{b}} = \prod_{i=1}^{N} C_{\pi(i)}^{x^{\pi(i)}} = \boldsymbol{C}^{\boldsymbol{x}}.$$

Perfect completeness now follows from the perfect completeness of the product argument and the perfect completeness of the multi-exponentiation argument.

Perfect SHVZK follows from the fact that the commitments are perfectly hiding and the underlying arguments are perfect SHVZK. To simulate the entire argument we can pick random commitments $\boldsymbol{c}_A, \boldsymbol{c}_B \leftarrow \mathrm{com}_{ck}(0, \ldots, 0)$ which can be done without knowing the witness for the correctness of the shuffle. The simulator then runs perfect SHVZK simulations of the product and multi-exponentiation arguments.

Finally, we have to show that we have perfect generalized special soundness. Given accepting transcripts with different challenges x, y, z, the extractor runs the extractors of the underlying product and multi-exponentiation argument to get openings.

We will now argue that with overwhelming probability the extracted openings of $\boldsymbol{c}_A$ must be of the form $\boldsymbol{a} = \{\pi(i)\}_{i=1}^N$ for some permutation $\pi \in \Sigma_N$. Consider the situation after round 3 where the prover has sent $\boldsymbol{c}_B$. The extractor of the multi-exponentiation argument gives us the opening $\boldsymbol{b}, \boldsymbol{s}$ of $\boldsymbol{c}_B$ showing that the opening of $\boldsymbol{c}_D$ must be of the form $\boldsymbol{d} = y\boldsymbol{a} + \boldsymbol{b}$. The product argument shows that

$$\prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(yi + x^i - z).$$

This has negligible probability over z in succeeding unless there exists a permutation π such that $d_i = y\pi(i) + x^{\pi(i)}$. This shows $ya_i + b_i = y\pi(i) + x^{\pi(i)}$, which has negligible probability over y of being true unless $a_i = \pi(i)$. Furthermore, we see that $b_i = x^{\pi(i)}$.

Each choice x_j gives us a witness containing $\rho^{(j)}$ such that

$$\boldsymbol{C}^{\boldsymbol{x}_j} = \mathcal{E}_{pk}(1; \rho^{(j)}) \prod_{i=1}^{N}(C'_i)^{x_j^{\pi(i)}} = \mathcal{E}_{pk}(1; \rho^{(j)}) \boldsymbol{C'}_{\pi^{-1}}^{\boldsymbol{x}_j}.$$

The $N \times N$ matrix

$$X = \begin{pmatrix} x_1^1 & \dots & x_N^1 \\ \vdots & & \vdots \\ x_1^N & \dots & x_N^N \end{pmatrix}$$

can be viewed as a submatrix of a transposed Vandermonde matrix. If $x_1, \dots, x_N$ are different then X is invertible. We now have

$$\boldsymbol{C} = (\boldsymbol{C}^X)^{X^{-1}} = \left(\mathcal{E}_{pk}(\boldsymbol{1}; \boldsymbol{\rho}) \boldsymbol{C'}_{\pi^{-1}}^{X}\right)^{X^{-1}} = \mathcal{E}_{pk}(\boldsymbol{1}; \boldsymbol{\rho} X^{-1}) \boldsymbol{C'}_{\pi^{-1}}.$$

This gives us a permutation π and $\boldsymbol{\rho}\,' = (-\boldsymbol{\rho} X^{-1})_\pi$ such that $\boldsymbol{C'} = \mathcal{E}_{pk}(\boldsymbol{1}; \boldsymbol{\rho}\,')\boldsymbol{C}_\pi$. All the extracted witnesses are the witnesses known by the prover. Otherwise, the prover could use the extractor to break the commitment scheme and find a second opening for one of the commitments. □

Efficiency: The argument consists of m commitments plus the cost of the product argument and the multi-exponentiation argument. Therefore, it consists in total of $5n$ field elements, $7m$ commitments and $2m$ ciphertexts. The last two terms are equal to a cost of $11m$ group elements if $\mathbb{H} = \mathbb{G} \times \mathbb{G}$. Thus, the cost is $O(m + n)$.

The prover has to calculate to $2N$ exponentiations to commit to $\boldsymbol{a}$ and $\boldsymbol{b}$; together with the underlying protocols the computation cost is $O(N \log m)$ exponentiations in $\mathbb{G}$, since the multi-exponentiation technique yields the most exponentiations. Allowing extra interaction the cost of the prover can be reduced to $O(N)$ exponentiations in $\mathbb{G}$. Furthermore, the prover has to calculate $O(N \log m)$ multiplications in $\mathbb{Z}_q$, this cost arises mainly from the product argument.

The verifier has to calculate $12m + 8n$ exponentiations plus the cost to calculate $\boldsymbol{C}^{\boldsymbol{x}}$ and C in the underlying multi-exponentiation argument, which costs $2N$ exponentiations in $\mathbb{H}$. Thus, the verifier's computation cost is $O(N)$ exponentiations in $\mathbb{G}$, if $\mathbb{H} = \mathbb{G} \times \mathbb{G}$. and the cost to calculate $N + 15m + 9n$ multiplications in $\mathbb{Z}_q$.

Both parties can use multi-exponentiations techniques, see Section 4.3, to reduce the computation burden or gain performance speed up from batch verification [BGR98, Gro10]. Thus, the values are quite conservative counting only single exponentiations.

Example: Let $\mathbb{G} = \langle G \rangle = \langle 46 \rangle \subset \mathbb{Z}^*_{179}$, with prime order $|q| = 89$. The public keys and group description are $pk = 74$ and $\{\mathbb{G}, p, q\} = \{\langle 46 \rangle, 179, 89\}$, $ck = \{G_1, \ldots, G_4, H\} = \{G_1, \ldots, G_4, H\} = \{46, 177, 161, 100, 72\}$. The input ciphertexts are

$$\boldsymbol{C} = \begin{pmatrix} (76, 141) & (47, 89) & (146, 36) & (149, 59) \\ (161, 47) & (31, 14) & (59, 145) & (93, 43) \\ (13, 5) & (141, 108) & (80, 149) & (22, 60) \end{pmatrix}$$

and the output ciphertexts

$$\boldsymbol{C}' = \begin{pmatrix} (20, 66) & (75, 149) & (144, 57) & (42, 88) \\ (15, 9) & (51, 64) & (25, 15) & (59, 138) \\ (158, 29) & (61, 142) & (116, 88) & (106, 65) \end{pmatrix}.$$

The prover knows permutation $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 5 & 12 & 7 & 10 & 9 & 6 & 2 & 8 & 1 & 11 & 3 & 4 \end{pmatrix}$ and random elements

$$\boldsymbol{\rho} = (35,\ 35,\ 86,\ 11,\ 29,\ 25,\ 77,\ 45,\ 20,\ 48,\ 58,\ 73)^T$$

such that $\boldsymbol{C}' = \mathcal{E}_{pk}(1; \boldsymbol{\rho})\boldsymbol{C}_\pi$.

The prover first sets $\boldsymbol{a} = (5,\ 12,\ 7,\ 10,\ 9,\ 6,\ 2,\ 8,\ 1,\ 11,\ 3,\ 4)^T$, picks $r_A = (64, 88, 65)^T$, and commits themselves to $\boldsymbol{a}$ in

$$\boldsymbol{c_a} = (16,\ 15,\ 116)^T.$$

The verifier picks random challenge $x = 48$ and the prover calculates

$$\boldsymbol{b} = x^{\pi(i)} = (83,\ 85,\ 60,\ 36,\ 23,\ 68,\ 79,\ 32,\ 48,\ 37,\ 54,\ 11)^T,$$

picks $r_b = (47,\ 34,\ 15)^T$ and computes

$$\boldsymbol{c_b} = (48,\ 61,\ 19)^T.$$

Now, the verifier answers with random challenge $z = 51$ and $y = 24$. Both parties define

$\boldsymbol{z} = (-z, \ldots, -z)$, compute

$$c_{-z} = \mathrm{com}_{ck}(\boldsymbol{z}; 0) = 95 \qquad \boldsymbol{c}_D = \boldsymbol{c}_A^{\,y}\boldsymbol{c}_B = (121,\ 42,\ 67)^T$$

and engage in a product argument of openings $\boldsymbol{d} + \boldsymbol{z}$ and $\boldsymbol{t}$ such that

$$\boldsymbol{c}_D\boldsymbol{c}_{-z} = \mathrm{com}_{ck}(\boldsymbol{d} + \boldsymbol{z}; \boldsymbol{t}\,) = (22,\ 89,\ 81)^T \qquad \text{and} \qquad \prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(yi + x^i - z) = 10.$$

The prover can calculate the openings of $\boldsymbol{c}_D$ themselves as

$$\boldsymbol{d} = y\boldsymbol{a} + \boldsymbol{b} = (31,\ 21,\ 79,\ 62,\ 38,\ 55,\ 48,\ 14,\ 24,\ 86,\ 72,\ 7)^T$$

$$\boldsymbol{t} = y\boldsymbol{r} + \boldsymbol{s} = (42,\ 33,\ 80)^T,$$

and engage in a product argument.

The prover computes $\rho = -\boldsymbol{\rho} \cdot \boldsymbol{b} = 38$, sets

$$\boldsymbol{x} = (48,\ 79,\ 54,\ 11,\ 83,\ 68,\ 60,\ 32,\ 23,\ 36,\ 37,\ 85)^T.$$

Then both parties engage in a multi-exponentiation argument of $\boldsymbol{b}$, $\boldsymbol{s}$ and ρ such that

$$\boldsymbol{C}^{\boldsymbol{x}} = \mathcal{E}_{pk}(1; \rho)\boldsymbol{C'}^{\boldsymbol{b}} \qquad \text{and} \qquad \boldsymbol{c}_B = \mathrm{com}_{ck}(\boldsymbol{b}; \boldsymbol{s}).$$

The verifier accepts if $\boldsymbol{c}_A, \boldsymbol{c}_B \in \mathbb{G}^m$ and the both underlying arguments are valid.

8.3 Multi-exponentiation Argument

Given ciphertexts $C_{11}, \ldots, C_{mn}$ and C we will in this section give an argument of knowledge of openings of commitments $\boldsymbol{c}_A$ to $A = \{a_{ij}\}_{i,j=1}^{n,m}$ such that

$$C = \mathcal{E}_{pk}(1; \rho)\prod_{i=1}^{m}\boldsymbol{c}_i^{\,\boldsymbol{a}_i} \qquad \text{and} \qquad \boldsymbol{c}_A = \mathrm{com}_{ck}(A; \boldsymbol{r})$$

where $\boldsymbol{c}_i = (C_{i1}, \ldots, C_{in})$ and $\boldsymbol{a}_j = (a_{1j}, \ldots, a_{nj})^T$.

To explain the idea in the protocol let us for simplicity assume $\rho = 0$ and the prover knows the openings of $\boldsymbol{c}_A$, and leave the question of SHVZK for later. In other words, we will for now just explain how to convince the verifier in a communication-efficient manner that $C = \prod_{i=1}^{m}\boldsymbol{c}_i^{\,\boldsymbol{a}_i}$. The prover can calculate the ciphertexts

$$E_k = \prod_{\substack{1 \le i,j \le m \\ j = (k-m)+i}} \boldsymbol{c}_i^{\,\boldsymbol{a}_j},$$

where $E_m = C$. To visualize this consider the following matrix

$$\begin{array}{c} \\ \begin{pmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \\ \vdots \\ \boldsymbol{c}_m \end{pmatrix} \end{array} \begin{array}{c} \begin{pmatrix} \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_m \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{c}_1^{\boldsymbol{a}_1} & \ddots & \boldsymbol{c}_1^{\boldsymbol{a}_m} \\ \boldsymbol{c}_2^{\boldsymbol{a}_1} & \ddots & \boldsymbol{c}_2^{\boldsymbol{a}_m} \\ \ddots & \ddots \quad \ddots & \ddots \\ \boldsymbol{c}_m^{\boldsymbol{a}_1} & \ddots & \boldsymbol{c}_m^{\boldsymbol{a}_m} \end{pmatrix} \\ E_1 \quad \cdots \quad E_{m-1} \end{array} \begin{array}{c} \\ \begin{matrix} E_{2m-1} \\ \vdots \\ E_{m+1} \end{matrix} \\ E_m \end{array}$$

The prover sends the ciphertexts $E_1, \ldots, E_{2m-1}$ to the verifier. The ciphertext $C = E_m$ is the product of the main diagonal and the other E_k's are the products of the other diagonals. The prover will use a batch-proof to simultaneously convince the verifier that all the diagonal products give their corresponding E_k.

The verifier selects a challenge $x \leftarrow \mathbb{Z}_q^*$ at random. The prover sets

$$\boldsymbol{x} = (x, x^2, \ldots, x^m)^T,$$

opens $\boldsymbol{c}_A^{\boldsymbol{x}}$ to $\boldsymbol{a} = \sum_{j=1}^m x^j \boldsymbol{a}_j$, and the verifier checks

$$C^{x^m} \prod_{\substack{k=1 \\ k \neq m}}^{2m-1} E_k^{x^k} = \prod_{i=1}^m \boldsymbol{c}_i^{(x^{m-i}\boldsymbol{a})}.$$

Since x is chosen at random, the prover has negligible probability of convincing the verifier unless the x^k-related terms match on each side of the equality for all k. In particular, since $\boldsymbol{a} = \sum_{j=1}^m x^j \boldsymbol{a}_j$ the x^m-related terms give us

$$C^{x^m} = \prod_{i=1}^m \boldsymbol{c}_i^{\,x^{m-i} \sum_{\substack{1 \leq j \leq m \\ m=m-i+j}} x^j \boldsymbol{a}_j} = \left(\prod_{i=1}^m \boldsymbol{c}_i^{\boldsymbol{a}_i} \right)^{x^m}$$

and allow the verifier to conclude $C = \prod_{i=1}^m \boldsymbol{c}_i^{\boldsymbol{a}_i}$.

Finally, to make the argument honest verifier zero-knowledge we have to avoid leaking information about the exponent vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$. The prover therefore calculates the commitment to a random vector $\boldsymbol{a}_0 \leftarrow \mathbb{Z}_q^n$ and after seeing the challenge x they reveal $\boldsymbol{a} = \boldsymbol{a}_0 + \sum_{j=1}^m x^j \boldsymbol{a}_j$. Since $\boldsymbol{a}_0$ is chosen at random this vector does not leak any information about the exponents.

Another possible source of leakage is the products of the diagonals. The prover will therefore randomize each E_k by multiplying it with a random ciphertext $\mathcal{E}_{pk}(G^{b_k}; \tau_k)$. Now each E_k is a uniformly random group element in $\mathbb{H}$; thus, it will not leak information about the exponents. Of course, this would make it possible to encrypt anything in the E_k and allow cheating. To get around this problem the prover has to commit to the b_k's used in the random encryptions and the verifier will check that the prover uses $b_m = 0$. The full argument that also covers the case $\rho \neq 0$ can be found below.

Statement: $\{\mathbb{G}, p, q\},\ pk,\ ck,\ \boldsymbol{c}_1, \ldots, \boldsymbol{c}_m \in \mathbb{H}^n\ ,\ C \in \mathbb{H}$ and $\boldsymbol{c}_A \in \mathbb{G}^m$.

Prover's witness: $A = \{\boldsymbol{a}_j\}_{j=1}^m \in \mathbb{Z}_q^{n \times m}$, $\boldsymbol{r} \in \mathbb{Z}_q^m$ and $\rho \in \mathbb{Z}_q$ such that

$$C = \mathcal{E}_{pk}(1; \rho) \prod_{i=1}^{m} \boldsymbol{c}_i^{\boldsymbol{a}_i} \qquad \text{and} \qquad \boldsymbol{c}_A = \mathrm{com}_{ck}(A; \boldsymbol{r}).$$

Initial message: Compute

1. $c_{A_0} = \mathrm{com}_{ck}(\boldsymbol{a}_0; r_0)$, where $\boldsymbol{a}_0 \leftarrow \mathbb{Z}_q^n, r_0 \leftarrow \mathbb{Z}_q$

2. $c_{b_j} = \mathrm{com}_{ck}(b_j; s_j)$ for $j = 0, \ldots, 2m-1$ where $b_j, s_j \leftarrow \mathbb{Z}_q$ and $b_m = 0, s_m = 0$

3. For $k = 0, \ldots, 2m-1$, $\tau_k \leftarrow \mathbb{Z}_q$ and $\tau_m = \rho$

$$E_k = \mathcal{E}_{pk}(G^{b_k}; \tau_k) \cdot \prod_{\substack{i=1, j=0 \\ j=(k-m)+i}}^{m,m} \boldsymbol{c}_i^{\boldsymbol{a}_j}$$

Send: $c_{A_0}, \{c_{b_j}\}_{j=0}^{2m-1}, \{E_k\}_{k=0}^{2m-1}$

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Set $\boldsymbol{x} = (x, x^2, \ldots, x^m)^T$ and compute

1. $\overline{\boldsymbol{a}} = \boldsymbol{a}_0 + A\boldsymbol{x} \qquad \overline{r} = r_0 + \boldsymbol{r} \cdot \boldsymbol{x}$

2. $\overline{b} = b_0 + \sum_{j=1}^{2m-1} b_j x^k \qquad \overline{s} = s_0 + \sum_{k=1}^{2m-1} s_j x^j$

3. $\overline{\tau} = \tau_0 + \sum_{j=1}^{2m-1} \tau_j x^k$

Send: $\overline{\boldsymbol{a}}, \overline{r}, \overline{b}, \overline{s}, \overline{\tau}$.

