Master's Thesis

Efficient Zero-Knowledge Multi-Key Verifiable Shuffles: Implementation, Security and Applications

Aaron Kimmig

Examiner:	Prof. Dr. Christian Schindelhauer
Second Examiner:	Prof. Dr. Jörn Müller-Quade (Karlsruher Institut für Technologie)

University of Freiburg · Faculty of Engineering

Department of Computer Science

Chair for Computer Networks and Telematics

Prof. Dr. Christian Schindelhauer

December 11th, 2023

Writing Period

July 13$^{\text{th}}$ – December 11$^{\text{th}}$, 2023

Examiner

Prof. Dr. Christian Schindelhauer

Second Examiner

Prof. Dr. Jörn Müller-Quade (Karlsruher Institut für Technologie)

Master Thesis

Effizienter verifizierbarer Nullwissen-Mischer mit mehreren Schlüsseln: Implementierung, Sicherheit und Anwendungen

Aaron Kimmig

Gutachter:	Prof. Dr. Christian Schindelhauer
Zweitgutachter:	Prof. Dr. Jörn Müller-Quade (Karlsruher Institut für Technologie)

Albert-Ludwigs-Universität Freiburg · Technische Fakultät

Institut für Informatik

Lehrstuhl für Rechnernetze und Telematik

Prof. Dr. Christian Schindelhauer

11. Dezember 2023

Bearbeitungszeit

13. Juli – 11. Dezember 2023

Gutachter

Prof. Dr. Christian Schindelhauer

Zweitgutachter

Prof. Dr. Jörn Müller-Quade (Karlsruher Institut für Technologie)

Eidesstattliche Erklärung zur Eigenständigen Erstellung dieser Abschlussarbeit

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Des Weiteren erkläre ich, dass diese Arbeit weder als Ganzes noch in Teilen aus durch künstliche Intelligenz erzeugtem Text oder Programmcode besteht oder davon abgeleitet wurde.

Freiburg, 11. Dezember 23

Ort, Datum

Unterschrift

Abstract

This master's thesis explores efficient zero-knowledge multi-key verifiable shuffles, their implementation, security and applications. The multi-key shuffle allows a shuffler to perform a secret permutation and re-encryption on ElGamal ciphertexts. These ciphertexts are sent by users of the system ("senders") to some other users ("receivers") together with their public keys. The shuffler proves the correctness of the shuffle-and-re-encrypt operation in a zero-knowledge manner. Concatenation of such multi-key shuffles forms a bidirectional mix network.

Bünz et al. have presented multi-key verifiable shuffles in their recent paper draft "Efficient Multi-Key Verifiable Shuffles from Short Arguments for Randomized Algorithms" (2023) which can be retrieved from https://scholar.harvard.edu/files/jsarathy/files/bp_shuffle_draft_4.pdf. In this thesis, we complete their protocol with an auxiliary zero-knowledge protocol for proving power relations between secret Pedersen committed numbers, we formally prove its zero-knowledge properties, and present an interactive proof-of-concept implementation.

We explore the standard practical use-case of the multi-key shuffle, namely a public anonymous bulletin board with equally anonymous reply channels, and the limitations of the multi-key shuffle. We do not implement a masked single-key shuffle as used in mental card games where each ciphertext to be permuted is encrypted by the same key. We show how a naive approach on changing the multi-key shuffle into a single-key shuffle fails. Finally, we discuss the security properties of the zero-knowledge protocol and some common security issues that arise when implementing zero-knowledge protocols.

Contents

1. Introduction

This master's thesis explores a zero-knowledge protocol that realizes provably anonymous communication channels with a zero-trust cryptographic setup. This is a novelty in the field of cryptography and therefore drew the attention of the author.

We verify and implement log-sized zero-knowledge proofs for multi-key verifiable shuffles as proposed by Bünz et al. in their paper draft [1] and explore their potential applications like bidirectional mix networks which can provide anonymous communication channels.

We will see that the design of the protocol prevents its direct application in mental card games. While Bünz et al. do not state if the multi-key shuffle could be used to implement masked single-key shuffles – as they are needed for mental card games – our naive approaches to transform it into a masked single-key shuffle protocol fail.

To make the multi-key shuffle work, some errata in Bünz et al.'s draft need to be fixed and some omissions need to be filled in. A major missing part is a protocol to prove an n-th power relation between two secret numbers in a zero-knowledge manner. We formally prove its correctness, and integrate it into the main multi-key shuffle argument that has already been provided in the draft.

The multi-key verifiable shuffle has been implemented as an interactive, configurable, and documented proof-of-concept network application. We recommend reading the prover's and verifier's transcript of an argument alongside this thesis which can be found as attachments to it.

2. Related Work

2.1. Zero-Knowledge Proofs

In 1985, Goldwasser et al. introduced zero-knowledge proofs [2]. With zero-knowledge proofs, a prover $\mathcal{P}$ convinces a verifier $\mathcal{V}$ that she knows some secret information satisfying certain properties without revealing anything about this information to the verifier other than that. The communication which occurs between $\mathcal{P}$ and $\mathcal{V}$ forms the proof and $\mathcal{P}$'s secret information is the witness.

Zero-Knowledge Arguments. In 1992 Kilian proved that the communication cost of zero-knowledge arguments can be smaller than the witness size [3]. Zero-knowledge arguments have computational soundness in contrast to zero-knowledge proofs which have statistical soundness. Nowadays, the term proof is often used to describe zero-knowledge arguments as well. This also applies to the paper this thesis is based on.

Discrete Logarithm Assumption. The first zero-knowledge protocol that was based on the discrete logarithm assumption is Schnorr's identification protocol [4]. Today, most zero-knowledge arguments are based on this assumption, and so is Bünz et al.'s multi-key verifiable shuffle.

Efficient Proofs. Kilian's findings motivated the search for protocols with shorter arguments as well as optimized prover and verifier times for basic arguments like zero-knowledge range checks, shuffles, and more generally, arbitrary arithmetic circuits. In 2018, Bünz et al. achieved proof sizes logarithmic in the length of the input for range

arguments and logarithmic in the size of the circuit for deterministic arithmetic circuits with Bulletproofs [5]. In 2019, Hoffmann et al. achieved improvements on Bulletproofs [6] and recently in 2022, the efficiency of Bulletproofs range proofs has been optimized further by Eagen et al. [7].

2.2. Shuffle Arguments

Masked Shuffles. Masked (i.e. secret) single-key shuffles are part of Schindelhauer's 1998 toolbox for mental card games [8] which Barnett and Smart improved in 2003 [9] which in turn was implemented by Stamer in 2005 [10]. Practical problems like electronic voting can be simulated by mental card games, where masked single-key shuffles are also the bottleneck for smaller proof sizes. Bayer and Groth achieved $\mathcal{O}(\sqrt{n})$ proof size for those masked shuffles in 2012 [11] based on previous work by Groth [12] [13]. Hoffmann et al. reduced it to $\mathcal{O}(\log n)$ in 2019 [6]. Bünz et al. suggested that proof sizes of $\mathcal{O}(\log n)$ for verifiable shuffles are possible in [5] by implementing a sorting circuit. However, this does not fit the setting of masked shuffles with ElGamal encrypted values, which is required for mental card games. Hoffmann et al. [6] mention this problem in Appendix C remark C.1 and state that it is not obvious to them how Bulletproofs could be modified to work with ElGamal encrypted values instead of Pedersen committed values. This problem is explored in detail in section 4.8 with regards to the multi-key shuffle.