Verification: Accept if and only if

1. $c_{A_0}, c_{b_0}, \ldots, c_{b_{2m-1}} \in \mathbb{G}$

2. $E_0, \ldots, E_{2m-1} \in \mathbb{H}$

3. $\overline{\boldsymbol{a}} \in \mathbb{Z}_q^n$, and $\overline{r}, \overline{b}, \overline{s}, \overline{\tau} \in \mathbb{Z}_q$

4. $c_{b_m} = \mathrm{com}_{ck}(0; 0)$ and $E_m = C$

5. $c_{A_0}\boldsymbol{c}_A^{\boldsymbol{x}} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}};\overline{r}) \qquad c_{b_0}\prod_{j=1}^{2m-1} c_{b_j}^{x^j} = \mathrm{com}_{ck}(\overline{b};\overline{s})$

6. $E_0\prod_{k=1}^{2m-1} E_k^{x^k} = \mathcal{E}_{pk}(G^{\overline{b}};\overline{\tau})\prod_{i=1}^{m}\boldsymbol{c}_i^{x^{m-i}\overline{\boldsymbol{a}}}$.

Theorem 33. *The protocol above is a public coin perfect generalized $\Sigma-$protocol of openings $\boldsymbol{a}_1,\ldots,\boldsymbol{a}_m,\boldsymbol{r}$ and randomness ρ such that $C = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^{m}\boldsymbol{c}_i^{\boldsymbol{a}_i}$.*

Proof. It follows by direct verification that $E_m = C$ and $c_{A_0}\boldsymbol{c}_A^{\boldsymbol{x}} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}};\overline{r})$ and

$$c_{b_0}\prod_{k=1}^{2m-1} c_{b_k}^{x^k} = \mathrm{com}_{ck}(b;s).$$

Perfect completeness now follows from

$$\begin{aligned}
E_0\prod_{k=1}^{2m-1}E_k^{x^k} &= \prod_{k=0}^{2m-1}\left(\mathcal{E}_{pk}(G^{b_k};\tau_k)\prod_{\substack{i=1,j=0\\ j=(k-m)+i}}^{m,m}\boldsymbol{c}_i^{\boldsymbol{a}_j}\right)^{x^k}\\
&= \mathcal{E}_{pk}\left(G^{\sum_{k=0}^{2m-1}b_kx^k};\sum_{k=0}^{2m-1}\tau_kx^k\right)\prod_{k=0}^{2m-1}\prod_{\substack{i=1,j=0\\ j=(k-m)+i}}^{m,m}\boldsymbol{c}_i^{x^k\boldsymbol{a}_j}\\
&= \mathcal{E}_{pk}(G^b;\tau)\prod_{i=1}^{m}\prod_{j=0}^{m}\boldsymbol{c}_i^{x^{m-i+j}\boldsymbol{a}_j}\\
&= \mathcal{E}_{pk}(G^b;\tau)\prod_{i=1}^{m}\boldsymbol{c}_i^{x^{m-i}\sum_{j=0}^{m}x^j\boldsymbol{a}_j} = \mathcal{E}_{pk}(G^b;\tau)\prod_{i=1}^{m}\boldsymbol{c}_i^{x^{m-i}\boldsymbol{a}}.
\end{aligned}$$

Now we will prove that we have perfect SHVZK. On challenge x the simulator picks $\overline{\boldsymbol{a}} \leftarrow \mathbb{Z}_q^n$ and $\overline{r} \leftarrow \mathbb{Z}_q$ at random and sets $c_{A_0} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}};\overline{r})\boldsymbol{c}_A^{-\boldsymbol{x}}$. The simulator also picks $\overline{b},\overline{s},s_1,\ldots,s_{m-1},s_{m+1},\ldots,s_{2m-1} \leftarrow \mathbb{Z}_q$ and defines $s_m = 0$. It computes commitments $\boldsymbol{c}_{b_1},\ldots,\boldsymbol{c}_{b_{2m-1}}$ as $c_{b_k} = \mathrm{com}_{ck}(0;s_k)$ and $c_{b_0} = \mathrm{com}_{ck}(\overline{b};\overline{s})\cdot\prod_{k=1}^{2m-1}c_{b_k}^{-x^k}$. Finally it picks random ciphertexts $E_1,\ldots,E_{m-1},E_{m+1},\ldots,E_{2m-1} \leftarrow \mathbb{H}$ and $\overline{\tau} \leftarrow \mathbb{Z}_q$, sets $E_m = C$ and computes

$$E_0 = \mathcal{E}_{pk}(G^b;\overline{\tau})\prod_{i=1}^{m}\boldsymbol{c}_i^{(x^{m-i}\boldsymbol{a})}\prod_{k=1}^{2m-1}E_k^{-x^k}.$$

The simulated argument is $c_{A_0},\{c_{b_j}\}_{j=0}^{2m-1},\{E_k\}_{k=0}^{2m-1},x,\overline{\boldsymbol{a}},\overline{r},\overline{b},\overline{s},\overline{\tau}$.

The next step is to prove that the simulation on challenge x is indistinguishable from a real argument with challenge x. The commitment scheme is perfectly hiding so the distribution of the commitments $c_{b_1},\ldots,c_{b_{m-1}},c_{b_{m+1}},\ldots,c_{b_{2m-1}}$ is identical to the distribution we get in a real argument and $c_{b_m} = \mathrm{com}_{ck}(0;0)$ as in a real argument. The ciphertexts $E_1,\ldots,E_{m-1}$, $E_{m+1},\ldots,E_{2m-1}$ are uniformly random as in a real argument and $E_m = C$ as in a real argument. In the real argument the values $\boldsymbol{a}_0,r_0,b_0,s_0$, and τ_0 are picked at random giving us that $\overline{\boldsymbol{a}},\overline{r},\overline{b},\overline{s},\overline{\tau}$ are uniformly random just as in the simulation. So, up to this point we have the same probability distribution in both real arguments and simulated arguments and the remaining parts c_{A_0},c_{b_0},E_0 are now uniquely defined by the

verification equations. It follows that real arguments and simulated arguments have identical probability distributions.

Finally, we will show that the argument has perfect generalized special soundness. Given $2m$ accepting transcripts with no collision among the challenges, the extractor can extract witness as follows. The first $m+1$ transcripts give us a transposed Vandermonde matrix

$$X = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_{m+1} \\ \vdots & \vdots & & \vdots \\ x_1^m & x_2^m & \dots & x_{m+1}^m \end{pmatrix}.$$

Since $x_1, \dots, x_{m+1}$ are different X is invertible.

We now have for each x_ℓ an opening of $\boldsymbol{c}_{A_0}\boldsymbol{c}_A^{\boldsymbol{x}_\ell} = \mathrm{com}_{ck}(\overline{\boldsymbol{a}}^{(\ell)}, \overline{r}^{(\ell)})$. Let A_x be the matrix with columns $\overline{\boldsymbol{a}}^{(1)} \ \dots \ \overline{\boldsymbol{a}}^{\,(m+1)}$ and let $\boldsymbol{r}_x = (\overline{r}^{(1)}, \dots, \overline{r}^{(m+1)})$. We then have

$$(c_{A_0}, \dots, c_{A_m}) = \left((c_{A_0}, \dots, c_{A_m})^X\right)^{X^{-1}} = \mathrm{com}_{ck}(A_x; \boldsymbol{r}_x)^{X^{-1}} = \mathrm{com}_{ck}(A_x X^{-1}; \boldsymbol{r}_x X^{-1}),$$

which gives us openings $\boldsymbol{a}_0, r_0, \dots, \boldsymbol{a}_m, r_m$ of the commitments $c_{A_0}, \dots, c_{A_m}$. In a similar way the extractor can compute openings $b_0, s_0, \dots, b_{2m-1}, s_{2m-1}$ of $c_{B_0}, \dots, c_{B_{2m-1}}$. We now have for $\ell = 1, \dots, 2m$ that $\overline{\boldsymbol{a}}^{(\ell)} = \sum_{j=0}^{m} x_\ell^j \boldsymbol{a}_j$, this means

$$\prod_{i=1}^{m} \boldsymbol{c}_i^{\,x_\ell^{m-i}\overline{\boldsymbol{a}}^{(\ell)}} = \prod_{i=1}^{m} \boldsymbol{c}_i^{\,\sum_{j=0}^{m} x_\ell^{m+j-i}\overline{\boldsymbol{a}}_j} = \prod_{k=0}^{2m-1} \left(\prod_{\substack{j=0 \\ 1 \le m-k+j \le m}}^{m} \boldsymbol{c}_{(m-k)+j}^{\,\overline{\boldsymbol{a}}_j} \right)^{x_\ell^k}.$$

Let $\boldsymbol{E} = (E_0, \dots, E_{2m-1})$ and let Y be the inverse of the $2m \times 2m$ matrix

$$X = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_1^{2m-1} & x_2^{2m-1} & \dots & x_{2m}^{2m-1} \end{pmatrix}$$

and let $\boldsymbol{y}_i$ be the i-th column vector in Y (numbered 0 through $2m-1$). We get for each $i = 0, \ldots, 2m-1$

$$\begin{aligned}
E_i = \boldsymbol{E}^{X\boldsymbol{y}_i} &= \prod_{\ell=1}^{2m}\left(\prod_{k=0}^{2m-1} E_k^{x_\ell^k}\right)^{y_{\ell i}} \\
&= \prod_{\ell=1}^{2m}\left(\mathcal{E}_{pk}\left(G^{b^{(\ell)}};\tau^{(\ell)}\right)\prod_{k=0}^{2m-1}\Big(\prod_{\substack{j=0\\1\le m-k+j\le m}}^{m} \boldsymbol{c}_{(m-k)+j}^{\boldsymbol{a}_j}\Big)^{x_\ell^k}\right)^{y_{\ell i}} \\
&= \mathcal{E}_{pk}\left(G^{\sum_{\ell=1}^{2m} b^{(\ell)} y_{\ell i}};\sum_{\ell=1}^{2m}\tau^{(\ell)} y_{\ell i}\right)\prod_{k=0}^{2m-1}\left(\prod_{\substack{j=0\\1\le m-k+j\le m}}^{m} \boldsymbol{c}_{(m-k)+j}^{\boldsymbol{a}_j}\right)^{\sum_{\ell=1}^{2m} x_\ell^k y_{\ell i}} \\
&= \mathcal{E}_{pk}\left(G^{b_i};\tau_i\right)\prod_{\substack{j=0\\1\le m-i+j\le m}}^{m} \boldsymbol{c}_{(m-i)+j}^{\boldsymbol{a}_j}\ ,
\end{aligned}$$

where $\tau_i = \sum_{\ell=1}^{2m}\tau^{(\ell)} y_{\ell i}$.

The verifier checks $c_{b_m} = \mathrm{com}_{ck}(0;0)$ so the binding property implies that there is negligible probability for the extractor extracting $b_m \neq 0$. Since $E_m = C$ we now have

$$C = \mathcal{E}_{pk}(G^0;\tau_m)\prod_{\substack{j=0\\1\le m-m+j\le m}}^{m} \boldsymbol{c}_{m-m+j}^{\boldsymbol{a}_j} = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^{m}\boldsymbol{c}_i^{\boldsymbol{a}_i}$$

with $\rho = \tau_m$. This shows that we have extracted a valid witness for the statement. This opening is the same as known by the prover. If this is not the case, the prover could run the extractor to find an different opening for at least one of their commitments with non-negligible probability, and the commitment scheme is broken. □

Efficiency: During the protocol the prover sends $2m$ ciphertexts, $2m$ commitments, and n field elements to the verifier; thus, the communication complexity is $O(m+n)$.

The prover has to compute $2m$ commitment which needs $2m$ exponentiations. Furthermore they have to calculate the ciphertexts E_k, naïvely this costs mN exponentiations in $\mathbb{H}$. Using the technique described in the next section, this cost can be brought down to $O(N\log m)$ exponentiations in $\mathbb{G}$. Allowing more rounds of interaction we can even achieve a computation cost of $O(N)$. Both techniques gain from multi-exponentiation techniques, see Section 4.3, to bring the cost of the prover down further. On top off this the prover has to calculate $N + 10m + n$ multiplications.

The verifier has to calculate $7m + 2n$ exponentiations in $\mathbb{G}$, plus N exponentiations in $\mathbb{H}$, so the cost is $O(N)$. He also has to calculate the same number of multiplications. Again the verifier gains from multi-exponentiations techniques, see Section 4.3, or batch verification [BGR98, Gro10].

Example: Let $\mathbb{G} = \langle G\rangle = \langle 46\rangle \subset \mathbb{Z}_{179}^*$, with prime order $|q| = 89$. The public keys and the group description are $pk = 74$ and $\{\mathbb{G}, p, q\} = \{\langle 47\rangle, 179, 89\}$, $ck = \{G_1, \ldots, G_4, H\} =$

$\{46, 177, 161, 100, 72\}$. The ciphertexts

$$\boldsymbol{C}' = \begin{pmatrix} (20,66) & (75,149) & (144,57) & (42,88) \\ (15,9) & (51,64) & (25,15) & (59,138) \\ (158,29) & (61,142) & (116,88) & (106,65) \end{pmatrix},$$

$C = (64, 173)$, and $c_A = (48,\ 61,\ 19)^T$. The prover knows witnesses

$$A = \begin{pmatrix} 83 & 85 & 60 & 36 \\ 23 & 68 & 79 & 32 \\ 48 & 37 & 54 & 11 \end{pmatrix},$$

$\boldsymbol{r} = (67,\ 34,\ 15)^T$, $\rho = 38$ such that

$$C = \mathcal{E}_{pk}(1; \rho) \prod_{i=1}^{m} \boldsymbol{c}_i^{\boldsymbol{a}_i} \qquad \text{and} \qquad \boldsymbol{c}_A = \mathrm{com}_{ck}(\boldsymbol{a}; \boldsymbol{r}).$$

The prover picks random $\boldsymbol{a}_0 = (82,\ 12,\ 54,\ 52)^T \leftarrow \mathbb{Z}_{89}^4$, $r_0 = 47$, computes

$$c_{A_0} = 48,$$

picks

$$\boldsymbol{b} = (17,\ 53,\ 0,\ 56,\ 0)^T \quad \boldsymbol{s} = (76,\ 35,\ 60,\ 84,\ 74)^T$$

and computes the commitments to the values b_k.

$$\boldsymbol{c}_b = (1,\ 29,\ 156,\ 1,\ 67,\ 22)^T.$$

Lastly, the prover calculates the values

$$e_k = \prod_{\substack{i=1,j=0 \\ j=(k-m)+i}}^{m,m} \boldsymbol{c}_i^{\boldsymbol{a}_j},$$

$$\boldsymbol{e} = \big((124,67) \quad (161,156) \quad (83,52) \quad (19,22) \quad (116,161) \quad (12,51)\big),$$

picks $\boldsymbol{\tau} = (13, 14, 53, 83, 70, 31)^T$, and re-encrypts $\boldsymbol{e}$ as follows

$$\begin{aligned} \boldsymbol{E_k} &= \mathcal{E}_{pk}(G^{b_k}, \tau_k) e_k \\ &= \big((20,70) \quad (76,124) \quad (141,49) \quad (64,173) \quad (146,61) \quad (52,151)\big). \end{aligned}$$

The prover sends all values $c_{A_0}, \boldsymbol{c}_b, \boldsymbol{E}$ to the verifier, who answers with challenge $x = 62$. Then the prover calculates answers

$$\overline{\boldsymbol{a}} = (52,\ 46,\ 0,\ 4)^T \quad \overline{r} = 50 \quad \overline{b} = 63 \quad \overline{s} = 23 \quad \overline{\tau} = 19.$$

The verifier, after receiving the answers, checks if all commitments are in $\mathbb{G}$, all ciphertexts are in $\mathbb{H}$ and all answers are valid. He defines $\overline{x} = (x, x^2, x^3, x^4)$ and further checks

$$c_{b_m} = 1 = \mathrm{com}_{ck}(0;0) \qquad (\checkmark) \qquad E_m = (64, 173) = C \qquad (\checkmark)$$

$$c_{A_0}\boldsymbol{c}_A^{\boldsymbol{x}} = 87 = \mathrm{com}_{ck}(\overline{\boldsymbol{a}};\overline{r}) \qquad (\checkmark) \qquad c_{b_0}\prod_{k=1}^{2m-1} c_{b_k}^{x^k} = 107 = \mathrm{com}_{ck}(\overline{b};\overline{s}) \qquad (\checkmark)$$

$$E_0 \prod_{k=1}^{2m-1} E_k^{x^k} = (66, 46) = \mathcal{E}_{pk}(G^{\overline{b}};\overline{\tau}) \prod_{i=1}^{m} \boldsymbol{c}_i^{x^{m-i}\boldsymbol{a}_i}.$$

8.3.1 The prover's computation

The argument we just described has efficient verification and very low communication complexity, but the prover has to compute

$$E_0, \ldots, E_{2m-1}.$$

In this section we will for clarity ignore the randomization needed to get honest verifier zero-knowledge, which can be added in a straightforward manner at little extra computational cost. So, let us say we need to compute for $k = 1, \ldots, 2m-1$ the elements

$$E_k = \prod_{\substack{i=1,j=1 \\ j=(k-m)+i}}^{m,m} \boldsymbol{c}_i^{\boldsymbol{a}_j}.$$

This can be done by first computing the m^2 products $\boldsymbol{c}_i^{\boldsymbol{a}_j}$ and then computing the E_k's as suitable products of some of these values. Since each product $\boldsymbol{c}_i^{\boldsymbol{a}_j}$ is of the form

$$\prod_{\ell=1}^{n} C_{i\ell}^{a_{j\ell}}$$

this gives a total of $m^2 n$ exponentiations in $\mathbb{H}$. For large m this cost is prohibitive.

It turns out that we can do much better by using techniques inspired by multiplication of integers and polynomials, such as Karatsuba [KO63], Toom-Cook [Too00, Coo66], and the Fast Fourier Transform [CT65]. A common theme in these techniques is to compute the coefficients of the product $p(x)q(x)$ of two degree $m-1$ polynomials $p(x)$ and $q(x)$ by evaluating $p(x)q(x)$ in $2m-1$ points $\omega_0, \ldots, \omega_{2m-2}$ and using polynomial interpolation to recover the coefficients of $p(x)q(x)$ from $p(\omega_0)q(\omega_0), \ldots, p(\omega_{2m-2})q(\omega_{2m-2})$.

If we pick $\omega \in \mathbb{Z}_q$ we can evaluate the vectors

$$\prod_{i=1}^{m} \boldsymbol{c}_i^{\omega^{m-i}} \qquad\qquad \sum_{j=1}^{m} \omega^{j-1} \boldsymbol{a}_j.$$

This gives us

$$\left(\prod_{i=1}^{m} \boldsymbol{c}_i^{\omega^{m-i}}\right)^{\sum_{j=1}^{m} \omega^{j-1} \boldsymbol{a}_j} = \prod_{k=1}^{2m-1} \left(\prod_{\substack{i=1,j=1 \\ j=(k-m)+i}}^{m,m} \boldsymbol{c}_i^{\boldsymbol{a}_j}\right)^{\omega^{k-1}} = \prod_{k=1}^{2m-1} E_k^{\omega^{k-1}}.$$

Picking $2m-1$ different $\omega_0, \ldots, \omega_{2m-2} \in \mathbb{Z}_p$ we get the $2m-1$ ciphertexts

$$\prod_{k=1}^{2m-1} E_k^{\omega_0^{k-1}}, \ \ldots, \ \prod_{k=1}^{2m-1} E_k^{\omega_{2m-2}^{k-1}}.$$

The $\omega_0, \ldots, \omega_{2m-2}$ are different and therefore the transposed Vandermonde matrix

$$\begin{pmatrix} 1 & \ldots & 1 \\ \vdots & & \vdots \\ \omega_0^{2m-2} & \ldots & \omega_{2m-2}^{2m-2} \end{pmatrix}$$

is invertible. Let $\boldsymbol{y}_i = (y_0, \ldots, y_{2m-2})^T$ be the i-th column of the inverse matrix. We can now compute E_i as

$$E_i = \prod_{\ell=0}^{2m-2} \left(\prod_{k=1}^{2m-1} E_k^{\omega_\ell^{k-1}}\right)^{y_\ell} = \prod_{\ell=0}^{2m-2} \left(\left(\prod_{i=1}^{m} \boldsymbol{c}_i^{\omega_\ell^{m-i}}\right)^{\sum_{j=1}^{m} \omega_\ell^{j-1} \boldsymbol{a}_j}\right)^{y_\ell}$$

This means the prover can compute $E_1, \ldots, E_{2m-1}$ as linear combinations of

$$\left(\prod_{i=1}^{m} \boldsymbol{c}_i^{\omega_0^{m-i}}\right)^{\sum_{j=1}^{m} \omega_0^{j-1} \boldsymbol{a}_j} \quad \ldots \quad \left(\prod_{i=1}^{m} \boldsymbol{c}_i^{\omega_{2m-2}^{m-i}}\right)^{\sum_{j=1}^{m} \omega_{2m-2}^{j-1} \boldsymbol{a}_j}.$$

The expensive step in this computation is to compute $\prod_{i=1}^{m} \boldsymbol{c}_i^{\omega_0^{m-i}}, \ldots, \prod_{i=1}^{m} \boldsymbol{c}_i^{\omega_{2m-2}^{m-i}}$.

If $2m-2$ is a power of 2 and $2m-2 | q-1$ we can pick $\omega_1, \ldots, \omega_{2m-2}$ as roots of unity, i.e. $\omega_k^{2m-2} = 1$. This allows us to use the Fast Fourier Transformation "in the exponent" to simultaneously calculate $\prod_{i=1}^{m} \boldsymbol{c}_i^{\omega_k^{m-i}}$ in all of the roots of unity using only $O(mn \log m)$ exponentiations. This is asymptotically the fastest technique we know for computing $E_0, \ldots, E_{2m-2}$.

Unfortunately, the FFT is not well suited for being used in combination with multi-exponentiation techniques and in practice it takes a while before the asymptotic behavior kicks in. Therefore, for small m it is useful to consider other strategies. Inspired by the Toom-Cook method for integer multiplication, we may for instance choose $\omega_0, \omega_1, \ldots, \omega_{2m-2}$ to be small integers. When m is small even the largest exponent ω_k^{2m-2} will remain small. For instance, if $m = 4$ we may choose
$\omega_k \in \{0, -1, 1, -2, 2, -3, 3\}$, which makes the largest exponent $\omega_k^{m-1} = 3^3 = 27$. This makes it cheap

to compute each $\prod_{i=1}^{m} \boldsymbol{c}_i^{\omega_k^{m-i}}$ because the exponents are very small.

The basic step of Toom-Cook outlined above can be optimized by choosing the evaluation points carefully. However, the performance degrades quickly as m grows. Using recursion it is possible to get subquadratic complexity also for large m, however, the cost still grows relatively fast. In the next section we will therefore describe an interactive technique for reducing the prover's computation. In our implementation, see Section 8.5, we have used a combination of the interactive technique and Toom-Cook as the two techniques work well together.

8.3.2 Trading computation for interaction

We will present an interactive technique that can be used to reduce the prover's computation. The prover wants to show that C has the same plaintext as the product of the main diagonal of following matrix (here illustrated for $m = 16$). The technique is based on [GI08] interaction technique to reduce the numbers of multiplication needed in their protocol.

$$\begin{pmatrix}
\boldsymbol{c}_1^{\boldsymbol{a}_1} & \boldsymbol{c}_1^{\boldsymbol{a}_2} & \boldsymbol{c}_1^{\boldsymbol{a}_3} & \boldsymbol{c}_1^{\boldsymbol{a}_4} & & & & & & & & \\
\boldsymbol{c}_2^{\boldsymbol{a}_1} & \boldsymbol{c}_2^{\boldsymbol{a}_2} & \boldsymbol{c}_2^{\boldsymbol{a}_3} & \boldsymbol{c}_2^{\boldsymbol{a}_4} & & & & \ddots & & & & \\
\boldsymbol{c}_3^{\boldsymbol{a}_1} & \boldsymbol{c}_3^{\boldsymbol{a}_2} & \boldsymbol{c}_3^{\boldsymbol{a}_3} & \boldsymbol{c}_3^{\boldsymbol{a}_4} & & & & & & & & \\
\boldsymbol{c}_4^{\boldsymbol{a}_1} & \boldsymbol{c}_4^{\boldsymbol{a}_2} & \boldsymbol{c}_4^{\boldsymbol{a}_3} & \boldsymbol{c}_4^{\boldsymbol{a}_4} & & & & & & & & \\
& & & & \ddots & & & & & & & \\
& & & & & \boldsymbol{c}_{13}^{\boldsymbol{a}_{13}} & \boldsymbol{c}_{13}^{\boldsymbol{a}_{14}} & \boldsymbol{c}_{13}^{\boldsymbol{a}_{15}} & \boldsymbol{c}_{13}^{\boldsymbol{a}_{16}} \\
& \ddots & & & & \boldsymbol{c}_{14}^{\boldsymbol{a}_{13}} & \boldsymbol{c}_{14}^{\boldsymbol{a}_{14}} & \boldsymbol{c}_{14}^{\boldsymbol{a}_{15}} & \boldsymbol{c}_{14}^{\boldsymbol{a}_{16}} \\
& & & & & \boldsymbol{c}_{15}^{\boldsymbol{a}_{13}} & \boldsymbol{c}_{15}^{\boldsymbol{a}_{14}} & \boldsymbol{c}_{15}^{\boldsymbol{a}_{15}} & \boldsymbol{c}_{15}^{\boldsymbol{a}_{16}} \\
& & & & & \boldsymbol{c}_{16}^{\boldsymbol{a}_{13}} & \boldsymbol{c}_{16}^{\boldsymbol{a}_{14}} & \boldsymbol{c}_{16}^{\boldsymbol{a}_{15}} & \boldsymbol{c}_{16}^{\boldsymbol{a}_{16}}
\end{pmatrix}$$

In the previous section the prover calculated all m^2 entries of the matrix. But we are only interested in the product along the diagonal so we can save computation by just focusing on the blocks close to the main diagonal.

Let us explain the idea in the case of $m = 16$. We can divide the matrix into 4×4 blocks and only use the four blocks that are on the main diagonal. Suppose the prover wants to demonstrate

$$C = \prod_{i=1}^{16} \boldsymbol{c}_i^{\boldsymbol{a}_i}.$$

Let us for now just focus on soundness and return to the question of honest verifier zero-knowledge later. The prover starts by sending $E_0, E_1, E_2, E_3, E_4, E_5, E_6$ that are the products along the diagonals of the elements in the blocks that we are interested in, i.e.

$$E_0 = \prod_{i=1}^{4} \boldsymbol{c}_{4i}^{\boldsymbol{a}_{4i-3}}, \ldots, E_6 = \prod_{i=1}^{4} \boldsymbol{c}_{4i-3}^{\boldsymbol{a}_{4i}}$$

and $E_3 = C$. The verifier sends a random challenge x and using the homomorphic properties of the en-

cryption scheme and of the commitment scheme both the prover and the verifier can compute $\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_4$ and $c_{A'_1}, \ldots, c_{A'_4}$ as

$$\boldsymbol{c}'_i = \boldsymbol{c}_{4i-3}^{x^3}\boldsymbol{c}_{4i-2}^{x^2}\boldsymbol{c}_{4i-1}^{x}\boldsymbol{c}_{4i} \qquad\qquad c_{A'_j} = c_{A_{4j-3}}c_{A_{4j-2}}^{x}c_{A_{4j-1}}^{x^2}c_{A_{4j}}^{x^3}.$$

They can also both compute $C' = \prod_{k=0}^{6} E_k^{x^k}$. The prover and the verifier now engage in an SHVZK argument for the smaller statement $C' = \prod_{i=1}^{4} \boldsymbol{c}_i'^{\,\boldsymbol{a}'_i}$. The prover can compute a witness for this statement with $\boldsymbol{a}'_i = \boldsymbol{a}_{4i-3} + x\boldsymbol{a}_{4i-2} + x^2\boldsymbol{a}_{4i-1} + x^3\boldsymbol{a}_{4i}$. This shows

$$C^{x^3}\prod_{\substack{k=0\\k\neq 3}}^{6} E_k^{x^k} = \prod_{i=1}^{4}(\boldsymbol{c}_{4i-3}^{x^3}\boldsymbol{c}_{4i-2}^{x^2}\boldsymbol{c}_{4i-1}^{x}\boldsymbol{c}_{4i})^{\,(\boldsymbol{a}_{4i-3}+x\boldsymbol{a}_{4i-2}+x^2\boldsymbol{a}_{4i-1}+x^3\boldsymbol{a}_{4i})}.$$

Looking at the x^3-related terms, we see this has negligible chance of holding for a random x unless $C = \prod_{i=1}^{16} \boldsymbol{c}_i^{\boldsymbol{a}_i}$, which is what the prover wanted to demonstrate.

We will generalize the technique to reducing a statement $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m, C, c_{A_1}, \ldots, c_{A_m}$ with a factor μ to a statement $\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_{m'}, C', c_{A'_1}, \ldots, c_{A'_{m'}}$ where $m = \mu m'$. To add honest verifier zero-knowledge to the protocol, we have to prevent the E_k's from leaking information about $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$. We do this by randomizing each E_k with a random ciphertext $\mathcal{E}_{pk}(G^{b_k}; t_k)$. To prevent the prover using the randomization to cheat they will have to commit themselves to the b_k's before seeing the challenge x.

Statement: $\{\mathbb{G}, p, q\},\ pk,\ ck,\ \boldsymbol{c}_1, \ldots, \boldsymbol{c}_m \in \mathbb{H}^n,\ C \in \mathbb{H}$ and $c_{A_1}, \ldots, c_{A_m} \in \mathbb{G}$ where $m = \mu m'$.

Prover's witness: $A \in \mathbb{Z}_q^{n\times m}, \boldsymbol{r} \in \mathbb{Z}_q^m$ and $\rho \in \mathbb{Z}_q$ such that

$$C = \mathcal{E}_{pk}(1; \rho)\prod_{i=1}^{m} \boldsymbol{c}_i^{\boldsymbol{a}_i} \qquad \text{and} \qquad \boldsymbol{c}_A = \mathrm{com}_{ck}(A; \boldsymbol{r}).$$

Initial message: Compute

1. $\boldsymbol{c}_b = \mathrm{com}_{ck}(\boldsymbol{b}; \boldsymbol{s})$, where $\boldsymbol{b} = (b_0, \ldots, b_{2\mu-2}), \boldsymbol{s} \leftarrow \mathbb{Z}_q^{2\mu-1}$, and $b_{\mu-1} = s_{\mu-1} = 0$

2. For $k = 0, \ldots, 2\mu - 1$, $\tau_k \leftarrow \mathbb{Z}_q$ and $\tau_{\mu-1} = \rho$

$$E_k = \mathcal{E}_{pk}(G^{b_k}; \tau_k)\prod_{\ell=0}^{m'-1}\prod_{\substack{i=1,j=1\\ j=(k+1-\mu)+i}}^{\mu,\mu} \boldsymbol{c}_{\mu\ell+i}^{\boldsymbol{a}_{\mu\ell+j}}$$

Send: $\boldsymbol{c}_b, \boldsymbol{E} = (E_0, \ldots, E_{2\mu-2})$.

Challenge: $x \leftarrow \mathbb{Z}_q^*$

Answer: Set $\boldsymbol{x} = (1, x, \ldots, x^{2\mu-2})^T$ and compute

1. $b = \boldsymbol{b} \cdot \boldsymbol{x} \qquad s = \boldsymbol{s} \cdot \boldsymbol{x}$

2. $\boldsymbol{a}'_\ell = \sum_{j=1}^{\mu} x^{j-1}\boldsymbol{a}_{\mu(\ell-1)+j} \qquad r'_\ell = \sum_{j=1}^{\mu} x^{j-1} r_{\mu(\ell-1)+j}$ for $\ell = 1, \ldots, m'$

3. $\rho' = \boldsymbol{\tau} \cdot \boldsymbol{x}$

4. $\boldsymbol{c}'_\ell = \prod_{i=1}^{\mu} \boldsymbol{c}_{\mu(\ell-1)+i}^{x^{\mu-i}} \qquad c_{A'_\ell} = \prod_{j=1}^{\mu} c_{A_{\mu(\ell-1)+j}}^{x^{j-1}}$ for $\ell = 1 \ldots m'$

5. $C' = \mathcal{E}_{pk}(G^{-b}; 0)\boldsymbol{E}^{\boldsymbol{x}}$

Engage in an SHVZK argument of openings $\boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_{m'}, \boldsymbol{r}'$ and ρ' such that

$$C' = \mathcal{E}_{pk}(1; \rho') \prod_{\ell=1}^{m'} \boldsymbol{c}_\ell'^{\,\boldsymbol{a}'_\ell}.$$

Verification: Accept if and only if

1. $\boldsymbol{c}_b \in \mathbb{G}^{2\mu-1}$

2. $E_0, \ldots, E_{2\mu-2} \in \mathbb{H}$

3. $b, s \in \mathbb{Z}_q$

4. $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0; 0) \qquad E_{\mu-1} = C \qquad \boldsymbol{c}_b^{\boldsymbol{x}} = \mathrm{com}_{ck}(b; s)$

5. The SHVZK argument for $\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_{m'}, C', c_{A'_1}, \ldots, c_{A'_{m'}}$ is valid.