Mix nets. Chaum introduced mix nets that allow sending messages through unidirectional anonymous channels in 1981 [14]. As far as we know Bünz et al.'s multi-key verifiable shuffle is the first realization of an inherently bidirectional mix net. Until then, only unidirectional mix nets were known. One example of such a mix net is the decryption mix net which first lets its users encrypt their (cleartext or ciphertext) messages by all public keys of the mixing layers, then routes them through those layers which step-by-step shuffle and decrypt the messages until finally the original messages appear in a random order; see figure 1. Leveraging the power of zero-knowledge proofs, this

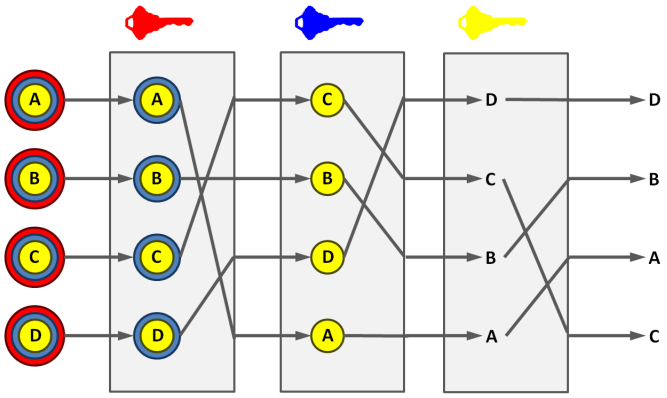

Figure 1.: Decryption mix net of depth 3 with 4 message channels
Illustration by *Primepq from English Wikipedia*, licensed under CC BY-SA 3.0

paradigm is taken to the next level by the multi-key shuffle.

Multi-Key Verifiable Shuffles. A multi-Key verifiable shuffle mixes and re-encrypts ElGamal ciphertexts alongside the public keys of their authors (to make answering to the messages possible for the recipient) and the public keys of their receivers (to allow the receiver to efficiently identify their messages). Each multi-key verifiable shuffle stage is accompanied by a zero-knowledge protocol that proves the correctness of the shuffle, i.e. the shuffle and re-encryption operation has been performed honestly.

Randomized Arithmetic Circuits. Randomized arithmetic circuits are a generalization of Bulletproofs arithmetic circuits that allow for smaller proof sizes by a constant factor for some randomized algorithms. They were introduced by Bünz et al. in [1] after the multi-key verifiable shuffle which is just an example of a randomized arithmetic circuit.

Due to the complexity of the multi-key verifiable shuffle alone, we do not look further into the potential of randomized arithmetic circuits and if and how these could be used to implement masked shuffles.

2.3. Contribution

This thesis contributes a completion, proof-of-concept implementation, verification and resources to help understand the multi-key verifiable shuffle proposed by Bünz et al. [1]. For this, we present the theoretical foundations to describe and prove in detail the correctness of an efficient zero-knowledge protocol for efficient proofs of an n-th power relation of two Pedersen committed secret numbers based on the Bulletproofs paradigm. This power relation protocol is not given in [1] but is needed as a sub protocol for the multi-key shuffle.

3. Background

We follow the notation in Bünz et al.'s draft on multi-key verifiable shuffles and recapitulate the definitions they provide in sections 3.2 to 3.5. We recommend reading that draft before this thesis or as an alternative, read the notebook transcripts of the proof-of-concept implementation alongside this thesis.

3.1. Notation

Our notation uses standards from zero-knowledge cryptography and differs in the following points from [1]: Throughout this thesis we use the same notation standards as [1], section *3.1 Preliminaries/Notation* with small differences.

- Variables V and W are swapped, such that V commits to secret shift s and W commits to s^n.
- Denote the sum of all entries of $\boldsymbol{k}$ with $\bar{k}$
- Variables exclusive to the power relation protocol PR are subscripted by W like in α_W or $\boldsymbol{a}_{R,W}$.
- The prime order of the finite cyclic group $\mathbb{G}$ is denoted by q. In the implementation, the prime modulus for elliptic curves over finite fields $\mathbb{F}_p$ resp. discrete exponentiation is denoted by p. Random challenges are sampled from $\mathbb{Z}_q^*$.
- For consistency, $\boldsymbol{y}_z^{n-2}$ is renamed to $\boldsymbol{y}_z^{n-1}$.
- ElGamal ciphertext randomness is renamed from $\boldsymbol{r}$ to $\boldsymbol{\gamma}$ to avoid name collision with the name for the vector of the right side of the inner product statement $\boldsymbol{r}$.

Let us explore some commonly used expressions:

- $a \leftarrow\!\!\$\ \mathbb{F}$: Uniformly sample some element a from the set $\mathbb{F}$ at random.

- $negl(\lambda)$: A negligible probability depending on the security parameter λ. λ usually is derived from the key length and the cryptosystem being used. What is considered an actual negligible probability has to be defined by the security policies of an implementation.

- Setup: The $\mathsf{Setup}(1^\lambda)$ algorithm is a publicly defined routine that derives the prime order cyclic group of ciphertexts $\mathbb{G}$, the prime modulus p and the generators g, $\boldsymbol{g}$, $\boldsymbol{h}$ and possibly other constant values the protocol depends on from the security parameter λ.

- *NUMS*: The acronym *NUMS* is derived from "nothing up my sleeve". A set of generators of $\mathbb{G}$ is called *NUMS* generators if it is chosen such that it can be trusted that nobody knows a discrete log relation between them. This can be achieved by selecting them with a cryptographic pseudorandom number generator that is initialized with a fixed random seed.

- CRS: CRS is short for the common reference string which in the context of this thesis consists of the user generated ciphertexts to be shuffled by the multi-key verifiable shuffle.

Let $g \in \mathbb{G}$, $\boldsymbol{g}, \boldsymbol{h} \in \mathbb{G}^n$, $s \in \mathbb{Z}_q$, $\boldsymbol{x} \in \mathbb{Z}_q^n$. Recall the notation for scaling and exponentiating vectors of scalars and group elements: $s \cdot \boldsymbol{x} = (s \cdot x_1, \ldots, s \cdot x_n)$, $g^{\boldsymbol{x}} = (g^{x_1}, \ldots, g^{x_n})$, $\boldsymbol{g}^s = (g_1^s, \ldots, g_n^s)$ and $\boldsymbol{g}^{\boldsymbol{x}} = \prod_{i=1}^n g_i^{x_i}$.

The implementation of the multi-key shuffle uses a variant of this notation where vectors are marked with an overscript arrow instead of bold font. It also uses elliptic curve cryptography and therefore additive notation. For example, the expression $\boldsymbol{g} \cdot \boldsymbol{h}$ is

transformed into $\overrightarrow{g} + \overrightarrow{h}$ (code and documentation) and $\boldsymbol{g}^{\boldsymbol{a}_L}$ is transformed into $\overrightarrow{a_L} \cdot \overrightarrow{g} = \overrightarrow{g} \cdot \overrightarrow{a_L}$ (code) and $\langle \overrightarrow{a_L}, \overrightarrow{g} \rangle = \langle \overrightarrow{g}, \overrightarrow{a_L} \rangle$ (documentation).

For short, we call the multi-key shuffle MKS and the power relation arguments PR.