Theorem 34. *The protocol above is a public coin perfect generalized $\Sigma-$protocol of $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m, \boldsymbol{r}$ such that $C = \mathcal{E}_{pk}(1; \rho) \prod_{i=1}^{m} \boldsymbol{c}_i^{\boldsymbol{a}_i}$*

Proof. Let us first argue that we have perfect completeness. It follows by straightforward verification that $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0; 0)$, and $C = E_{\mu-1}$, and $\boldsymbol{c}_b^{\boldsymbol{x}} = \mathrm{com}_{ck}(b; s)$. Both the prover and the verifier can compute the reduced statement $\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_{m'}, C', \boldsymbol{c}_{A'}$ as described above and the prover can compute openings of $\boldsymbol{c}_{A'}$ and $\rho' = \boldsymbol{\tau} \cdot \boldsymbol{x}$. Perfect completeness now follows from the perfect completeness of the underlying SHVZK argument because $C' = \mathcal{E}_{pk}(1; \rho') \prod_{i=1}^{m'} \boldsymbol{c}_i'^{\,\boldsymbol{a}'_i}$. To see this is the case, we compute

$$\begin{aligned}
C' &= \mathcal{E}_{pk}(G^{-b};0)\prod_{k=0}^{2\mu-2} E_k^{x^k} \\
&= \mathcal{E}_{pk}(G^{-\boldsymbol{b}\cdot\boldsymbol{x}};0)\prod_{k=0}^{2\mu-2}\left(\mathcal{E}_{pk}(G^{b_k};\tau_k)\prod_{\ell=0}^{m'-1}\prod_{\substack{i=1,j=1\\ j=(k+1-\mu)+i}}^{\mu,\mu}\boldsymbol{c}_{\mu\ell+i}^{\boldsymbol{a}_{\mu\ell+j}}\right)^{x^k} \\
&= \mathcal{E}_{pk}\big(G^0;\boldsymbol{\tau}\cdot\boldsymbol{x}\big)\prod_{k=0}^{2\mu-2}\prod_{\ell=0}^{m'-1}\prod_{\substack{i=1,j=1\\ j=(k+1-\mu)+i}}^{\mu,\mu}\big(\boldsymbol{c}_{\mu\ell+i}^{x^{\mu-i}}\big)^{x^{j-1}\boldsymbol{a}_{\mu\ell+j}} \\
&= \mathcal{E}_{pk}(1;\rho')\prod_{\ell=1}^{m'}\prod_{k=0}^{2\mu-2}\prod_{\substack{i=1,j=1\\ j=(k+1-\mu)+i}}^{\mu,\mu}\big(\boldsymbol{c}_{\mu(\ell-1)+i}^{x^{\mu-i}}\big)^{x^{j-1}\boldsymbol{a}_{\mu(\ell-1)+j}} \\
&= \mathcal{E}_{pk}(1;\rho')\prod_{\ell=1}^{m'}\boldsymbol{c}_\ell'^{\,\boldsymbol{a}'_\ell}
\end{aligned}$$

Let us now describe the SHVZK simulator. On challenge x it picks $c_{b_1},\ldots,c_{b_{2\mu-2}} \leftarrow \mathbb{G}$, $b, s \leftarrow \mathbb{Z}_q$, and $E_0,\ldots,E_{2\mu-2} \leftarrow \mathbb{H}$. It sets $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0;0)$ and $E_{\mu-1} = C$, and computes

$$c_{b_0} = \mathrm{com}_{ck}(b;s)\prod_{j=1}^{2\mu-2} c_{b_j}^{-x^j}.$$

Now it runs the simulator for the SHVZK argument on $\boldsymbol{c}'_1,\ldots,\boldsymbol{c}'_{m'}, C', c_{A'_1},\ldots,c_{A'_{m'}}$.

Both in real arguments and simulated arguments we have uniformly random ciphertexts $E_0,\ldots,E_{\mu-2},E_\mu,\ldots,E_{2\mu-2}$, and $E_{\mu-1}=C$. The commitment scheme is perfectly hiding, so we have that $c_{b_1},\ldots,c_{b_{\mu-2}}$, $c_{b_\mu},\ldots,c_{b_{2\mu-2}}$ are uniformly random in real arguments and $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0;0)$ just as in the simulation. In a real argument we have $b_0, s_0 \leftarrow \mathbb{Z}_q$ chosen at random, which implies that b, s are uniformly random just as in the simulation. Given all these values the verification equation uniquely defines

$$c_{b_0} = \mathrm{com}_{ck}(b;s)\prod_{k=1}^{2\mu-2} c_{b_k}^{-x^k}.$$

It now follows from the perfect SHVZK of the underlying argument that the simulation is perfect.

Now, we have to show that the argument has perfect generalized special soundness.

Suppose now the extractor E has satisfactory answers to $2\mu-1$ challenges $x_1,\ldots,x_{2\mu-1}$, with no collision. The matrix X with columns of the form $\boldsymbol{x}_\ell = (1,\ x_\ell,\ \ldots, x_\ell^{2\mu-2})^T$ is a transposed Vandermonde matrix and as $x_1,\ldots,x_{2\mu-1}$ are different X is invertible. Define $\boldsymbol{b}_x = (b^{(1)},\ \ldots,\ b^{(2\mu-1)})$ and $\boldsymbol{s}_x = (s^{(1)},\ \ldots,\ s^{(2\mu-1)})$. We now have

$$\boldsymbol{c}_b = (\boldsymbol{c}_b^{\,X})^{X^{-1}} = \mathrm{com}_{ck}(\boldsymbol{b}_x;\boldsymbol{s}_x)^{X^{-1}} = \mathrm{com}_{ck}(\boldsymbol{b}_x X^{-1};\boldsymbol{s}_x X^{-1})$$

which gives us openings $b_0,\ldots,b_{2\mu-1},s_0,\ldots,s_{2\mu-1}$ of all the commitments $c_{b_0},\ldots,c_{b_{2\mu-2}}$ the prover sent.

The extractor E for the underlying SHVZK arguments provides us with openings $(\boldsymbol{a}'_j)^{(\ell)}, (r'_j)^{(\ell)}$ of $c_{A'_j} = \prod_{k=1}^{\mu} c_{A_{j\mu+k}}^{x^{j-1}}$. Using a similar technique as we did for the b_i's we can by taking appropriate linear combinations find openings $\boldsymbol{a}_1, r_1, \ldots, \boldsymbol{a}_m, r_m$ of $c_{A_1}, \ldots, c_{A_m}$.

The extractor of the underlying argument give us $\rho'^{(1)}, \ldots, \rho'^{(2\mu-1)}$ in response to the challenges $x_1, \ldots, x_\ell$ satisfying $C'^{(\ell)} = \mathcal{E}_{pk}(1; \rho'^{(\ell)}) \prod_{i=1}^{m'} \left(\boldsymbol{c}_i'^{(\ell)}\right)^{\boldsymbol{a}_i'^{(\ell)}}$. We will now argue that a linear combination of those will give us the desired ρ, needed to complete our argument. Let $\boldsymbol{y} = (y_1, \ldots, y_{2\mu-1})$ be the $(\mu-1)$-th column of X^{-1} such that

$$\sum_{\ell=1}^{2\mu-1} x_\ell^k y_\ell = 1$$

for $k = \mu - 1$, else 0, and define

$$\rho = \sum_{\ell=1}^{2\mu-1} \rho'^{(\ell)} y_\ell.$$

The verifier checks $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0;0)$, so the binding property implies that $b_{\mu-1} = 0$. We then have

$$\begin{aligned}
C &= E_{\mu-1} = \prod_{\ell=1}^{2\mu-1} \left(\prod_{k=0}^{2\mu-2} E_k^{x_\ell^k} \right)^{y_\ell} = \prod_{\ell=1}^{2\mu-1} \left(\mathcal{E}_{pk}(G^{b^{(\ell)}}; 0) C'^{(\ell)} \right)^{y_\ell} \\
&= \prod_{\ell=1}^{2\mu-1} \left(\mathcal{E}_{pk}(G^{b^{(\ell)}}; \rho'^{(\ell)}) \prod_{u=1}^{m'} \left(\boldsymbol{c}_u'^{(\ell)}\right)^{\boldsymbol{a}_u'^{(\ell)}} \right)^{y_\ell} \\
&= \mathcal{E}_{pk}\Big(G^{\sum_{\ell=1}^{2\mu-1} b^{(\ell)} y_\ell}; \sum_{\ell=1}^{2\mu-1} \rho'^{(\ell)} y_\ell \Big) \prod_{\ell=1}^{2\mu-1} \left(\prod_{u=0}^{m'-1} \left(\prod_{i=1}^{\mu} \boldsymbol{c}_{u\mu+i}^{x_\ell^{\mu-i}} \right)^{\sum_{j=1}^{\mu} x_\ell^{j-1} \boldsymbol{a}_{u\mu+j}} \right)^{y_\ell} \\
&= \mathcal{E}_{pk}(G^{b_{\mu-1}}; \rho) \prod_{\ell=1}^{2\mu-1} \left(\prod_{u=0}^{m'-1} \prod_{k=0}^{2\mu-2} \left(\prod_{\substack{i=1,j=1 \\ j=(k-\mu+1)+i}}^{2\mu-2} \boldsymbol{c}_{u\mu+i}^{\boldsymbol{a}_{u\mu+j}} \right)^{x_\ell^k} \right)^{y_\ell} \\
&= \mathcal{E}_{pk}(G^0; \rho) \prod_{u=0}^{m'-1} \prod_{k=0}^{2\mu-2} \left(\prod_{\substack{i=1,j=1 \\ j=(k-\mu+1)+i}}^{2\mu-2} \boldsymbol{c}_{u\mu+i}^{\boldsymbol{a}_{u\mu+j}} \right)^{\sum_{\ell=1}^{2\mu-1} y_\ell x_\ell^k} \\
&= \mathcal{E}_{pk}(1; \rho) \prod_{u=0}^{m'-1} \prod_{\substack{i=1,j=1 \\ j=(\mu-1-\mu+1)+i}}^{2\mu-2} \boldsymbol{c}_{u\mu+i}^{\boldsymbol{a}_{u\mu+j}} = \mathcal{E}_{pk}(1; \rho) \prod_{i=1}^{m} \boldsymbol{c}_i^{\boldsymbol{a}_i}.
\end{aligned}$$

This means E has extracted a valid witness, which is the same as the witness known by the prover. Otherwise, the commitment scheme is broken, as the prover can use the extractor to find a second opening for one of their commitments. □

Efficiency: In each call of the argument with $m = m'\mu$ the prover sends 2μ ciphertexts and 2μ commitments to the verifier. So the communication cost for one round is $O(\mu)$.

To calculate the commitments the prover has to calculate 4μ exponentiations in $\mathbb{G}$. The cost to calculate on E_k consists of $m'\mu^2 n$ exponentiations in $\mathbb{H}$; therefore, the cost to calculate all 2μ values of E_k is $m'\mu^3 n$. On top of this the prover has to calculate $2m'\mu^3 n + m'\mu n + 7\mu$ multiplications.

The verifier needs to calculate 6μ exponentiations in $\mathbb{G}$ and 4μ multiplications to check the argument. Furthermore, the verifier has to calculate the new C' which costs $m'n$ exponentiations in $\mathbb{H}$. Thus, the computation cost of the prover is $O(N)$.

Assuming $m = \mu^\nu$ and both parties engage in $\boldsymbol{u}$ rounds of the protocol. The total communication cost is then $2\mu\nu$ elements of $\mathbb{H}$ and $2\mu\nu$ group elements. The provers cost adds to approximately $\mu^2 N$ exponentiations in $\mathbb{H}$ and $(3\mu^3+\mu)N+7\nu\mu$ multiplications. In each call of the argument the verifier has to calculate C' this costs adds to $O(N)$ exponentiations in $\mathbb{H}$, whereas the cost to check all arguments counts to $6\mu\nu$ exponentiations in $\mathbb{G}$ and $4\mu\nu$ multiplications.

8.4 Implementation and Practical Results

To obtain some experimental results and to analyze the shuffle argument further we implemented it in C++ using the approach described in Chapter 4. We collected data for different groups, we looked at different combination of groups with order 160-bit, 256-bit, and 384-bit, and moduli values of 1 248-bit, 1 536-bit, 2 432-bit and a 3 248-bit. Furthermore, we run the argument on a 160-bit subgroup modulo a 1 024-bit prime to compare our implementation with early protocols.

Most of the groups we used are not standard in research community but we chose them to analyze the behavior of our shuffle argument in more detail. Please see also Section 4.1.1 for a more detailed discussion on the security and usability of these groups.

In total we experimented with five different implementations to compare their relative merit:

1. Without any optimizations at all.
2. Using multi-exponentiation techniques.
3. Using multi-exponentiation and the Fast Fourier Transform.
4. Using multi-exponentiation and a round of the interactive technique with $\mu = 4$ and Toom-Cook for $m' = 4$ giving $m = \mu m' = 16$.
5. Using multi-exponentiation and two rounds of the interactive technique first with $\mu = 4$ and Toom-Cook for $m' = 4$ giving $m = \mu^2 m' = 64$.

In our first experiment we used the same underlying group $\mathbb{G}$ for the Pedersen commitments, that means the commitments are generated in $\mathbb{G}$, and the ElGamal encryption works over $\mathbb{H} = \mathbb{G} \times \mathbb{G}$. $\mathbb{G}$ was chosen as an order q subgroup of $\mathbb{Z}_p^*$, where $|q| = 160$ and $|p| = 1\,248$. We also used a group $\mathbb{G}$ with a 256-bit order modulo a 1 536-bit prime to conduct the same experiment. The results can be found in Table 8.1, 8.2 , 8.3, 8.4.

Table 8.1 states the results for different values $m = 1, 8, 16, 64$ for $N = 10\,000$ for $|q| = 160$ and $|p| = 1\,248$, and Table 8.2 for $N = 100\,000$ for the same group. Firstly, we see that the performance of the un-optimized prover gets slower for growing m. The increase of the runtime is less than the increase of m, it seems that the calculations of the E_k dominates the performance. For the version optimized with multi-exponentiation techniques or FFT we notice a similar behavior for the prover.

$N = 10\,000$	Optimization	Total time	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 1$	Un-optimized	72 s	39 s	33 s
	Multi-expo	20 s	11 s	9 s
$m = 8$	Un-optimized	77 s	63 s	15 s
	Multi-expo	17 s	14 s	3 s
	FFT	26 s	23 s	3 s
$m = 16$	Un-optimized	122 s	109 s	13 s
	Multi-expo	28 s	25 s	3 s
	FFT	32 s	29 s	3 s
	Toom-Cook	20 s	14 s	6 s
$m = 64$	Un-optimized	407 s	394 s	13 s
	Multi-expo	128 s	124 s	4 s
	FFT	50 s	46 s	4 s
	Toom-Cook	23 s	17 s	6 s
	Time Shuffle	7 s		

Table 8.1: Run-time of the shuffle arguments in seconds for $N = 10\,000$ and different choices of m for a subgroup of order 160-bit modulo a 1 248-bit prime.

$N = 100\,000$	Optimization	Total time	Time $\mathcal{P}$	Time $\mathcal{V}$
m=1	Un-optimized	724 s	393 s	331 s
	Multi-expo	366 s	205 s	161 s
m=8	Un-optimized	773 s	628 s	145 s
	Multi-expo	196 s	157 s	41 s
	FFT	301 s	251 s	50 s
m=16	Un-optimized	1 227 s	1 094 s	133 s
	Multi-expo	251 s	223 s	28 s
	FFT	323 s	294 s	29 s
	Toom-Cook	191 s	137 s	54 s
m=64	Un-optimized	4 064 s	3 941 s	124 s
	Multi-expo	822 s	795 s	26 s
	FFT	440 s	413 s	26 s
	Toom-Cook	181 s	126 s	55 s
	Time Shuffle	71 s		

Table 8.2: Run-time of the shuffle arguments in seconds for $N = 100\,000$ and different choices of m for a subgroup of order 160-bit modulo a 1 248-bit prime.

For the un-optimized verifier we see that for growing m the performance gets better, this indicates that the asymptotic cost of $N + 15m + 9n$ multiplications influences the run-time. Again the same is true for the first two optimized versions. In the case of Toom-Cook we see that the verifier runs around two times slower than the optimized verifier, this is explained by the slightly increase in the number of exponentiation.

We also see that the un-optimized version is slow and the performance gains a lot from any optimization technique. For the prover it seems that the multiplication has no influence at all; however, for the verifier the cost of the multiplications seems to have some influence.

We see that the plain multi-exponentiation techniques yield better results than the FFT method for small m; the better asymptotic behavior of the FFT only kicks in for $m > 16$. We expected that the

$N = 10\,000$	Optimization	Total time	Time $\mathcal{P}$	Time $\mathcal{V}$
m=1	Un-optimized	163 s	88 s	74 s
	Multi-expo	40 s	22 s	18 s
m=8	Un-optimized	175 s	142 s	33 s
	Multi-expo	35 s	29 s	7 s
	FFT	55 s	48 s	7 s
m=16	Un-optimized	278 s	247 s	30 s
	Multi-expo	60 s	53 s	7 s
	FFT	68 s	61 s	7 s
	Toom-Cook	41 s	29 s	12 s
m=64	Un-optimized	920 s	892 s	28 s
	Multi-expo	280 s	271 s	9 s
	FFT	107 s	98 s	9 s
	Toom-Cook	47 s	34 s	13 s
	Time Shuffle	15 s		

Table 8.3: Run-time of the shuffle arguments in seconds for $N = 10\,000$ and different choices of m for a subgroup of order 256-bit modulo a 1 536-bit prime.

$N = 100\,000$	Optimization	Total time	Time $\mathcal{P}$	Time $\mathcal{V}$
m=1	Un-optimized	1 635 s	888 s	746 s
	Multi-expo	684 s	378 s	3104 s
m=8	Un-optimized	1 748 s	1418 s	330 s
	Multi-expo	451 s	360 s	91 s
	FFT	5620 s	534 s	86 s
m=16	Un-optimized	2 778 s	2 477 s	301 s
	Multi-expo	536 s	476 s	60 s
	FFT	695 s	631 s	64 s
	Toom-Cook	384 s	273 s	111 s
m=64	Un-optimized	9 204 s	8 925 s	279 s
	Multi-expo	1 745 s	1 690 s	56 s
	FFT	941 s	886 s	56 s
	Toom-Cook	355 s	245 s	110 s
	Time Shuffle	145 s		

Table 8.4: Run-time of the shuffle arguments in seconds for $N = 100\,000$ and different choices of m for a subgroup of order 256-bit modulo a 1 536-bit prime.

Toom-Cook inspired version with added interaction give us the best running time and communication cost; however, this only true for large N. In the case of $N = 100\,000$ the performance of the prover optimized with Toom-Cook and two extra rounds of interaction out performances all other settings and therefore the complete argument is fastest even considering the slower verifier in this case. For $N = 10\,000$ plain multi-exponentiation techniques lead to best performance in overall when m is chosen as 8. All these conclusions are backed up by the data for $|q| = 256$ and $|p| = 1\,536$, which can be found in Table 8.3 and 8.4.

In the following experiments we will focus on data obtained with $m = 8, 16, 64$ and with the four different levels of optimization. Just to find out how fast our argument can be in different settings, it would be enough just to collect data for the Toom-Cook versions for big N and for the plain multi-

exponentiation techniques for small N. However, the influence of the FFT might change for different settings.

We want to see how the shuffle argument behave for even bigger N; thus, we collected data for all levels of optimizations for $N = 1\,000\,000$ and different m. The result for $|q| = 160$ and $|p| = 1\,248$, and $|q| = 256$ and $|p| = 1\,536$ can be found in Table 8.5 and Table 8.6. We see that in this case the version with two extra rounds of interaction leads also to the best performance. But, we also see that the asymptotic behavior of the FFT kicks in for smaller m and FFT beats the multi-exponentiation techniques for $m > 8$.

$N = 1\,000\,000$	Optimization	Total time	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	3 061 s	2 457 s	603 s
	FFT	3 424 s	2 819 s	605 s
$m = 16$	Multi-expo	4 615 s	4 036 s	546 s
	FFT	3 916 s	3 370 s	546 s
	Toom-Cook	2 770 s	2 052 s	718 s
$m = 64$	Multi-expo	12 131 s	11 566 s	412 s
	FFT	4 937 s	4 519 s	418 s
	Toom-Cook	2 410 s	1 781 s	629 s
	Time Shuffle	679 s		

Table 8.5: Run-time of the shuffle arguments in seconds for $N = 1\,000\,000$ and different choices of m for a subgroup of order 160-bit modulo a 1 248-bit prime.

$N = 1\,000\,000$	Optimization	Total time	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	5 627 s	4 525 s	1 102 s
	FFT	6 883 s	5 778 s	1 105 s
$m = 16$	Multi-expo	8 435 s	7 461 s	974 s
	FFT	7 956 s	6 985 s	972 s
	Toom-Cook	5 316 s	3 882 s	1 434 s
$m = 64$	Multi-expo	23 066 s	22 328 s	738 s
	FFT	10 399 s	9 665 s	734 s
	Toom-Cook	4 467 s	3 262 s	1 205 s
	Time Shuffle	1 461 s		

Table 8.6: Run-time of the shuffle arguments in seconds for $N = 1\,000\,000$ and different choices of m for a subgroup of order 256-bit modulo a 1 536-bit prime.

For the next experiment we kept the size of the subgroup fixed and used different moduli values. Table 8.7 states the data for $N = 10\,000$ for $|q| = 160$ and $|p| = 1\,248,\ 1\,536$ and $3\,248$; whereas Table 8.8 give the data for $N = 100\,000$ for the same groups. In tables 8.9 and 8.10 we find the same date for a fixed 256-bit subgroup modulo a 1 536-bit, a 2 432-bit, and a 3 248-bit prime.

We notice that increasing the modulo value by a certain factor, increases the complete run-time; however, the increase is not linear with the increase of the moduli value, the bigger p gets the more suffers the performance of the argument. One possible reason for this is that exponentiations get more expensive for bigger moduli and as seen before the cost of the exponentiations is the dominant part of the argument; thus, this cost slows everything down. This finding means that increasing the security of

$\|q\| = 160$		$\|p\| = 1\,248$		$\|p\| = 1\,536$		$\|p\| = 3\,248$	
$N = 10\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	14 s	3 s	17 s	4 s	54 s	13 s
	FFT	23 s	3 s	31 s	4 s	110 s	13 s
$m = 16$	Multi-expo	25 s	3 s	31 s	4 s	102 s	13 s
	FFT	29 s	3 s	40 s	4 s	145 s	13 s
	Toom-Cook	14 s	6 s	19 s	7 s	63 s	24 s
$m = 64$	Multi-expo	124 s	4 s	156 s	5 s	544 s	18 s
	FFT	46 s	4 s	64 s	5 s	235 s	18 s
	Toom-Cook	17 s	6 s	21 s	8 s	73 s	28 s
	Time Shuffle	7 s		10 s		37 s	

Table 8.7: Run-time of the shuffle arguments in seconds for $N = 10\,000$ for different m for a subgroup of order 160-bit modulo $|p| = 1\,248,\ 1\,536$, and $3,248$.

	$\|q\| = 160$	$\|p\| = 1\,248$		$\|p\| = 1\,536$		$\|p\| = 3\,248$	
$N = 100\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	157 s	40 s	187 s	50 s	539 s	130 s
	FFT	251 s	50 s	325 s	50 s	1 109 s	130 s
$m = 16$	Multi-expo	223 s	28 s	277 s	36 s	880 s	108 s
	FFT	294 s	29 s	403 s	35 s	1 431	108 s
	Toom-Cook	137 s	54 s	176 s	67 s	585 s	227 s
$m = 64$	Multi-expo	795 s	26 s	989 s	33 s	3 248 s	105 s
	FFT	413 s	26 s	581 s	33 s	2 138 s	105 s
	Toom-Cook	126 s	55 s	158 s	69 s	515 s	231 s
	Time Shuffle	71 s		100 s		369 s	

Table 8.8: Run-time of the shuffle arguments in seconds for $N = 100\,000$ for different m for a subgroup of order 160-bit modulo $|p| = 1\,248,\ 1\,536$, and $3\,248$.

	$\|q\| = 256$	$\|p\| = 1\,536$		$\|p\| = 2\,432$		$\|p\| = 3\,248$	
$N = 10\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	29 s	7 s	57 s	14 s	87 s	21 s
	FFT	48 s	7 s	109 s	14 s	171 s	21 s
$m = 16$	Multi-expo	53 s	7 s	106 s	13 s	164 s	21 s
	FFT	61 s	7 s	142 s	13 s	225 s	21 s
	Toom-Cook	29 s	12 s	63 s	25 s	99 s	40 s
$m = 64$	Multi-expo	271 s	9 s	564 s	19 s	880 s	29 s
	FFT	98 s	9 s	230 s	19 s	365 s	29 s
	Toom-Cook	34 s	13 s	73 s	28 s	115 s	44 s
	Time Shuffle	15 s		36 s		57 s	

Table 8.9: Run-time of the shuffle arguments in seconds for $N = 10\,000$ for different m for a subgroup of order 256-bit modulo $|p| = 1\,536,\ 2\,432$, and $3\,248$.

the shuffle argument by increasing the modulo value leads to a slower performance, and the increase in the run-time is higher than the increase of the security.

Tables 8.11, 8.12, 8.13, 8.14, state the data for the reverse case, that means the modulo value stays fixed and the order of the group changes. In this setting we decided to use $|p| = 1\,536$ and $|p| = 3\,248$, we see that the run-time gets slower for bigger subgroup sizes. For the prover we notice that the increase

	$\|q\| = 256$	$\|p\| = 1\,536$		$\|p\| = 2\,432$		$\|p\| = 3\,248$	
$N = 100\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	360 s	91 s	606 s	154 s	894 s	214 s
	FFT	534 s	86 s	1 122 s	153 s	1 737 s	214 s
$m = 16$	Multi-expo	476 s	60 s	923 s	125 s	1 415 s	175 s
	FFT	631 s	64 s	1 409 s	117 s	2 220 s	175 s
	Toom-Cook	266 s	109 s	586 s	241 s	921 s	375 s
$m = 64$	Multi-expo	1 690 s	56 s	3 408 s	110 s	5,279 s	169 s
	FFT	886 s	56 s	2 093 s	110 s	3 320 s	169 s
	Toom-Cook	259 s	81 s	514 s	234 s	808 s	365 s
	Time Shuffle	145 s		572 s		572 s	

Table 8.10: Run-time of the shuffle arguments in seconds for $N = 100\,000$ for different m for a subgroup of order 256-bit modulo $|p| = 1\,536, 2\,432$, and $3\,248$.

of the performance is linear with the increase of the subgroup size, this behavior seems to be the same for all values N. For the verifier we see a similar performance; however, for small N the increase of the run-time is slightly bigger and for big N slightly smaller than the increase of $|p|$. This indicates that increasing the security of our shuffle argument by adjusting the group size leads to a similar change in the run-time.

	$\|p\| = 1\,536$	$\|q\| = 160$		$\|q\| = 256$	
$N = 10\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	17 s	4 s	29 s	7 s
	FFT	31 s	4 s	48 s	7 s
$m = 16$	Multi-expo	31 s	4 s	53 s	7 s
	FFT	40 s	4 s	61 s	7 s
	Toom-Cook	19 s	7 s	29 s	12 s
$m = 64$	Multi-expo	158 s	5 s	271 s	9 s
	FFT	64 s	5 s	98 s	9 s
	Toom-Cook	21 s	8 s	34 s	13 s
	Time Shuffle	10 s		15 s	

Table 8.11: Run-time of the shuffle arguments in seconds for $N = 10,000$ for different m for different subgroups modulo a 1 536-bit prime.

The next experiment is inspired by the discussion of Groth [Gro10] on how to speed up shuffle arguments in practice. He claims that it is possible to speed up the shuffle by choosing two different groups for the commitment scheme and the encryption. The only constraint is that both modular groups have the same order. To analyze this in more detail we decided to run our shuffle argument for this setting. We fixed our order to be 160-bit and used three different combinations for the moduli values. First we used the same group for the commitments and the encryption, in our case for a 1 248-bit, a 2 432-bit, and a 3 248-bit prime. Then, we tested the shuffle for different groups for the commitment and the encryption, here we set the commitment modulus to be 1 248-bit and the encryption modulus to be 2 432-bit or 3 248-bit. The results can be found in Table 8.15 - 8.18. We also conducted the same experiment for $|q| = 256$ and $|p| = 1\,536, 2\,432, 3\,248$.

Keeping the commitment group fixed and changing the group underlying encryption obvious leads

	$\|p\| = 1\,536$	$\|q\| = 160$		$\|q\| = 256$	
$N = 100\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	187 s	50 s	360 s	91 s
	FFT	325 s	50 s	534 s	86 s
$m = 16$	Multi-expo	277 s	36 s	476 s	60 s
	FFT	403 s	35 s	631 s	64 s
	Toom-Cook	176 s	67 s	266 s	109 s
$m = 64$	Multi-expo	989 s	33 s	1 690 s	56 s
	FFT	581 s	33 s	886 s	56 s
	Toom-Cook	158 s	69 s	259 s	81 s
	Time Shuffle	100 s		145 s	

Table 8.12: Run-time of the shuffle arguments in seconds for $N = 100\,000$ for different m for different subgroups modulo a 1 536-bit prime.

	$\|p\| = 3\,248$	$\|q\| = 160$		$\|q\| = 256$		$\|q\| = 384$	
$N = 10\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	54 s	13 s	87 s	21 s	129 s	31 s
	FFT	110 s	13 s	171 s	21 s	243 s	31 s
$m = 16$	Multi-expo	102 s	13 s	164 s	21 s	244 s	31 s
	FFT	145 s	13 s	225 s	21 s	321 s	31 s
	Toom-Cook	63 s	24 s	99 s	40 s	146 s	57 s
$m = 64$	Multi-expo	545 s	18 s	880 s	29 s	1,310 s	44 s
	FFT	235 s	18 s	365 s	29 s	522 s	44 s
	Toom-Cook	73 s	28 s	115 s	44 s	171 s	62 s
	Time Shuffle	37 s		57 s		83 s	

Table 8.13: Run-time of the shuffle arguments in seconds for $N = 10\,000$ for different m for different subgroups modulo a 3 248-bit prime.

	$\|p\| = 3\,248$	$\|q\| = 160$		$\|q\| = 256$		$\|q\| = 384$	
$N = 100\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	539 s	130 s	894 s	214 s	1 355 s	309 s
	FFT	1 109 s	130 s	1 737 s	214 s	2 475 s	306 s
$m = 16$	Multi-expo	880 s	108 s	1 415 s	175 s	2 100 s	258 s
	FFT	1 431 s	108 s	2 220 s	175 s	3 165 s	258 s
	Toom-Cook	585 s	227 s	921 s	375 s	1 356 s	537 s
$m = 64$	Multi-expo	3 284 s	105 s	5 279 s	169 s	7 836 s	250 s
	FFT	2 138 s	105 s	3 320 s	169 s	4 745 s	250 s
	Toom-Cook	514 s	231 s	808 s	365 s	1 192 s	519 s
	Time Shuffle	369 s		572 s		824 s	

Table 8.14: Run-time of the shuffle arguments in seconds for $N = 100\,000$ for different m for different subgroups modulo a 3 248-bit prime.

to an increase of the run-time. For the verifier this increase is similar to the rise of the modulo value p, as their run-time is dominated by the verification of the ciphertexts. A similar increase can be seen for the prover.

If we compare the run-time of the case with two different groups to the situation with one group with big modulo value, we see that there is only a small increase in the performance of the verifier. This

	$\|q\| = 160$	$\|p_1\| =$ 1 248	$\|p_2\| =$ 1 248	$\|p_1\| =$ 1 248	$\|p_2\| =$ 2 432	$\|p_1\| =$ 2 432	$\|p_2\| =$ 2 432
$N = 100\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
m=8	Multi-expo	157 s	40 s	317 s	83 s	342 s	91 s
	FFT	251 s	50 s	656 s	79 s	676 s	88 s
m=16	Multi-expo	223 s	28 s	530 s	72 s	549 s	76 s
	FFT	294 s	29 s	843 s	66 s	858 s	68 s
	Toom-Cook	137 s	54 s	336 s	135 s	358 s	139 s
m=64	Multi-expo	795 s	26 s	2 003 s	64 s	2 011 s	65 s
	FFT	413 s	26 s	1 262 s	64 s	1 273	65 s
	Toom-Cook	126 s	55 s	217 s	61 s	316 s	140 s
	Time Shuffle	71 s		222 s		218 s	

Table 8.15: Run-time of the shuffle arguments in seconds for $N = 100\,000$ for different m for $|q| = 160$ and different moduli values $p_1\, p_2$ with $|p_1| = 1\,248,\ 2\,432$, $|p_2| = 1\,248,\ 2\,432$.