3.2. Assumptions

The zero-knowledge properties of MKS (multi-key shuffle argument) and PR (power relation argument) are based on the discrete logarithm assumption and the equivalent discrete logarithm relation:

Discrete Logarithm. Given a publicly known generator g of a finite cyclic group $\mathbb{G}$ of prime order q with security parameter λ and a randomly chosen element h of this group, there is no algorithm that can choose the discrete logarithm a that fulfills $g^a = h$ in a reasonable time. Formally: For all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that

$$\begin{aligned} &\text{given } (\mathbb{G}, q, g) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda), h \leftarrow\!\!\$\ \mathbb{G} : \\ &\Pr\left[\mathcal{A}(\mathbb{G}, q, g, h) = a, \text{ where } g^a = h\right] \leq negl(\lambda) \end{aligned}$$

Discrete Logarithm Relation. Given a finite cyclic group $\mathbb{G}$ of prime order q with security parameter λ and some randomly chosen elements of this group, there is no algorithm that can find a non-trivial relation, i.e. exponents, between them. Formally: For all $n \geq 2$ and for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that

$$\begin{aligned} &\text{given } \mathbb{G} \leftarrow \mathsf{Setup}(1^\lambda), g_1, \ldots, g_n \leftarrow\!\!\$\ \mathbb{G} : \\ &\Pr\left[(\mathbb{G}, g_1, \ldots, g_n) = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n \text{ where } \prod_{i=1}^{n} g^{a_i} = 1_\mathbb{G} \ \wedge\ \exists i : a_i \neq 0\right] \leq negl(\lambda) \end{aligned}$$

Subversion of the Common Reference String (CRS). It is required for the multi-key verifiable shuffle that the prover does not subvert the CRS: The prover must not compromise soundness of Pedersen commitments by creating generators which she knows a discrete logarithm relation for. Therefore it is also assumed that there is no collusion between the prover (shuffler) and the users of the multi-key verifiable shuffle system. This assumption implies that no discrete logarithm relation is known for the CRS. Without this assumption a trusted setup would become a necessity. By some individuals and institutions, a trusted setup is a red flag for their zero-knowledge cryptography production applications.

Fiat-Shamir transform. The Fiat-Shamir transform presented by named authors in 1986 [15] is a heuristic based on the random oracle model which removes interactivity from a public coin zero-knowledge argument system. The random oracle model is usually realized by applying a cryptographic hash function to all messages the prover would have sent to the verifier until the prover expects a random challenge from the verifier. The resulting hash value is used as substitute for the random challenge. Subsequent random challenges are generated by applying the hash function to the last random challenge returned by the transform and all messages that the prover would have sent to the verifier after that.

In the following, we assume the Fiat-Shamir transform to be secure to remove interactivity from zero-knowledge protocols.

3.3. Definitions

This thesis depends on the common terms that are used in the domain of zero-knowledge proofs which we will recapitulate in this section. For all formal definitions we refer to [1], sections 3.3 and 3.4. [11] uses similar definitions.

Let public parameters $(\mathbb{G}, q, g, h, \boldsymbol{g})$, where the discrete logarithm relation between group elements $g, h, \boldsymbol{g}$ is unknown, be given.

Pedersen Commitments. The Pedersen commitment scheme is a *homomorphic, perfectly hiding* and *computationally binding*. Pedersen commitments are used in the multi-key shuffle and power relation arguments by the prover to commit to secrets towards the verifier. For example, the prover commits to some secret scalars $a, b \in \mathbb{Z}_q$ and some secret vector $\boldsymbol{v} \in \mathbb{Z}_q^n$ by

$$\begin{aligned}
&\alpha, \beta \leftarrow\!\!\$\, \mathbb{Z}_q \\
&A = g^a \cdot h^\alpha \in \mathbb{G} \\
&B = g^b \cdot h^\beta \in \mathbb{G} \\
&\gamma \leftarrow\!\!\$\, \mathbb{Z}_q \\
&V = g^{\boldsymbol{v}} \cdot h^\gamma = h^\gamma \prod_{i=1}^{n} g_i^{v_i} \in \mathbb{G}
\end{aligned}$$

and sends the Pedersen commitments A and V to the verifier while keeping $a, \alpha, b, \beta, \boldsymbol{v}, \gamma$ secret.

Pedersen scalar commitments are

- *homomorphic* because of the distributivity of exponents that contain the secrets:

$$\begin{aligned}
A \cdot B &= (g^a h^\alpha) \cdot (g^b h^\beta) \\
&= g^{a+b} \cdot h^{\alpha+\beta}
\end{aligned}$$

- *perfectly hiding:* Knowing only A, it is impossible even for a computationally unbounded adversary to determine which value of a the prover committed to because the blinding factor h^α which is also unknown to the adversary could take any value in $\mathbb{G}$, hence can never be divided out of the commitment A.
- *computationally binding:* By contradiction, after committing to A, it is infeasible for the prover to alter a and α into some $a' \neq a$ and α' such that $g^{a'} \cdot h^{\alpha'} = A$ because this means a non-trivial discrete log relation between g and h, $g^{a-a'} h^{\alpha-\alpha'} = 1_\mathbb{G}$, has been found which is a contradiction to the discrete logarithm assumption.

Arguing equivalently, Pedersen vector commitments are *homomorphic, perfectly hiding*

and *computationally binding.*

Commitments without blinding factors to secrets that are sampled uniformly at random from $\mathbb{G}$ are *homomorphic, computationally hiding* and *perfectly binding.* If the secrets are not uniformly sampled, the hiding property is lost.

Zero-Knowledge Arguments. With a zero-knowledge argument of knowledge a computationally bound prover $\mathcal{P}$ convinces a verifier $\mathcal{V}$ that, given computational hardness assumptions and the output σ of the common Setup algorithm, she has a witness w fulfilling a statement u which can be represented by a polynomial-time relation $\mathcal{R}$ without revealing the witness.
The algorithms $\mathsf{Setup}, \mathcal{P}$ and $\mathcal{V}$ form an argument of knowledge for a relation $\mathcal{R}$. In composition, they have *perfect completeness* and *computational witness-extended emulation.*

Public Coin. The protocols MKS and PR are both *public coin*, meaning the verifier's random challenges are made public, chosen uniformly at random and independently from the messages sent by the prover.

Perfect Completeness. $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ has perfect completeness if $\mathcal{V}$ only accepts proofs generated by the protocol if and only if $(\sigma, u, w) \in \mathcal{R}$, or more formally, if for all adversaries $\mathcal{A}$ with non-uniform polynomial-time (*non-uniform* means $\mathcal{A}$ can reproduce $\mathcal{V}$'s random challenges with the help of a polynomial length advice string additionally given to it),

$$\sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w) \leftarrow \mathcal{A}(\sigma) :$$
$$(\sigma, u, w) \in \mathcal{R} \Leftrightarrow \mathcal{V} \text{ accepts argument of } \mathcal{P}(\sigma, u, w) \text{ and } \mathcal{V}(\sigma, u)$$

Computational Witness-Extended Emulation. Computational witness-extended emulation is a notion of soundness (like "soundness" in Pedersen commitments) adapted for longer zero-knowledge protocols. Its formal definition is very technical, so we describe it intuitively: $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ has computational witness-extended emulation if, whenever

an argument that can be efficiently generated and is accepted by the verifier with some probability, there exists an efficient extractor which generates the same argument but additionally computes a witness for that argument with almost the same probability. The extractor is assumed to have interaction rewinding capabilities (keeping state of $\mathcal{P}$ while refreshing random state of $\mathcal{V}$), which it uses to compute the witness.

Perfect Special Honest-Verifier Zero-Knowledge (SHVZK). Informally, an argument of knowledge is SHVZK if the probability distribution of the messages sent between $\mathcal{P}$ and $\mathcal{V}$ can be simulated efficiently without knowing the witness when the verifier's challenge values are known in advance and the verifier's challenges are chosen at random.

Subversion Zero-Knowledge. If an argument of knowledge remains SHVZK under subversion of the common reference string (CRS) σ, it is subversion (perfect special honest-verifier) zero knowledge. Given the assumption that the prover does not subvert the CRS, a trivial check that all group elements e of the CRS satisfy $e \neq 1_{\mathbb{G}}$ ensures Pedersen commitments remain sound (proof in appendix A of [1]).

Multi-Key Verifiable Shuffle. The multi-key verifiable shuffle is explained in section 1, Technical Overview, Multi-Key Verifiable Shuffle, and more formally in sections 3.5 to 4.2 of [1]. A multi-key verifiable shuffle permutes (by a secret permutation π) and re-keys respectively re-encrypts (both with the same secret shift s) an input list of triples containing a public receiver key (of which the intended receiver knows the private key), an ElGamal ciphertext (encrypted under the receiver's public key), and a public reply key (of which the sender knows the private key), to output a list of triples of new public receiver keys, ElGamal ciphertexts and public reply keys which are unlinkable to the sender's identities and main public keys.

Note that, in our understanding, Bünz et al.'s informal description of the multi-key shuffle misses to include the receiver's public keys while their formal definition did not include the reply keys anymore. Without the reply keys the bidirectionality of the mix net is lost. Without the receiver's public keys the intended recipients can not efficiently

identify the ciphertexts that they should decrypt: While recipients may fail to decrypt ciphertexts not intended for them, they still would have to try to extract the message in the exponent for *all* ciphertexts in the shuffle which is very inefficient and not safe from false identification of messages under the same security parameter as the encryption of the ciphertexts. To avoid this, the sender must add the public keys of the message recipients to the input of the shuffle as Bünz et al. propose in their formal description.

Thus, in the multi-key shuffle input, let $\boldsymbol{h} = g^{\boldsymbol{x}}$ be the main public keys of the message receivers ($\boldsymbol{x}$ the private keys), $\boldsymbol{b} = g^{\boldsymbol{\gamma}}$ be the randomizer parts (with secret randomness $\boldsymbol{\gamma}$) of the ElGamal ciphertexts, $\boldsymbol{c} = g^{\boldsymbol{m}}\boldsymbol{h}^{\boldsymbol{\gamma}}$ be the message parts (messages $\boldsymbol{m}$ in the exponent) under the corresponding public keys $\boldsymbol{h}$ of the ElGamal ciphertexts, and $\boldsymbol{d} = g^{\boldsymbol{y}}$ be the public reply keys ($\boldsymbol{y}$ the private keys) where all of $\boldsymbol{h}, \boldsymbol{b}, \boldsymbol{\gamma}, \boldsymbol{c}, \boldsymbol{m}, \boldsymbol{d}, \boldsymbol{y}$ have the same length n which is the size of the shuffle. The pair $(\boldsymbol{b}, \boldsymbol{c})$ forms the ElGamal ciphertext such that the triple $(\boldsymbol{h}, (\boldsymbol{b}, \boldsymbol{c}), \boldsymbol{d})$ forms the triple of the multi-key shuffle.

In the multi-key shuffle argument the shuffler must argue that for inputs $\boldsymbol{h}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ and outputs $\boldsymbol{h}', \boldsymbol{b}', \boldsymbol{c}', \boldsymbol{d}'$ it knows a permutation π and a secret s such that, for all $i \in \{1, \ldots, n\}, h'_i = h^s_{\pi(i)}, b'_i = b^s_{\pi(i)}, c'_i = c^s_{\pi(i)}, d'_i = d^s_{\pi(i)}$. A complete formal definition (without $\boldsymbol{d}$, $\boldsymbol{y}$ and $\boldsymbol{d}'$) is given in section 4.1 Shuffle Primitive of [1].

3.4. Elliptic Curve Cryptography

This thesis' proof-of-concept implementation of MKS which contains PR uses an elliptic curve over a finite field $\mathbb{F}_p$ of prime order p that yields the group $\mathbb{G}$ of prime order q containing the elliptic curve points. A lot of research has been done on finding elliptic curves and representations that can be implemented securely and efficiently on modern hardware for cryptography [16, 17, 18, 19, 20]. Not for maximum security, but for simplicity of implementation and limitations of available libraries, we use the Weierstrass elliptic curve known as secp256k1as specified in [21], section 2.4.1.

To give a hint on how discrete exponentiation in Schnorr groups [4] [22] would be done, we include its setup routine in the implementation. For reasons of clarity and comprehensibility we do not implement more of it – for shuffle sizes of 4 the already very lengthy prover and verifier notebook transcripts would have become longer than 100 pages.

3.5. Bulletproofs

Bulletproofs are a zero-knowledge scheme without trusted setup introduced by Bünz et al. in 2018 [5]. They transform linear constraints on secret values such as range requirements into an inner product relation which they convert into a log-sized zero-knowledge argument by an *inner product argument*. The inner product argument has been implemented as part of efficient MKS; For the mathematical details on the inner product protocol we refer to the original paper.

Bulletproofs, MKS and PR, which the latter is explained in detail in this thesis, utilize established techniques to transform linear constraints into an inner product relation and build complex zero-knowledge arguments:

- compose sub-protocols with single challenges into a larger protocol with multiple challenges (justified by the Schwartz-Zippel Lemma [23, 24, 25] on randomly testing equality of multivariate polynomials),
- probabilistically prove a constraint on a vector of secret values with a single random challenge and a single commitment with a vector of powers of the random challenge, and
- polynomial identity checks with blinding applied

In section 1.1 of [6] Hoffmann et al. showed in detail why these techniques are valid and secure. In the following they may be used without explicitly naming or explaining them again.

3.6. Power Relations

A proof of power relation (PR for short) is a cryptographic protocol which provides a zero-knowledge argument that a secret value s_n is the n-th power of another secret value s. A proof of power relation is part of the MKS protocol presented in [1] where PR has deliberately been omitted by the authors in section 4.3, "Arguing that a_R is well-formed" and "Bulletproofs sub-routine".

Definition: Power Relation Argument System. Let $\mathsf{pp} = (\mathbb{G}, q, g, h)$ be public parameters generated by $\mathsf{Setup}(1^\lambda)$. A power relation argument system PR is an interactive argument system $\langle \mathcal{P}, \mathcal{V} \rangle$ which, on private input $s, s_n \in \mathbb{Z}_q^*, \sigma_V, \sigma_W \in \mathbb{Z}_q$ to the prover $\mathcal{P}$ and public input $V \in \mathbb{G}, W \in \mathbb{G}, n > 0 \in \mathbb{Z}$), convinces the verifier $\mathcal{V}$ that $s_n = s^n$ where $V = g^s \cdot h^{\sigma_V}$ and $W = g^{s_n} \cdot h^{\sigma_W}$ are Pedersen commitments to s resp. s_n.

Definition: Security of PR. PR is secure if it satisfies perfect completeness (proof: 5.4), subversion zero-knowledge (proof: 5.5), and computational witness-extended emulation (proof: 5.6), and if public input to PR satisfies IND-CPA$_{\mathsf{PR}}$ privacy (proof: 5.1), as defined next.