	$\|q\| = 160$	$\|p_1\| =$ 1 248	$\|p_2\| =$ 1 248	$\|p_1\| =$ 1 248	$\|p_2\| =$ 3 248	$\|p_1\| =$ 3 248	$\|p_2\| =$ 3 248
$N = 100\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
m=8	Multi-expo	157 s	40 s	480 s	114 s	539 s	130 s
	FFT	251 s	50 s	1 062 s	115 s	1 109 s	130 s
m=16	Multi-expo	223 s	28 s	835 s	101 s	880 s	108 s
	FFT	294 s	29 s	1 392 s	101 s	1 431 s	108 s
	Toom-Cook	137 s	54 s	536 s	217 s	585 s	227 s
m=64	Multi-expo	795 s	26 s	3 232 s	101 s	3 284 s	105 s
	FFT	413 s	26 s	2 102 s	102 s	2 138 s	105 s
	Toom-Cook	126 s	55 s	338 s	95 s	514 s	231 s
	Time Shuffle	71 s		368 s		369 s	

Table 8.16: Run-time of the shuffle arguments in seconds for $N = 100\,000$ for different m for $|q| = 160$ and different moduli values p_1, p_2 with $|p_1| = 1\,248,\ 3\,248$, $|p_2| = 1\,248,\ 3\,248$.

	$\|q\| = 256$	$\|p_1\| =$ 1 536	$\|p_2\| =$ 1 536	$\|p_1\| =$ 1 536	$\|p_2\| =$ 2 432	$\|p_1\| =$ 2 432	$\|p_2\| =$ 2 432
$N = 100\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	360 s	91 s	544 s	140 s	606 s	154 s
	FFT	534 s	86 s	1 035 s	134 s	1 122 s	153 s
$m = 16$	Multi-expo	476 s	60 s	854 s	105 s	923 s	125 s
	FFT	631 s	64 s	1 312 s	107 s	1 409 s	117 s
	Toom-Cook	266 s	109 s	534 s	225 s	586 s	241 s
$m = 64$	Multi-expo	1 690 s	56 s	3 213 s	104 s	3 408 s	110 s
	FFT	886 s	56 s	1 953 s	103	2 093 s	110 s
	Toom-Cook	259 s	81 s	343 s	96 s	514 s	234 s
	Time Shuffle	145 s		354 s		358 s	

Table 8.17: Run-time of the shuffle arguments in seconds for $N = 100\,000$ for different m for $|q| = 256$ and different moduli values p_1, p_2 with $|p_1| = 1\,536,\ 2\,432$ $|p_2| = 1\,536,\ 2\,432$.

was expected as the verifier calculated only a small number of commitments. For the prover the rise of the run-time is noticeable, but they have to calculate a big number of commitments.

All in all, the difference between the two versions is small for our parameter, choosing the different

	$\|q\| = 256$	$\|p_1\| =$ 1 536	$\|p_2\| =$ 1 536	$\|p_1\| =$ 1 536	$\|p_2\| =$ 3 248	$\|p_1\| =$ 3 248	$\|p_2\| =$ 3 248
$N = 100\,000$	Optimization	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$	Time $\mathcal{P}$	Time $\mathcal{V}$
$m = 8$	Multi-expo	360 s	91 s	815 s	193 s	894 s	214 s
	FFT	534 s	86 s	1 660 s	192 s	1 737 s	214 s
$m = 16$	Multi-expo	476 s	60 s	1 350 s	164 s	1 415 s	175 s
	FFT	631 s	64 s	2 153 s	164 s	2 220 s	175 s
	Toom-Cook	266 s	109 s	852 s	362 s	921 s	375 s
$m = 64$	Multi-expo	1 690 s	56 s	5 228 s	165 s	5 279 s	168 s
	FFT	886 s	56 s	3 259 s	165 s	3 320 s	169 s
	Toom-Cook	259 s	81 s	534 s	149 s	808 s	365 s
	Time Shuffle	145 s		573 s		572 s	

Table 8.18: Run time of the shuffle arguments in seconds for $N = 100\,000$ for different m for $|q| = 160$ and different moduli values p_1, p_2 with $|p_1| = 1\,248, 3\,248$, $|p_2| = 1\,248, 3\,248$.

between p_1 and p_2 bigger would lead to an increased performance gain using two different groups. However, one has to be careful to choose all values in a way that the required security level is still accomplished.

Round	1	2	3	4	5	6	7	8	9
$m = 1$	1 KB	0.1 KB	1 KB	0.2 KB	4 KB	778 KB	782 KB	1 KB	5.5 MB
$m = 8$	4 KB	0.1 KB	4 KB	0.2 KB	26 KB	99 KB	107 KB	1 KB	487 KB
$m = 16$	7 KB	0.1 KB	1 KB	0.2 KB	51 KB	50 KB	65 KB	3 KB	238 KB
$m = 64$	30 KB	0.1 KB	30 KB	0.2 KB	208 KB	22 KB	74 KB	10 KB	61 KB

Table 8.19: Size of each round for $N = 10\,000$ for different m for $|q| = 256$ and $|p| = 1536$.

Next, we have to say a few words on the shuffling operations. All tables also state the time of one shuffle operation, we see that in general the total execution of one zero-knowledge argument is around 3 times of the execution of one shuffle operation. However, if for the commitments a smaller moduli value is used we see that our argument can be evaluated in a similar time as the entire shuffle operation.

Lastly, we have to discuss the argument size, Table 8.19 states the size of each round for $|q| = 256$, $|p| = 1\,536$, and $N = 100\,000$. In round $2, 4, 6$ and 8 the verifier sends challenges to the prover. We see that challenges in round 2 and 4 are really cheap. Whereas in round 6 and 8 the challenge contains a few kilobytes. However, the cost of these two rounds can be reduced to 0.1 kilobytes, with a slightly increase of the run-time, in our implementation the verifier transfers vectors $(x, x^2, \ldots)$ to the prover.

The size of round $1, 3$ are dominated by the size of $O(m)$ commitments and round 5 by the size of $O(m)$ commitments and ciphertexts, and we see that the size for all 3 rounds behave linear to the increase of m. Round 9 is dominated by the transfer of $5n$ field elements, for our different values m, n we see that the cost of this round gets smaller for bigger values m. We also see that the cost of this round dominates the argument size for small m, which seems a little bit surprising as we only send field elements instead of group elements in round $1, 3, 5$. But, the number of commitments is very small in this round and the huge number of field elements leads to the effect we see. In round 7 the prover sends $2m$ group elements and $7n$ field elements, again for small m the huge number of field elements dominates

the argument size. For growing m the number of field elements drops and with this also the size of the round until the number of commitments take over and the size grows again. This effect occurs for m between 16 and 64.

Theoretically we should get best communication performance for $m = n = \sqrt{N}$; nevertheless, the practical analysis of the different parameters shows that the actual minimum of the argument size occurs much earlier. For this special parameters setting, the minimum occurs for $m = 64$, Table 8.20 states the complete run-time for different combination of q and p and $N = 10\,000$. We see that in this case the minimum seems also to occur between 16 and 64.

	$\lvert q\rvert =$	160	160	160	256	256	256	384
	$\lvert p\rvert =$	1 248	1 536	3 248	1 536	2 432	3 248	3 248
	Statement	15 MB	19 MB	39 MB	19 MB	29 MB	39 MB	39 MB
	Witness	0.5 MB	0.5 MB	0.5 MB	0.8 MB	0.8 MB	0.8 MB	1.3 MB
$m = 8$	Multi-expo	0.5 MB	0.5 MB	0.5 MB	0.7 MB	0.8 MB	0.8 MB	1.1 MB
$m = 16$	Multi-expo	0.3 MB	0.4 MB	0.4 MB	0.4 MB	0.5 MB	0.5 MB	0.7 MB
	Toom-Cook	0.3 MB	0.3 MB	0.4 MB	0.4 MB	0.4 MB	0.5 MB	0.7 MB
$m = 64$	Multi-expo	0.3 MB	0.6 MB	0.8 MB	0.4 MB	0.6 MB	0.8 MB	0.9 MB
	Toom-Cook	0.2 MB	0.3 MB	0.4 MB	0.3 MB	0.4 MB	0.5 MB	0.5 MB

Table 8.20: Size of the shuffle argument in Mega Byte for $N = 10\,000$ for different m and for different values q, p.

In Table 8.21 we find the same data for $N = 100\,000$. As the values of m are fixed for each experiment n is 10 time bigger for this N than before. So, we expect that the number of field elements dominates the argument size for bigger m and in fact we cannot determine the minimum for our choices of m; it seems that it occurs for $m > 64$. So, we see that for growing N the minimum occurs later and therefore, the assumed minimum for $n = m$ is asymptotically correct.

The former discussion has shown that for $N = 10\,000$ we have best performance for $m = 8$ and in Table 8.20 we see that size for $m = 8$ is less than 1 megabyte for all security parameters. This is much smaller than the size of the input and output ciphertexts. Therefore, our argument has size which is sublinear in the statement size. Moreover, Table 8.20 also states the size of the witnesses we see that the argument is also sublinear in the witness size.

Similar observations can be made for $N = 100\,000$, in this case Toom-Cook with extra interaction leads to best run-time and also to a very small argument. For $|p| \leq 2\,432$ the whole argument is under 1 megabytes and even for $|p| = 3\,248$ we have an argument size under 2 megabyte, for parameters which give the best run-time.

Lastly, in Table 8.22 we see the argument size of for a group of 256-bit order modulo a 1 536-bit prime and for $N = 10\,000,\ 100\,000$, and $N = 1\,000\,000$. We see that for small m the size of our shuffle argument behaves linearly with the increase of the number of ciphertexts. However for bigger m this is not the case and the increase of the size is bigger, the exact reason for this is not clear. We see that also for $N = 1\,000\,000$ the size of the shuffle argument is still small under 10 megabytes for Toom-Cook with extra interaction which leads to the best computation performance.

To conclude, our shuffle argument has a very small argument size, independent from the underlying

	$\|q\| =$	160	160	160	256	256	256	384
	$\|p\| =$	1 248	1 536	3 248	1 536	2 432	3 248	3 248
	Statement	151 MB	186 MB	392 MB	186 MB	294 MB	392 MB	392 MB
	Witness	5.5 MB	5.5 MB	5.5 MB	8.6 MB	8.6 MB	8.6 MB	MB
$m = 8$	Multi-expo	4.3 MB	4.3 MB	4.4 MB	6.9 MB	6.9 MB	6.9 MB	10.3 MB
$m = 16$	Multi-expo	2.2 MB	2.3 MB	2.3 MB	3.5 MB	3.5 MB	3.6 MB	5.0 MB
	Toom-Cook	2.2 MB	2.2 MB	2.3 MB	3.5 MB	3.5 MB	3.5 MB	5.2 MB
$m = 64$	Multi-expo	0.8 MB	1.1 MB	1.2 MB	1.2 MB	1.4 MB	1.6 MB	2.0 MB
	Toom-Cook	0.6 MB	0.8 MB	0.9 MB	0.9 MB	1.0 MB	1.2 MB	1.5 MB

Table 8.21: Size of the shuffle argument in Mega Byte for $N = 100\,000$ for different m and for different values q, p.

		$N = 10\,000$	$N = 100\,000$	$N = 1\,000\,000$
	Statement	19 MB	186 MB	1 859 MB
	Witness	0.8 MB	8.6 MB	85 MB
$m = 1$	Multi-expo	5.5 MB	51.9 MB	544.9 MB
$m = 8$	Multi-expo	0.7 MB	6.9 MB	68.6 MB
$m = 16$	Multi-expo	0.4 MB	3.5 MB	34.2 MB
	Toom-Cook	0.4 MB	3.5 MB	34.1 MB
$m = 64$	Multi-expo	0.4 MB	1.2 MB	8.4 MB
	Toom-Cook	0.3 MB	0.9 MB	7.5 MB

Table 8.22: Size of the shuffle argument in Mega Byte for $N = 10\,000, 100\,000, 1\,000\,000$ for different m for $|q| = 256$ and $|p| = 1536$.

groups and security parameters. The performance for medium range $N \leq 100\,000$ is on our personal computer under 20 minutes, even for long term security. In real life applications, such as e-voting, the protocol runs on high speed servers which leads to much better performance. For $N = 1\,000\,000$ our performance takes over one hour for medium range parameters, and interpolating this with the other results, we expect a run-time around of three to four hours for long term security. This seems not practical; however, the use of high speed computers and better implementation techniques would help to reduce the run-time and in this case our argument might be usable in real life for huge N.

8.5 Comparison

8.5.1 Theoretical comparison

Previous work in the literature mainly investigated the case where we use ElGamal encryption and commitments over the same group $\mathbb{G}$, i.e. $\mathbb{H} = \mathbb{G} \times \mathbb{G}$ were used. Table 8.23 gives the asymptotic behavior of these protocols compared to our protocol for $N = mn$ as m and n grow.

We see that our argument size is much smaller than the former ones. Furthermore, our verifier gives best asymptotic behavior so far. It seems so far that $4N$ exponentiations is the asymptotic cost for the verifier using ElGamal encryption and counting single exponentiations, as they have to handle each input and output ciphertext at least once.

For the prover, our computation cost is bigger than the computation cost of former provers. However, for small m we are in a similar range and together with our light verifier the run-time of the complete

SHVZK argument	Rounds	Time $\mathcal{P}$ Expos	Time $\mathcal{V}$ Expos	Size Elements
[FS01]	3	$8N$	$10N$	$5N\ \mathbb{G}$+ $N\ \mathbb{Z}_q$
[FMM$^+$02]	5	$9N$	$10N$	$5N\ \mathbb{G}$ + $N\ \mathbb{Z}_q$
[Gro10]	7	$6N$	$6N$	$3N\ \mathbb{Z}_q$
[Fur05]	3	$7N$	$8N$	$N\ \mathbb{G}$+ $2N\ \mathbb{Z}_q$
[TW10]	5	$9N$	$11N$	$3N\ \mathbb{G} + 4N\ \mathbb{Z}_q$
[GI08]	7	$3mN$	$4N$	$3m^2\ \mathbb{G}$+ $3n\ \mathbb{Z}_q$
This work	9	$2\log(m)N$	$4N$	$11m\ \mathbb{G}$+ $5n\ \mathbb{Z}_q$

Table 8.23: Comparison of the protocols with ElGamal encryption.

argument should be in a similar range or faster.

In our protocol, we may as detailed in Section 8.3.1 use FFT techniques to reduce the prover's computation to $O(N \log m)$ exponentiations as listed in Table 8.23. Furthermore, by increasing the round complexity as in Section 8.3.2 we could even get a linear complexity of $O(N)$ exponentiations. These techniques do not apply to the other shuffle arguments, in particular it is not possible to use FFT techniques to reduce the factor m in the shuffle by Groth and Ishai [GI08].

As the multi-exponentiation argument, which is the most expensive step, already starts in round 3 we can insert two rounds of interactive reduction as described in Section 8.3.2 without increasing the round complexity above 9 rounds. For practical parameters this would give us enough of a reduction to make the prover's computation comparable to the schemes with linear $O(N)$ computation.

The figures in Table 8.23 are for non-optimized versions of the schemes. All of the schemes may for instance benefit from the use of multi-exponentiation techniques, see Section 4.3 for how to compute a product of n exponentiations using only $O\left(\frac{n}{\log n}\right)$ multiplications. The schemes may also benefit from randomization techniques, where the verifier does a batch verification of all the equations it has to check.

8.5.2 Comparison with other implementations

Furukawa, Miyauchi, Mori, Obana, and Sako [FMM$^+$02] gave performance results for a mix-net using a version of the Furukawa-Sako [FS01] shuffle arguments. They optimized the mix-net by combining the shuffling and decryption operations into one. They used three shuffle centers communicating with each other and their results included both the process of shuffling and the cost of the arguments. So, to compare the values we multiply our shuffle argument times by 3 and add the cost of our shuffling operation on top of that. The comparison can be found in Table 8.24.

$N = 100\,000$	[FMM$^+$02]	This paper
Single argument	51 min	15 min
Argument size	66 MB	2 MB
Total mix-net time	3 hrs. 44 min	53 min

Table 8.24: Run-time comparison of [FMM$^+$02] (CPU: 1 GHz, Ram: 256 MB) to our shuffle argument (Toom-Cook with $m = 64$, CPU: 1.4 GHz, Ram: 256 MB) for a group $\mathbb{G}$ with order 160-bit modulo a 1 024-bit prime.

We expected to get better performance than they did, as their shuffle needs $19N$ exponentiation

in total and our protocol with Toom-Cook and 2 extra rounds of interaction needs 16 exponentiations. Indeed we see that our argument is faster and the communication cost is a factor 33 smaller. When adding the cost of shuffling and decryption to our argument we still have a speed increase by a factor 3 in Table 3 when comparing the two mix-net implementations and taking the difference in the machines into account.

Recently, Furukawa et al. [FMS10] announced a new implementation based on elliptic curve groups. Due to the speed of using elliptic curves this gave them a speedup of a factor 3. A similar speedup can be expected for our shuffle argument if we switch to using elliptic curves in our implementation.

Stamer [Sta05] reported on an implementation of Groth's shuffle [Gro10]. However, this was an un-optimized version and he only reported on results up to $N = 1\,000$.

Recently Wikström made a complete implementation of a mix-net in Java in [Wik10] called Verificatum, which is based on the shuffle argument in [TW10]. To produce comparable data, we ran the demo file with only one mix party in the non-interactive mode using the same modular group as in our protocol. Verificatum is a full mix-net implementation, for fairness in the comparison we only counted the time of the relevant parts for the shuffle argument. As described in Table 8.23 the theoretical performance of Verificatum's shuffle argument is $20N$ exponentiations, while our prover with Toom-Cook and 2 extra rounds of interaction uses $12N$ exponentiations and our verifier $4N$, so in total $16N$ exponentiations. So, we expect a similar run-time for the Verificatum mix-net. As shown in Table 8.25 we perform better, but due to the different programming languages used and different levels of optimization in the code we will not draw any conclusion except that both protocols are efficient and usable in current applications. In terms of size it is clear that our arguments leave a much smaller footprint than Verificatum; we save a factor 50 in the communication cost.

$N = 100\,000$	[TW10]	This paper Toom-Cook
Single argument	5 min	2 min
Argument size	37.7 MB	0.7 MB

Table 8.25: Run-time comparison of [TW10] to our shuffle argument for a group $\mathbb{G}$ with order 160-bit modulo a 1 024-bit prime.

Chapter 9

Conclusion

One of the important challenges in modern cryptology is to find efficient zero-knowledge proof systems which can be used to construct real life protocols [Joh00]. In this work we addressed this problem and were able to answer the challenge positively for zero-knowledge arguments based on the discrete logarithm assumption. Our new zero-knowledge arguments are all sublinear in the statement size; in addition, our shuffle argument is also sublinear in the witness size. Furthermore, they all have low computation cost which is usable in real-world applications.

There are various reasons why we accomplished construction of light zero-knowledge arguments. First we have chosen our commitment scheme carefully. The general Pedersen commitment is homomorphic and therefore allows to verify sophisticated combinations of commitments, it therefore reduces the computation complexity. Also the Pedersen commitment is length reducing, since it allows to conceal many elements in one commitment, which is important to get sublinear communication cost. The next reason is the use of Vandermonde challenges. This affords us the sophisticated batch verification and therefore reduce the communication and computation complexity.

The previous two reasons apply to all our protocols; in addition, we used special techniques for different settings. To reduce the complexity of our polynomial arguments, we have written the X^i in binary, similar to multi-exponentiation techniques, which gives us a logarithmic cost. Another trick we used for different arguments, for example batch-polynomial evaluation or shuffle argument, is to arrange the witness in a $m \times n$ matrix, together with the general Pedersen commitment this leads to sublinear communication complexity. Lastly, we used Lagrange interpolation polynomials in our constructions, which leads to sophisticated ways to verify our commitments.

All these techniques combined gave us zero-knowledge arguments for various applications, which all have sublinear cost; therefore, compare very favorably against former work for the same problems. Theoretical comparison also showed that the asymptotic computation cost of our verifier is less than the cost of the comparable verifier of former protocols, for example our verifier in the non-membership argument, section 7.2, is much lighter than Brands et al.'s [BDD07] verifier.

On the downside, the comparison also showed that our prover of the polynomial argument and the non-membership argument have increased asymptotic complexity compared to the state of the art [BDD07]. Furthermore, our prover in the shuffle argument has only quasilinear computational com-

plexity. Nevertheless, for reasonable parameters the complexity is in the same range as for provers with linear complexity [TW10, Fur05, Gro10].

Therefore, just from the theoretical comparison, it was not clear if we were able to answer the challenge proposed by Johnson [Joh00] successfully. On the one hand we have the very light communication cost and fast verifier. On the other hand we have the prover with increased computation. To be able to say that we found practical zero-knowledge arguments, more work would be needed.

To evaluate the practicality of our new zero-knowledge arguments we implemented them in C++. We tested data for different parameters and different security levels.

For our polynomial evaluation argument we decided to implement the state of the art [BDD07] to have a direct comparison. We have seen that in this case our new argument size is much smaller than Brands et al.'s argument. Moreover, the running time of our complete argument is faster than Brands et al.'s argument; however, in the case of very big degree polynomials our prover needs more time than Brands et al.'s.

We also implemented also our non-membership argument and Brands et al.'s non-membership argument. We found out that also in this case our argument has the better running time and the better behavior for higher security levels. Nevertheless, our prover runs slower than Brands et al.'s prover for big blacklists and for really big blacklists the complete running time of Brands can be faster than our argument.

Finally, we also compared our shuffle argument with former implementations [FMS10, TW10] and found out that the complete run time of our argument can be faster than former work, if the parameters are chosen correctly; however, also for other parameter settings we are in the same performance range. More importantly our argument is much smaller than the argument of all former shuffle protocols; in addition, our argument size is even sublinear in the witness size.

To recap, the data showed for our implementation that the run time of our arguments is fast enough to be considered practical for parameters which offer medium to long time security, and therefore they are usable in real life. Randomization techniques [BGR98, Gro10] can be used to reduce the computational burden further. Furthermore, more sophisticated implementation techniques and faster machines would speed up the performance even further and this might be enough to make them practical for bigger parameter ranges or for higher security levels.

Our practical experiments showed four other points. Firstly, it is misleading only to look at the asymptotic computation cost of the most expensive operation to compare arguments theoretically. Extensive use of some cheap operation can dominate the run-time, and ignoring this cost in the asymptotic comparison can lead to false conclusions. To make this more clear, our polynomial evaluation argument prover needs only to calculate $O(\log D)$ exponentiations, whereas Brands et al.'s [BDD07] prover has to calculate $O(\sqrt{D})$ exponentiations. However, the conclusion that our prover performs asymptotically better than Brands et al.'s prover is wrong, as we have seen in Section 6.6.1. For big degrees of D the cost of the cheap multiplications dominate the performance of our prover.

As expected, the data supports the belief that replacing the most expensive operation by the same

asymptotic number of a cheaper operation and some smaller number of the expensive operation leads to better performance. In the case of Brand et al.'s non-membership argument we have seen that batching many equations together and therefore reducing the linear cost of the exponentiations to a square root cost but adding a linear multiplication cost on top, helped to reduce the run time of the verifier drastically. Therefore, we can conclude that it is always preferable to reduce the asymptotic cost of the most expensive operation in an interactive protocol; however, one has to be careful that at the same time the number of cheap operations do not start to dominant and the total asymptotic cost gets more expensive.

Next, we have seen that adding some cheap zero-knowledge protocols which consists of a few operations can lead to much slower performance. To convert Brands et al.'s non-membership proof into a polynomial evaluation argument one has to add only simple multiplication arguments which are by themselves cheap and efficient. However, in this case a large number of these arguments were added and in summation this cost doubled the original cost. Thus, it can be deceptive to say that two protocols behave the same, as they consist of the same basic protocol and only cheap operations were added. Asymptotically this might be correct, but for the performance it makes a huge difference. Depending on the numbers of operations added this cost can dominate the run-time and the performance can change drastically.

For our polynomial and blacklist protocols we have seen that the biggest part of the argument size comes from the group elements. In both arguments the cost of the challenges and the answers are very small compared to the group elements, since we send approximately the same number of elements. For our shuffle argument we have seen that a huge number of field elements can dominate the argument size, if the number of ciphertexts and commitments are very small compared to the number of field elements.

That means comparing only the asymptotic communication number of elements, which are sent between the two parties, of two different protocols can lead to a wrong conclusion, as the size of the different elements influence the size of the cost. For instance Groth's [Gro10] shuffle argument has linear communication cost; however, this cost arises from the linear number of field elements and only a constant small number of groups elements are sent in this argument. For certain parameters the cost of this shuffle argument could be smaller than our square root communication cost, since our number of group elements is not constant.

These results are not surprising; however, the results give real evidence that the points should be kept in mind when comparing different protocols. Only comparing the asymptotic cost of the most expensive operation of protocols can lead to wrong conclusions, the cost of the other operations should always be considered, as they can be dominant. Comparing the asymptotic cost of argument size has also to be done carefully, the different sizes of different elements has to be taken in account.

To conclude, our work has made a big step in answering the challenge presented by Johnson [Joh00] to construct practical and efficient zero-knowledge arguments which can be used as building blocks successfully. All our arguments have sublinear communication cost and perform best for medium size parameters, for example the degree of the polynomial or the number of ciphertexts in a shuffle. If the parameters are chosen in this range our zero-knowledge arguments are practical for medium to long time

security.

Further research might investigate if our techniques can be applied to other cryptographic protocols and make them more practical for the same medium parameter range, or find ways to reduce the computation cost of the prover.

To calculate the protocols' run-time we implemented it, but implementing is error-prone and time consuming. The implication of this being that to implement a new protocol every time a new protocol is constructed is undesirable. One way around the implementation could be a cost model. Knowing the costs of multiplications and exponentiations, along with the cost of memory access, we can compute the detailed cost of a protocol. Building such cost tables and checking them against real implementations, could be another direction to go.

One of the downsides of our shuffle argument is the high round complexity. To reduce this cost, one could use the linear algebra techniques by Seo [Seo11] as a starting point to construct a zero-knowledge product argument with fewer rounds, but similar computation and communication complexity.

Another interesting question is if it is possible to construct zero-knowledge protocols for big parameters as well. The practical evaluation and discussion of our results indicates that research could concentrate on reducing the number of commitments while keeping the number of all other elements small. This approach could lead to zero-knowledge protocols with very light communication.

Regarding the computational cost, our techniques lead to light verifiers and not so efficient provers. To make the protocols applicable for long term security and big parameter ranges, new ideas are needed to keep the number of exponentiations small without suffering the high number of multiplications, as we did.

.1 Proof for matrix M invertible

In this section we will demonstrate that a matrix

$$M = \begin{pmatrix} l(x_1) & l_1(x_1) & \dots & l_m(x_1) \\ & & \vdots & \\ l(x_{m+1}) & l_1(x_{m+1}) & \dots & l_m(x_{m+1}) \end{pmatrix}$$

is always invertible, for $x_i \neq x_j$ for all $1 \leq i, j \leq m$ and $i \neq j$. Linear algebra tells us, that a matrix A is invertible if $\det A \neq 0$; thus, we have to show that $\det M \neq 0$. We also know that if

$$A = B \cdot C,$$

then

$$\det A = \det B \det C.$$

The first step is to rewrite the matrix M as

$$
\begin{aligned}
M &= \begin{pmatrix} l(x_1) & l_1(x_1) & \dots & l_m(x_1) \\ & & \vdots & \\ l(x_{m+1}) & l_1(x_{m+1}) & \dots & l_m(x_{m+1}) \end{pmatrix} \\
&= \begin{pmatrix} 1 + a_1x_1 + \ldots + a_mx_1^m & 1 + a_{11}x_1 + \ldots + a_{1m-1}x_1^{m-1} & \dots & 1 + a_{m1}x_1 + \ldots + a_{mm-1}x_1^{m-1} \\ & & \vdots & \\ 1 + a_1x_{m+1} + \ldots + a_mx_{m+1}^m & 1 + a_{11}x_{m+1} + \ldots + a_{1m-1}x_{m+1}^{m-1} & \dots & 1 + a_{m1}x_{m+1} + \ldots + a_{mm-1}x_{m+1}^{m-1} \end{pmatrix} \\
&= \begin{pmatrix} 1 & x_1 & \dots & x_1^m \\ 1 & x_2 & \dots & x_2^m \\ & & \vdots & \\ 1 & x_{m+1} & \dots & x_{m+1}^m \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & \dots & 1 \\ a_1 & a_{11} & \dots & a_{m1} \\ & & \vdots & \\ a_{m-1} & a_{1m-1} & \dots & a_{mm-1} \\ a_m & 0 & \dots & 0 \end{pmatrix} \\
&= V \cdot P
\end{aligned}
$$

V is a transposed Vandermonde matrix, and therefore we have $\det V \neq 1$. The next step is to show that $\det P \neq 0$. To do this we expand along the last row and we get

$$
\det P = a_m \det \begin{pmatrix} 1 & \dots & 1 \\ a_{11} & \dots & a_{m1} \\ & \vdots & \\ a_{1m} & \dots & a_{mm} \end{pmatrix} = a_m Q.
$$

Each column of the matrix Q consists of the coefficient of the Lagrange polynomials $l_1(x), \ldots, l_m(x)$, sine the Lagrange polynomials are linear independent we can conclude

that the columns of the matrix are independent. Therefore, we have

$$\det M = \det V \cdot \det P = \underbrace{a_m}_{\neq 0} \underbrace{\det V}_{\neq 0} \cdot \underbrace{\det Q}_{\neq 0} \neq 0,$$

and can conclude that M is invertible for all values $x_1, \ldots, x_m$, for $x_i \neq x_j$ and $1 \leq i, j \leq m$, $i \neq j$.

Bibliography

[Abe98] M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. *EUROCRYPT*, LNCS vol. 1403:437–447, 1998.

[Abe99] M. Abe. Mix-networks on permutation networks. *ASIACRYPT*, LNCS vol. 1716:258–273, 1999.

[AH01] M. Abe and F. Hoshino. Remarks on mix-network based on permutation networks. *PKC*, LNCS vol. 1992:317–324, 2001.

[AST02] G. Ateniese, D. Song, and G. Tsudik. Quasi-efficient revocation in group signatures. *Financial Cryptography*, LNCS vol. 2357:183–197, 2002.

[Bab85] L. Babai. Trading group theory for randomness. *STOC*, pages 421–429, 1985.

[BBP91] J. Boyar, G. Brassard, and R. Peralta. Subquadratic zero-knowledge. *FOCS*, pages 69–78, 1991.

[BC86] G. Brassard and C. Crépeau. Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for sat and beyond. *FOCS*, pages 188–195, 1986.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BCY91] G. Brassard, C. Crépeau, and M. Yung. Constant-round perfect zero-knowledge computationally convincing protocols. *Theor. Comput. Sci.*, 84(1):23–52, 1991.

[BDD07] S. Brands, L. Demuynck, and B. De Decker. A practical system for globally revoking the unlinkable pseudonyms of unknown users. *ACISP*, LNCS vol. 4586:400–415, 2007.

[BdM93] J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). *EUROCRYPT*, LNCS vol. 765:274–285, 1993.

[BG92] M. Bellare and O. Goldreich. On defining proofs of knowledge. *CRYPTO*, LNCS vol. 740:390–420, 1992.

[BG12] S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. *EUROCRYPT*, LNCS vol. 7237:263–280, 2012.

[BG13] S. Bayer and J. Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. *EUROCRYPT*, LNCS vol. 7881:646–663, 2013.

[BGMW98] E. Brickell, D. Gordon, K. McCurley, and D. Wilson. Fast exponentiation with precomputation (extended abstract). *EUROCRYPT*, LNCS vol. 658:200–207, 1998.

[BGR98] M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. *EUROCRYPT*, LNCS vol. 1403:236–250, 1998.

[BJY97] M. Bellare, M. Jakobsson, and M. Yung. Round-optimal zero-knowledge arguments based on any one-way function. *EUROCRYPT*, LNCS vol. 1233:280–305, 1997.

[BLP93] J. Boyar, C. Lund, and R. Peralta. On the communication complexity of zero-knowledge proofs. *J. Cryptology*, 6(2):65–85, 1993.

[BMO90] M. Bellare, S. Micali, and R. Ostrovsky. The (true) complexity of statistical zero knowledge. *STOC*, pages 494–502, 1990.

[BOGG+88] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. *CRYPTO*, LNCS vol. 403:37–56, 1988.

[BP97] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. *EUROCRYPT*, LNCS vol. 1233:480–494, 1997.

[Bra97] S. Brands. Rapid demonstration of linear relations connected by boolean operators. *EUROCRYPT*, LNCS vol. 1233:318–333, 1997.

[BS01] E. Bresson and J. Stern. Efficient revocation in group signatures. *PKC*, LNCS vol. 1992:190–206, 2001.

[BS04] D. Boneh and H. Shacham. Group signatures with verifier-local revocation. *ACM Conference on Computer and Communications Security*, pages 168–177, 2004.

[CCS08] J. Camenisch, R. Chaabouni, and S. Shelat. Efficient protocols for set membership and range proofs. *ASIACRYPT*, LNCS vol. 5350:234–252, 2008.

[CD97] R. Cramer and I. Damgård. Fast and secure immunization against adaptive man-in-the-middle impersonation. *EUROCRYPT*, LNCS vol. 1233:75–87, 1997.