Definition: Privacy of PR. Define privacy in terms of indistinguishability under a chosen plaintext attack (IND-CPA). Informally, this means that input Pedersen commitments V and W have randomized encryption.

We use the security game with security parameter λ

$\text{IND-CPA}^{\mathcal{A}}_{\mathsf{PR}}(n)$

1 : $(\mathbb{G}, q, g, h) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda)$

2 : $s_0, s_1 \in \mathbb{Z}_q^* \leftarrow \mathcal{A}$

3 : $\sigma_{V_0}, \sigma_{W_0}, \sigma_{V_1}, \sigma_{W_1} \leftarrow\!\$\ \mathbb{Z}_q$

4 : $V_0 = g^{s_0} h^{\sigma_{V_0}}$

5 : $W_0 = g^{(s_0^n)} h^{\sigma_{W_0}}$

6 : $V_1 = g^{s_1} h^{\sigma_{V_0}}$

7 : $W_1 = g^{(s_1^n)} h^{\sigma_{W_1}}$

8 : $b \leftarrow\!\$\ \{0, 1\}$

9 : $b' \leftarrow\!\$\ \mathcal{A}(1^\lambda, n, V_b, W_b)$

10 : **return** $b = b'$

to describe privacy for PR.

The adversary wins her game with an advantage of $\mathsf{Adv}^{\text{IND-CPA}_{\mathsf{PR}}}_{\mathcal{A}}(\lambda) = \left|P[b = b'] - \frac{1}{2}\right|$. PR is private if for all interactive adversaries this advantage stays below $negl(\lambda)$.

4. Main Result

4.1. Problem Definition

The main goal of this thesis is to provide an implementation of multi-key verifiable shuffles based on formally verified protocols. Most of the theoretical groundwork was done by Bünz et al. in their paper draft [1] where a power relation protocol with formal description and proof has been left out and some minor imprecisions needed to be fixed. Being new to the field of zero-knowledge cryptography, this was neither a mathematical nor an implementational triviality for the author and therefore accepted as a valuable challenge.

4.2. Notes on the Multi-Key Shuffle Draft

The most important changes and additions we apply to the multi-key shuffle argument given in section 4.3, "Bulletproofs sub-routine" of [1] are:

- correct definitions of $\delta(y, z)$ and t_1
- τ_i and T_i only for $i \in \{3, 5\}$
- consider V and W in definition of τ_x
- corrections in $\mathcal{V}$'s final equality checks
- introduce PR as a standalone zero-knowledge argument and adapt it for MKS such that the n-th power relation between V and W is argued for

- direct integration of Fiat-Shamir transform into the implementation of the interactive protocol
- implementation of a $\mathsf{Setup}(1^\lambda)$ routine enabling a secure Fiat-Shamir transform
- show how the Bulletproofs sub-routine of MKS is connected to the inner product protocol
- add public keys of the receivers $\boldsymbol{d}$ to the CRS and the shuffle argument

4.3. Power Relation Argument

For the multi-key shuffle argument (MKS) the shuffler commits to the secret key shift s by $V = g^s$ and $W = g^{s^n}$. The structure of MKS requires a separate commitment W to s^n. This commitment can not be deduced from V and therefore it has to be proved that W is well-formed with respect to V in addition to the proof of shuffle.

In [1], section 4.3, before subsection "Bulletproofs sub-routine" it is explicitly stated that the power relation protocol has been left out. The power relation protocol (PR) may also be used independently of the multi-key shuffle. It is possible to run PR as a separate protocol. When run as a part of MKS, joint use of challenges reduces proof size which is the reason we integrate PR into the main protocol of MKS.

To ensure IND-CPA security, s must either be a value uniformly sampled from $\mathbb{Z}_q^*$ (as it is the case in MKS) or the commitments V and W be blinded by some $\sigma_V, \sigma_W \leftarrow\!\!\$\, \mathbb{Z}_q$ with $V = g^s h^{\sigma_V}$ resp. $W = g^{(s^n)} h^{\sigma_W}$.

PR is based on the argument for $\boldsymbol{a}_R$ being well-formed in MKS as suggested by Bünz et al. Given n, to argue that some W is a Pedersen commitment to the n-th power of a secret value s which has been committed to with V, the power relation has to be transformed into a Hadamard product of two vectors with linear constraints enforcing those vectors correspond to V (resp. s) and W (resp. s^n), which makes the application of the inner product argument possible. Up to the specific constraints the structure of PR is exactly like the Bulletproofs range proof protocol's, using techniques described

in 3.5. Variables belonging to PR are indexed with a W as in the full MKS protocol implementation:

The prover sets up $\boldsymbol{a}_{L,W}, \boldsymbol{a}_{R,W} \in \mathbb{Z}_q^*$ based on the secret number s

$$\begin{aligned} \boldsymbol{a}_{R,W} &= (s, s, \dots s) \text{ with length } n \\ \boldsymbol{a}_{L,W} &= (1, s, s^2, \dots s^{n-1}) \end{aligned}$$

and commits to them with $A_W = \boldsymbol{g}^{\boldsymbol{a}_{L,W}} \boldsymbol{h}^{\boldsymbol{a}_{R,W}} h^{\alpha_W}$ using random blinding value α_W.

With linear constraints

$$\begin{aligned} a_{R,W_1} &= s \\ \forall i \in \{1, \dots, n-1\} : a_{R,W_i} &= a_{R,W_{i+1}} \\ a_{L,W_1} &= 1 \\ \forall i \in \{1, \dots, n-1\} := a_{L,W_i} a_{R,W_i} &= a_{L,W_{i+1}} \end{aligned}$$

the prover argues that $a_{L,W_1} a_{R,W_1} = s$ and $a_{L,W_n} a_{R,W_n} = s^n$. Linking V to the first and W to the second of those products enables to prove their required n-th power relation in a Bulletproofs sub-routine.

For the second constraint, which argues that all entries of $\boldsymbol{a}_{R,W}$ are equal, a random challenge w is needed in addition to y and z.

To link V and W to the linear constraints, $\boldsymbol{a}_{L,W}$ and $\boldsymbol{a}_{R,W}$ have to be linearly combined with challenges from the verifier such that the inner product of resulting $\boldsymbol{l}_W$ resp. $\boldsymbol{r}_W$ yields a value $t_{0,W} = c_1 s + c_2 s^n + \delta_{yzw}$ such that c_1, c_2 and δ_{yzw} are linearly independent values that only depend on challenge values. Then $g^{t_{0,W}}$ contains $g^{c_1 s} = (g^s)^{c_2} = V_{\text{unblinded}}^{c_1}$ and equivalently $g^{c_2 s^n} = \left(g^{(s^n)}\right)^{c_2} = W_{\text{unblinded}}^{c_2}$ which links V and W to the linear constraints as desired.

Remark: When PR is used as sub-protocol of MKS, V does not need to be blinded because the value s which it commits to is already chosen uniformly at random, and

therefore $V_{\text{unblinded}} = V$. s^n is also distributed uniformly which enables us to also drop W's blinding value, hence $W_{\text{unblinded}} = W$.

We achieve representation of the above linear constraints by defining the left side and the right side of the Hadamard product as follows:

$$\begin{aligned}\boldsymbol{l}_W &= \boldsymbol{a}_{L,W} + z\boldsymbol{y}^{-n} + \boldsymbol{w}^n - \boldsymbol{w}_0^{n-1}y^{-1} \\ \boldsymbol{r}_W &= \boldsymbol{y}^n \circ \boldsymbol{a}_{R,W} - \boldsymbol{y}_z^{n-1}\end{aligned}$$

This yields

$$\begin{aligned}t_{0,W} &= \langle \boldsymbol{l}_W, \boldsymbol{r}_W \rangle \\ &= s\left((wy)^{n-1} + snz\right) + s^n y^{n-1} + \delta_{yzw} \\ &\quad \text{where } \delta_{yzw} = -z(2 + z + (n-1)y^{-1}) + (y^{-1} - w)\sum_{i=0}^{n-2}(wy)^i\end{aligned}$$

if and only if the linear constraints on $\boldsymbol{a}_{L,W}$ and $\boldsymbol{a}_{R,W}$ are fulfilled.

To avoid information leakage by directly sending $\boldsymbol{l}_W$ and $\boldsymbol{r}_W$ to the verifier, the prover samples random blinding vectors $\boldsymbol{s}_{L,W}$ and $\boldsymbol{s}_{R,W}$ right after setting up $\boldsymbol{a}_{L,W}$ resp. $\boldsymbol{a}_{R,W}$ and commits to them with $S_W = \boldsymbol{g}^{\boldsymbol{s}_{L,W}}\boldsymbol{h}^{\boldsymbol{s}_{R,W}}h^{\beta_W}$ using random blinding value β_W. Those blinding vectors need to be multiplied with a challenge from the verifier before they are added to $\boldsymbol{a}_{L,W}$ resp. $\boldsymbol{r}_{R,W}$.

This results in a polynomial identity test with

$$\begin{aligned}\boldsymbol{l}_W(X) &= \boldsymbol{a}_{L,W} + \boldsymbol{s}_{L,W}X + z\boldsymbol{y}^{-n} + \boldsymbol{w}^n - \boldsymbol{w}_0^{n-1}y^{-1} \\ \boldsymbol{r}_W(X) &= \boldsymbol{y}^n \circ (\boldsymbol{a}_{R,W} + \boldsymbol{s}_{R,W}X) - \boldsymbol{y}_z^{n-1} \\ t_W(X) &= \langle \boldsymbol{l}_W(X), \boldsymbol{r}_W(X) \rangle \\ &= t_{0,W} + t_{1,W}X + t_{2,W}X^2\end{aligned}$$

where the prover commits to $t_{1,W}$ and $t_{2,W}$ with blinding values $\tau_{1,W}$ resp. $\tau_{2,W}$ and then receives a random challenge point $X = x \leftarrow\!\!\$\ \mathbb{Z}_q^*$ from the verifier.

The prover computes the inner product by evaluating $t_W(x)$ and then combines blinding values from his previous commitments:

$$\begin{aligned} \hat{t_W} &= t_W(x) \\ \tau_{x,W} &= \sigma_V((wy)^{n-1} + sz) + \sigma_W y^{n-1} + \tau_{1,W}x + \tau_{2,W}x^2 \\ \mu_W &= \alpha + \beta x \end{aligned}$$

Finally, the verifier performs checks on the values and commitments received from the prover that will convince her that V and W indeed satisfy the n-th power relation. As the protocol is based on an inner product relation, the special inner product argument from [5] can be utilized. The checks involving $\boldsymbol{l}_W$ and $\boldsymbol{r}_W$ are then performed indirectly such that the power relation argument size is reduced to $\mathcal{O}(\log n)$. Find those checks in steps 13, 16 and 19 of the complete interactive protocol 1.

4.4. Complete Interactive MKS Protocol

In the complete protocol we mark our changes and corrections applied to the protocol given in [1], section 4.3, "Bulletproofs sub-routine", in light gray and the integrated power relation protocol in dark gray. We list computations that are only required if the special inner product protocol is not executed with a preceding "*". The protocol includes σ_V and σ_W as in the standalone PR argument. To indicate how it should be integrated into the MKS argument we mark computations where those blinding values appear and should be ignored, i.e. be set to zero, with $[\cdot]^\dagger$.

Algorithm 1 Complete Multi-Key Shuffle Argument

Statement: $\forall i \in \{1, \ldots, n\} : (h^s_{\pi(i)}, b^s_{\pi(i)}, c^s_{\pi(i)}, d^s_{\pi(i)}) = (h'_i, b'_i, c'_i, d'_i)$
CRS: $g \in \mathbb{G}, \boldsymbol{g} \in \mathbb{G}^n, g' \in \mathbb{G}, \boldsymbol{h}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d} \in \mathbb{G}^n$
Public input: $\boldsymbol{h}', \boldsymbol{b}', \boldsymbol{c}', \boldsymbol{d}' \in \mathbb{G}^n, V, W \in \mathbb{G}, n \in \mathbb{Z}$
Private input: $s \in \mathbb{Z}^*_q, \pi \in \Sigma_n, [\sigma_V, \sigma_W \in \mathbb{Z}_q]^\dagger$, with $V = g^s\ [\cdot h^{\sigma_V}]^\dagger$ and $W = g^{s^n}\ [\cdot h^{\sigma_W}]^\dagger$
$\boldsymbol{h}$: corresponding receiver's public keys

$\boldsymbol{b}$: ElGamal ciphertext – randomizer
$\boldsymbol{c}$: ElGamal ciphertext – message
$\boldsymbol{d}$: corresponding sender's public keys
$\boldsymbol{V}$: commitment to s
$\boldsymbol{W}$: commitment to s^n

1. $\mathcal{P}$ **:** If CRS does not pass CRSCheck, return $\perp$.
   Commit to shifted shuffle with the help of random challenges:
2. $\mathcal{V} \rightarrow \mathcal{P}$ **:** $r, u \leftarrow\$ \mathbb{Z}_q^*$
3. $\mathcal{P}$ **and** $\mathcal{V}$ **:**
   - $k_i = (r - u^i)\ i \in 1, \ldots, n$
   - $\hat{k} = \prod_{i=1}^{n} k_i$
   - $\bar{k} = \sum_{i=1}^{n} k_i$
   - $A_{R,h} = \prod_{i=1}^{n} h_i'^{k_i}$
   - $A_{R,b} = \prod_{i=1}^{n} b_i'^{k_i}$
   - $A_{R,c} = \prod_{i=1}^{n} c_i'^{k_i}$
   - $A_{R,d} = \prod_{i=1}^{n} d_i'^{k_i}$

   Prepare constraints to express ...
4. $\mathcal{P}$ **:**

   ...shifted shuffle as an inner product
   - $\boldsymbol{a}_R = (sk_{\pi^{-1}(1)}, \ldots, sk_{\pi^{-1}(n)})$
   - $\boldsymbol{a}_L = (1, sk_{\pi^{-1}(1)}, s^2 k_{\pi^{-1}(1)} k_{\pi^{-1}(2)}, \ldots, s^{n-1} \prod_{i=1}^{n-1} k_{\pi^{-1}(i)})$
   - $\alpha, \beta, \rho_h, \rho_b, \rho_c, \rho_d \leftarrow\$ \mathbb{Z}_q$
   - $\boldsymbol{s}_L, \boldsymbol{s}_R \leftarrow\$ \mathbb{Z}_q^n$

   ...power relation as an inner product.
   - $\boldsymbol{a}_{R,W} = (s, \ldots, s)$ with length n
   - $\boldsymbol{a}_{L,W} = (1, s, s^2, s^{n-1})$
   - $\alpha_W, \beta_W \leftarrow\$ \mathbb{Z}_q$
   - $\boldsymbol{s}_{L,W}, \boldsymbol{s}_{R,W} \leftarrow\$ \mathbb{Z}_q^n$