[CD98] R. Cramer and I. Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? *CRYPTO*, LNCS vol. 1462:424–441, 1998.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. *CRYPTO*, LNCS vol. 839:174–187, 1994.

[CF12] G. Di Crescenzo and V. Fedyukovych. Zero-knowledge proofs via polynomial representations. *MFCS*, LNCS vol. 7464:335–347, 2012.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[CHKO12] P. Camacho, Al. Hevia, M. Kiwi, and R. Opazo. Strong accumulators from collision-resistant hashing. *Int. J. Inf. Sec.*, 11(5):349–363, 2012.

[CKS09] J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. *PKC*, LNCS vol. 5443:481–500, 2009.

[CL02] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. *CRYPTO*, LNCS vol. 2442:61–76, 2002.

[Coo66] S. Cook. *On the minimum computation time of functions*. PhD thesis, Department of Mathematics, Harvard University, 1966. `http://cr.yp.to/bib/1966/cook.html`.

[COS86] D. Coppersmith, A. Odlyzko, and R. Schroeppel. Discrete logarithms in gf(p). *Algorithmica*, 1(1):1–15, 1986.

[CP92] D. Chaum and T. Pedersen. Wallet databases with observers. *CRYPTO*, LNCS vol. 740:89–105, 1992.

[CS97] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms, 1997.

[CT65] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comp.*, 19:297–301, 1965.

[Dam93] I. Damgård. Interactive hashing can simplify zero-knowledge protocol design without computational assumptions (extended abstract). *CRYPTO*, LNCS vol. 773:100–109, 1993.

[Dam98] I. Damgård. Commitment schemes and zero-knowledge protocols. *Lectures on Data Security*, pages 63–86, 1998.

[Dam00] I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. *EUROCRYPT*, LNCS vol. 1807:418–430, 2000.

[DGOW95] I. Damgård, O. Goldreich, T. Okamoto, and A. Wigderson. Honest verifier vs dishonest verifier in public coin zero-knowledge proofs. *CRYPTO*, LNCS vol. 963:325–338, 1995.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[dHSSV09] S. de Hoogh, B. Schoenmakers, B. Skoric, and J. Villegas. Verifiable rotation of homomorphic encryptions. *Public Key Cryptography*, LNCS vol. 5443:393–410, 2009.

[Die11] C. Diem. On the discrete logarithm problem in class groups of curves. *Math. Comput.*, 80(273):443–475, 2011.

[Din11] R. Dingledine. Tor and circumvention: Lessons learned - (abstract to go with invited talk). *CRYPTO*, LNCS vol. 6841:485–486, 2011.

[DK00] Y. Desmedt and K. Kurosawa. How to break a practical mix and design a new one. *EUROCRYPT*, LNCS vol. 1807:557–572, 2000.

[ElG84] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *CRYPTO*, LNCS vol. 196:10–18, 1984.

[FFS88] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, 1988.

[FMM+02] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. *Financial Cryptography*, LNCS vol. 2357:16–30, 2002.

[FMS10] J. Furukawa, K. Mori, and K. Sako. An implementation of a mix-net based network voting scheme and its use in a private organization. *Towards Trustworthy Elections*, LNCS vol. 6000:141–154, 2010.

[FO97] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. *CRYPTO*, LNCS vol. 1294:16–30, 1997.

[For87] L. Fortnow. The complexity of perfect zero-knowledge (extended abstract). *STOC*, pages 204–209, 1987.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. *CRYPTO*, LNCS vol. 263:186–194, 1986.

[FS89] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. *CRYPTO*, LNCS vol. 435:526–544, 1989.

[FS01] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. *CRYPTO*, LNCS vol. 2139:368–387, 2001.

[Fur05] J. Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE Transactions*, 88-A(1):172–188, 2005.

[Gau00] P. Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. *EUROCRYPT*, LNCS vol. 1807:19–34, 2000.

[Gau09] P. Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symb. Comput.*, 44(12):1690–1702, 2009.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. *EUROCRYPT*, LNCS vol. 7881:626–645, 2013.

[GI08] J. Groth and Y. Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. *EUROCRYPT*, LNCS vol. 4965:379–396, 2008.

[GK90] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *ICALP*, LNCS vol. 443:268–282, 1990.

[GK96] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for np. *J. Cryptology*, 9(3):167–190, 1996.

[GKR08] S. Goldwasser, Y. Kalai, and G. Rothblum. Delegating computation: interactive proofs for muggles. *STOC*, pages 113–122, 2008.

[GL07] J. Groth and S. Lu. Verifiable shuffle of large size ciphertexts. *PKC*, LNCS vol. 4450:377–392, 2007.

[GMP11] GMP. Gmp library. `http://gmplib.org/`, 2011.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). *STOC*, pages 291–304, 1985.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18:186–208, 1989.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. *CRYPTO*, LNCS vol. 263:171–185, 1986.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity for all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[GMY06] J. Garay, P. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. *J. Cryptology*, 19(2):169–209, 2006.

[Gol10] O. Goldreich. A short tutorial of zero-knowledge. `http://www.wisdom.weizmann.ac.il/~oded/PS/zk-tut10.ps`, 2010.

[Gro04] J. Groth. Honest verifier zero-knowledge arguments applied. *Dissertation Series DS-04-3, BRICS, 2004. PhD thesis. xii+119*, 2004.

[Gro09] J. Groth. Linear algebra with sub-linear zero-knowledge arguments. *CRYPTO*, LNCS vol. 5677:192–208, 2009.

[Gro10] J. Groth. A verifiable secret shuffle of homomorphic encryptions. *J. Cryptology*, 23(4):546–579, 2010.

[Gro11] J. Groth. Efficient zero-knowledge arguments from two-tiered homomorphic commitments. *ASIACRYPT*, LNCS vol. 7073:431–448, 2011.

[GS86] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *STOC*, pages 59–68, 1986.

[GSV98] O. Goldreich, A. Sahai, and S. Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. *STOC*, pages 399–408, 1998.

[HG11] R. Henry and I. Goldberg. Extending nymble-like systems. *IEEE Symposium on Security and Privacy*, pages 523–537, 2011.

[HHG10] R. Henry, K. Henry, and I. Goldberg. Making a nymbler nymble using verbs. *Privacy Enhancing Technologies*, LNCS vol. 6205:111–129, 2010.

[IKOS07] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. *STOC*, pages 21–30, 2007.

[JJ01] M. Jakobsson and A. Juels. An optimally robust hybrid mix network. *PODC*, pages 284–292, 2001.

[JJR02] M. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. *USENIX Security Symposium*, pages 339–353, 2002.

[JKTS07] P. Johnson, A. Kapadia, P. Tsang, and S. Smith. Nymble: Anonymous ip-address blocking. *Privacy Enhancing Technologies*, pages 113–133, 2007.

[JL00] S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. *EUROCRYPT*, LNCS vol. 2656:221–242, 2000.

[Joh00] D. Johnson. Challenges for theoretical computer scienc. `http://www2.research.att.com/~dsj/nsflist.html\#Crypto`, 2000.

[Jou13] A. Joux. A new index calculus algorithm with complexity l(1/4+o(1)) in very small characteristic. *IACR Cryptology ePrint Archive*, 2013:95, 2013.

[Kat12] J. Katz. Which languages have 4-round zero-knowledge proofs? *J. Cryptology*, 25(1):41–56, 2012.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). *STOC*, pages 723–732, 1992.

[Kil95] J. Kilian. Improved efficient arguments. *CRYPTO*, LNCS vol. 963:311–324, 1995.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Dokl.*, 7:595–596, 1963.

[KZG10] A. Kate, G. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. *ASIACRYPT*, LNCS vol. 6477:177–194, 2010.

[Len04] A. Lenstra. Key length, 2004.

[LH11] P. Lofgren and N. Hopper. Bnymble: More anonymous blacklisting at almost no cost (a short paper). *Financial Cryptography*, LNCS vol. 7035:268–275, 2011.

[Lim00] C. Lim. Efficient multi-exponentiation and application to batch verification of digital signatures. `http://dasan.sejong.ac.kr/~chlim/pub/multi_exp.ps`, 2000.

[Lin03] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *J. Cryptology*, 16(3):143–184, 2003.

[Lin10] Y. Lindell. A note on constant-round zero-knowledge proofs of knowledge. Cryptology ePrint Archive, Report 2010/656, 2010. `http://eprint.iacr.org/`.

[LL90] A. Lenstra and H. Lenstra. Algorithms in number theory. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 673–716. Elsevier and MIT Press, 1990.

[LL93] A. Lenstra and H. Lenstra. *The development of the number field sieve*. Springer-Verlag, 1993.

[LLX07] J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. *ACNS*, LNCS vol. 4521:253–269, 2007.

[Lou13] P. Louridas. Personal communication. 2013.

[LPY12] B. Libert, T. Peters, and M. Yung. Scalable group signatures with revocation. *EUROCRYPT*, LNCS vol. 7327:609–627, 2012.

[LV01] A. Lenstra and E. Verheul. Selecting cryptographic key sizes. *J. Cryptology*, 14(4):255–293, 2001.

[Mau95] U. Maurer. Fast generation of prime numbers and secure public-k cryptographic parameters. *J. Cryptology*, 8(3):123–155, 1995.

[McC90] K. McCurley. The discrete logarithm problem. *Proc.of Symposia*, 42:49–74, 1990.

[MH96] M. Michels and P. Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. *ASIACRYPT*, LNCS vol. 1163:125–132, 1996.

[Mil85] V. Miller. Use of elliptic curves in cryptography. *CRYPTO*, LNCS vol. 218:417–426, 1985.

[NBMV99] K. Nguyen, F. Bao, Y. Mu, and V. Varadharajan. Zero-knowledge proofs of possession of digital signatures and its applications. *ICICS*, LNCS vol. 1726:103–118, 1999.

[Nef01] A. Neff. A verifiable secret shuffle and its application to e-voting. *ACM CCS*, pages 116–125, 2001.

[NF05] T. Nakanishi and N. Funabiki. Verifier-local revocation group signature schemes with backward unlinkability from bilinear maps. *ASIACRYPT*, LNCS vol. 3788:533–548, 2005.

[NF06] T. Nakanishi and N. Funabiki. A short verifier-local revocation group signature scheme with backward unlinkability. *IWSEC*, LNCS vol. 4266:17–32, 2006.

[Ngu05] L. Nguyen. Accumulators from bilinear pairings and applications. *CT-RSA*, LNCS vol. 3376:275–292, 2005.

[NNL01] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. *CRYPTO*, LNCS vol. 2139:41–62, 2001.

[NOV06] M. Nguyen, S. Ong, and S. Vadhan. Statistical zero-knowledge arguments for np from any one-way function. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(075):3–14, 2006.

[NOVY92] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung. Perfect zero-knowledge arguments for np can be based on general complexity assumptions (extended abstract). *CRYPTO*, LNCS vol. 740:196–214, 1992.

[oEiCI12] European Network of Excellence in Cryptology II. ECRYPT II yearly report on algorithms and key lengths. `http://www.ecrypt.eu.org/`, 2012.

[Oka96] T. Okamoto. On relationships between statistical zero-knowledge proofs. *STOC*, pages 649–658, 1996.

[OV08] S. Ong and S. Vadhan. An equivalence between zero knowledge and commitments. *TCC*, LNCS vol. 4948:482–500, 2008.

[OVY93] R. Ostrovsky, R. Venkatesan, and M. Yung. Interactive hashing simplifies zero-knowledge protocol design. *EUROCRYPT*, LNCS vol. 765:267–273, 1993.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *EUROCRYPT*, LNCS vol. 1592:223–238, 1999.

[PB10] K. Peng and F. Bao. Improving applicability, efficiency and security of non-membership proof. *Data, Privacy, and E-Commerce, International Symposium on*, pages 39–44, 2010.

[PBDV04] K. Peng, C. Boyd, E. Dawson, and K. Viswanathan. A correct, private, and efficient mix network. *PKC*, LNCS vol. 2947:439–454, 2004.

[Ped91] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. *CRYPTO*, LNCS vol. 576:129–140, 1991.

[Pen11] K. Peng. A general, flexible and efficient proof of inclusion and exclusion. *CT-RSA*, 6558:33–48, 2011.

[PH78] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Trans. Inf. Theor.*, 24:106–110, 1978.

[PIK94] C. Park, K. Itoh, and K. Kurosawa. All/nothing election scheme and anonymous channel. *EUROCRYPT*, LNCS vol. 765:248–259, 1994.

[Poc14] H. Pocklington. The determination of the prime or composite nature of large numbers by fermat's theorem. *Proceedings of the Cambridge Philosphical Societ*, 18:29–30, 1914.

[Pol78] J. Pollard. Monte carlo methods for index computation mod p. *Math. Comp.*, 32:918–924, 1978.

[Pom87] C. Pomerance. Fast, rigorous factorization and discrete logarithm algorithms. *Proc. of the Japan-U.S. Joint Seminar*, pages 119–143, 1987.

[PP90] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of MIXes. *EUROCRYPT*, LNCS vol. 434:373–381, 1990.

[PTW09] R. Pass, W. Tseng, and D. Wikström. On the composition of public-coin zero-knowledge protocols. *CRYPTO*, LNCS vol. 5677:160–176, 2009.

[PV10] R. Pass and M. Venkitasubramaniam. Private coins versus public coins in zero-knowledge proof systems. *TCC*, LNCS vol. 5978:588–605, 2010.

[Sch91] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[Sch08] O. Schirokauer. The impact of the number field sieve on the discrete logarithm problem in finite fields. *MSRI Publications*, 44:397–420, 2008.

[Scy13] Scy, 2013.

[Seo11] J. Seo. Round-efficient sub-linear zero-knowledge arguments for linear algebra. *PKC*, LNCS vol. 6571:387–402, 2011.

[Sha71] D. Shanks. Class number, a theory of factorization, and genera. *Proc. Symp. Pure Math.*, 20:415–440, 1971.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. *EUROCRYPT*, LNCS vol. 1233:256–266, 1997.

[Sho09] V. Shoup. Ntl library. `http://www.shoup.net/ntl/`, 2009.

[SK95] K. Sako and J. Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. *EUROCRYPT*, LNCS vol. 921:393–403, 1995.

[Son01] D. Song. Practical forward secure group signature schemes. *ACM Conference on Computer and Communications Security*, pages 225–234, 2001.

[SS98] J. Silverman and J. Suzuki. Elliptic curve discrete logarithms and the index calculus. *ASIACRYPT*, LNCS vol. 1514:110–125, 1998.

[Sta05] H. Stamer. Efficient electronic gambling: An extended implementation of the toolbox for mental card games. *WEWoRC*, P-74:1–12, 2005.

[TKCS11] P. Tsang, A. Kapadia, C. Cornelius, and S. Smith. Nymble: Blocking misbehaving users in anonymizing networks. *IEEE Trans. Dependable Sec. Comput.*, 8(2):256–269, 2011.

[Too00] A. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. `http://www.de.ufpe.br/~toom/my_articles/engmat/MULT-E.PDF`, 2000.

[Tor13] Tor. The tor project inc. `https://www.torproject.org/`, 2013.

[TW87] M. Tompa and H. Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. *FOCS*, pages 472–482, 1987.

[TW10] B. Terelius and D. Wikström. Proofs of restricted shuffles. *AFRICACRYPT*, LNCS vol. 6055:100–113, 2010.

[TX03] G. Tsudik and S. Xu. Accumulating composites and improved group signing. *ASIACRYPT*, LNCS vol. 2894:269–286, 2003.

[Vad06] S. Vadhan. An unconditional study of computational zero knowledge. *SIAM J. Comput.*, 36(4):1160–1214, 2006.

[Wak68] A. Waksman. Corrigendum: “a permutation network”. *J. ACM*, 15(2):340, 1968.

[Wik02] D. Wikström. The security of a mix-center based on a semantically secure cryptosystem. *INDOCRYPT*, LNCS vol. 2551:368–381, 2002.

[Wik09] D. Wikström. A commitment-consistent proof of a shuffle. *ACISP*, LNCS vol. 5594:407–421, 2009.

[Wik10] D. Wikström. Verificatum. `http://www.verificatum.com/`, 2010.

[WWP07] P. Wang, H. Wang, and J. Pieprzyk. A new dynamic accumulator for batch updates. *ICICS*, LNCS vol. 4861:98–112, 2007.

[YYC+12] K. Yu, T. Yuen, S. Chow, S. Yiu, and L. Hui. Pe(ar)2: Privacy-enhanced anonymous authentication with reputation and revocation. *ESORICS*, LNCS vol. 7459:679–696, 2012.

[Zeu13] Zeus. https://github.com/grnet/zeus, 2013.

[ZL06] S. Zhou and D. Lin. Shorter verifier-local revocation group signatures from bilinear maps. *CANS*, LNCS vol. 4301:126–143, 2006.