   Commit to blinded constraint vectors:
5. $\mathcal{P} \rightarrow \mathcal{V}$ **:**
   - $A_L = \boldsymbol{g}^{\boldsymbol{a}_L} h^{\alpha}$
   - $S_L = \boldsymbol{g}^{\boldsymbol{s}_L} h^{\beta}$
   - $S_{R,h} = \boldsymbol{h}^{\boldsymbol{s}_R} h^{\rho_h}$
   - $S_{R,b} = \boldsymbol{b}^{\boldsymbol{s}_R} h^{\rho_b}$
   - $S_{R,c} = \boldsymbol{c}^{\boldsymbol{s}_R} h^{\rho_c}$
   - $S_{R,d} = \boldsymbol{d}^{\boldsymbol{s}_R} h^{\rho_c}$
   - $A_W = \boldsymbol{g}^{\boldsymbol{a}_{L,W}} \boldsymbol{h}^{\boldsymbol{a}_{R,W}} h^{\alpha_W}$
   - $S_W = \boldsymbol{g}^{\boldsymbol{s}_{L,W}} \boldsymbol{h}^{\boldsymbol{s}_{R,W}} h^{\beta_W}$

   Prepare combination of constraints for polynomial identity test which will enforce those constraints:

6. $\boldsymbol{\mathcal{V}} \rightarrow \boldsymbol{\mathcal{P}}$ **:** $y, z, w \leftarrow\!\$\, \mathbb{Z}_q^*$
7. $\boldsymbol{\mathcal{P}}$ **and** $\boldsymbol{\mathcal{V}}$ **:**
   - $\boldsymbol{y}^n = (1, y, y^2, \ldots, y^{n-1})$
   - $\boldsymbol{y}_z^{n-1} = (z, 1, y, \ldots, y^{n-2})$
   - $\delta_{yz} = -z^3(z + (n-1)y^{-1}) - z$
   - $\boldsymbol{w}^n = (1, w, w^2, \ldots, w^{n-1})$
   - $\boldsymbol{w}_0^{n-1} = (0, 1, w, \ldots, w^{n-2})$
   - $\delta_{yzw} = -z(2 + z + (n-1)y^{-1}) + (y^{-1} - w)\sum_{i=0}^{n-2}(wy)^i$
8. $\boldsymbol{\mathcal{P}}$ **:**

   MKS: Calculate coefficients of polynomial:
   - $\boldsymbol{l}(X) = \boldsymbol{a}_L + z^3\boldsymbol{y}^{-n} + \boldsymbol{s}_L X^2$
   - $\boldsymbol{r}(X) = (\boldsymbol{a}_R X + \boldsymbol{s}_R X^3) \circ \boldsymbol{y}^n - X\, \boldsymbol{y}_z^{n-1}$
   - $t(X) = \langle \boldsymbol{l}(X), \boldsymbol{r}(X) \rangle = \sum_{i \in \{1,3,5\}} t_i X^i$
   - $t_1 = y^{n-1}s^n\hat{k} + z^3 s\bar{k} + \delta_{yz}$

   MKS: Commit to coefficients:
   - $\tau_i \leftarrow\!\$\, \mathbb{Z}_q, i = \{3, 5\}$
   - $T_i = g^{t_i} h^{\tau_i}, i = \{3, 5\}$

   PR: Calculate coefficients of polynomial:
   - $\boldsymbol{l}_W(X) = \boldsymbol{a}_{L,W} + \boldsymbol{s}_{L,W} X + z\boldsymbol{y}^{-n} + \boldsymbol{w}^n - \boldsymbol{w}_0^{n-1} y^{-1}$
   - $\boldsymbol{r}_W(X) = \boldsymbol{y}^n \circ (\boldsymbol{a}_{R,W} + \boldsymbol{s}_{R,W} X) - y_z^{n-1}$
   - $t_W(X) = \langle \boldsymbol{l}_W(X), \boldsymbol{r}_W(X) \rangle = \sum_{i \in \{0,1,2\}} t_{i,W} X^i$
   - $t_{0,W} = s^n y^{n-1} + s((wy)^{n-1} + nz) + \delta_{yzw}$

   PR: Commit to coefficients:
   - $\tau_{i,W} \leftarrow\!\$\, \mathbb{Z}_q, i = \{1, 2\}$
   - $T_{i,W} = g^{t_{i,W}} h^{\tau_{i,W}}, i = \{1, 2\}$
9. $\boldsymbol{\mathcal{P}} \rightarrow \boldsymbol{\mathcal{V}}$ **:** $T_3, T_5,\ T_{1,W}, T_{2,W}$

   Evaluate MKS and PR polynomials at random challenge point $X = x$ and combine secret random blinding values:
10. $\boldsymbol{\mathcal{V}} \rightarrow \boldsymbol{\mathcal{P}}$ **:** $x \leftarrow\!\$\, \mathbb{Z}_q^*$
11. $\boldsymbol{\mathcal{P}} \rightarrow \boldsymbol{\mathcal{V}}$ **:**
    * $\boldsymbol{l} = \boldsymbol{l}(x)$
    * $\boldsymbol{r} = \boldsymbol{r}(x)$
    - $\hat{t} = \langle \boldsymbol{l}, \boldsymbol{r} \rangle$
    - $\tau_x = [(\sigma_V z^3 \bar{k} + \sigma_W n y^{n-1} \hat{k})x]^\dagger + \tau_3 x^3 + \tau_5 x^5$
    - $\mu_h = \alpha + \beta x^2 + \rho_h x^3$
    - $\mu_b = \alpha + \beta x^2 + \rho_b x^3$
    - $\mu_c = \alpha + \beta x^2 + \rho_c x^3$

- $\mu_d = \alpha + \beta x^2 + \rho_d x^3$
  * $\boldsymbol{l}_W = \boldsymbol{l}_W(x)$
  * $\boldsymbol{r}_W = \boldsymbol{r}_W(x)$
- $\hat{t_W} = \langle \boldsymbol{l}_W, \boldsymbol{r}_W \rangle$
- $\tau_{x,W} = [\sigma_V((wy)^{n-1} + sz) + \sigma_W y^{n-1}]^\dagger + \tau_{1,W} x + \tau_{2,W} x^2$
- $\mu_W = \alpha_W + \beta_W x$

12. $\mathcal{P}$ **and** $\mathcal{V}$ **:**
    - $\boldsymbol{h}^* = \boldsymbol{h}^{\boldsymbol{y}^{-n}}$
    - $\boldsymbol{b}^* = \boldsymbol{b}^{\boldsymbol{y}^{-n}}$
    - $\boldsymbol{c}^* = \boldsymbol{c}^{\boldsymbol{y}^{-n}}$
    - $\boldsymbol{d}^* = \boldsymbol{d}^{\boldsymbol{y}^{-n}}$

    Perform polynomial identity test with inner product protocol (proof size in $\mathcal{O}(n)$, defer steps marked with "*") or without inner product protocol (proof size in $\mathcal{O}(log_2 n)$, run steps marked with "*" immediately and stop after 13)
13. $\mathcal{V}$ **:**
    - $g^{\hat{t}} \cdot h^{\tau_x} \stackrel{?}{=} V^{z^3 \bar{k} x} \cdot W^{y^{n-1} \hat{k} x} \cdot g^{\delta_{yz}} \cdot T_3^{x^3} \cdot T_5^{x^5}$
    * $h^{\mu_h} \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot (\boldsymbol{h}^*)^{\boldsymbol{r}} \stackrel{?}{=} A_L \cdot A_{R,h}^x \cdot S_L^{x^2} \cdot S_{R,h}^{x^3} \cdot \boldsymbol{g}^{z^3 \boldsymbol{y}^{-n}} \cdot (\boldsymbol{h}^*)^{-x \boldsymbol{y}_z^{n-1}}$
    * $h^{\mu_b} \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot (\boldsymbol{b}^*)^{\boldsymbol{r}} \stackrel{?}{=} A_L \cdot A_{R,b}^x \cdot S_L^{x^2} \cdot S_{R,b}^{x^3} \cdot \boldsymbol{g}^{z^3 \boldsymbol{y}^{-n}} \cdot (\boldsymbol{b}^*)^{-x \boldsymbol{y}_z^{n-1}}$
    * $h^{\mu_c} \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot (\boldsymbol{c}^*)^{\boldsymbol{r}} \stackrel{?}{=} A_L \cdot A_{R,c}^x \cdot S_L^{x^2} \cdot S_{R,c}^{x^3} \cdot \boldsymbol{g}^{z^3 \boldsymbol{y}^{-n}} \cdot (\boldsymbol{c}^*)^{-x \boldsymbol{y}_z^{n-1}}$
    * $h^{\mu_d} \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot (\boldsymbol{d}^*)^{\boldsymbol{r}} \stackrel{?}{=} A_L \cdot A_{R,d}^x \cdot S_L^{x^2} \cdot S_{R,d}^{x^3} \cdot \boldsymbol{g}^{z^3 \boldsymbol{y}^{-n}} \cdot (\boldsymbol{d}^*)^{-x \boldsymbol{y}_z^{n-1}}$
    * $\hat{t} \stackrel{?}{=} \langle \boldsymbol{l}, \boldsymbol{r} \rangle$
    - $g^{t\hat{W}} \cdot h^{\tau_{x,W}} \stackrel{?}{=} V^{(wy)^{n-1} + nz} \cdot W^{y^{n-1}} \cdot g^{\delta_{yzw}} \cdot T_{1,W}^x \cdot T_{2,W}^{x^2}$
    * $h^{\mu_W} \cdot \boldsymbol{g}^{\boldsymbol{l}_W} \cdot (\boldsymbol{h}^*)^{\boldsymbol{r}_W} \stackrel{?}{=} A_W \cdot S_W^x \cdot \boldsymbol{g}^{z \boldsymbol{y}^{-n} + \boldsymbol{w}^n - \boldsymbol{w}_0^{n-1}} \cdot (\boldsymbol{h}^*)^{-\boldsymbol{y}_z^{n-1}}$
    * $\hat{t_W} \stackrel{?}{=} \langle \boldsymbol{l}_W, \boldsymbol{r}_W \rangle$

*13. $\mathcal{V}$ **:** Accept or Reject.

Initialize inner product argument - pad vectors to lengths of a power of 2:

14. $\mathcal{P}$ **and** $\mathcal{V}$ **:**
    - $\boldsymbol{k} = \lceil \log_2 \boldsymbol{n} \rceil$ (number of halving rounds)
    - $\boldsymbol{n_2} = \boldsymbol{2^k}$ (the smallest power of 2 greater or equal to $\boldsymbol{n}$)
    - $\boldsymbol{G} = \boldsymbol{g} \| (\boldsymbol{1}_{\mathbb{G}}, \ldots, \boldsymbol{1}_{\mathbb{G}})$, s.t. $\boldsymbol{G}$ has length $\boldsymbol{n_2}$
    - $\boldsymbol{H} = \boldsymbol{h}^* \| (\boldsymbol{1}_{\mathbb{G}}, \ldots, \boldsymbol{1}_{\mathbb{G}})$, s.t. $\boldsymbol{H}$ has length $\boldsymbol{n_2}$
    - $\boldsymbol{B} = \boldsymbol{b}^* \| (\boldsymbol{1}_{\mathbb{G}}, \ldots, \boldsymbol{1}_{\mathbb{G}})$, s.t. $\boldsymbol{B}$ has length $\boldsymbol{n_2}$
    - $\boldsymbol{C} = \boldsymbol{c}^* \| (\boldsymbol{1}_{\mathbb{G}}, \ldots, \boldsymbol{1}_{\mathbb{G}})$, s.t. $\boldsymbol{C}$ has length $\boldsymbol{n_2}$
    - $\boldsymbol{D} = \boldsymbol{d}^* \| (\boldsymbol{1}_{\mathbb{G}}, \ldots, \boldsymbol{1}_{\mathbb{G}})$, s.t. $\boldsymbol{D}$ has length $\boldsymbol{n_2}$
15. $\mathcal{P}$ **:**
    - $\boldsymbol{a}_{IP} = \boldsymbol{l} \| (0, \ldots, 0)$, s.t. $\boldsymbol{a}_{IP}$ has length n_2
    - $\boldsymbol{b}_{IP} = \boldsymbol{r} \| (0, \ldots, 0)$, s.t. $\boldsymbol{b}_{IP}$ has length n_2

- $c = \langle \boldsymbol{a}_{IP}, \boldsymbol{b}_{IP} \rangle$
- $\boldsymbol{a}_W = \boldsymbol{l}_W \| (0, \ldots, 0)$, s.t. $\boldsymbol{a}_W$ has length n_2
- $\boldsymbol{b}_W = \boldsymbol{r}_W \| (0, \ldots, 0)$, s.t. $\boldsymbol{b}_W$ has length n_2
- $c_W = \langle \boldsymbol{a}_W, \boldsymbol{b}_W \rangle$

Prepare conditions for halving:

- $P_h = \boldsymbol{G}^{\boldsymbol{a}_{IP}} \cdot \boldsymbol{H}^{\boldsymbol{b}_{IP}} \cdot g'^{c}$
- $P_b = \boldsymbol{G}^{\boldsymbol{a}_{IP}} \cdot \boldsymbol{B}^{\boldsymbol{b}_{IP}} \cdot g'^{c}$
- $P_c = \boldsymbol{G}^{\boldsymbol{a}_{IP}} \cdot \boldsymbol{C}^{\boldsymbol{b}_{IP}} \cdot g'^{c}$
- $P_d = \boldsymbol{G}^{\boldsymbol{a}_{IP}} \cdot \boldsymbol{D}^{\boldsymbol{b}_{IP}} \cdot g'^{c}$
- $P_W = \boldsymbol{G}^{\boldsymbol{a}_W} \cdot \boldsymbol{H}^{\boldsymbol{b}_W} \cdot g'^{c_W}$

Prepare conditions for halving with more complicated computations for the verifier:

16. $\boldsymbol{\mathcal{V}}$ **:**
    - $P_h = A_L \cdot A_{R,h}^{x} \cdot S_L^{x^2} \cdot S_{R,h}^{x^3} \cdot \boldsymbol{g}^{z^3 \boldsymbol{y}^{-n}} \cdot (\boldsymbol{h}^*)^{-x\boldsymbol{y}_z^{n-1}} \cdot h^{-\mu_h} \cdot g'^{\hat{t}}$
    - $P_b = A_L \cdot A_{R,b}^{x} \cdot S_L^{x^2} \cdot S_{R,b}^{x^3} \cdot \boldsymbol{g}^{z^3 \boldsymbol{y}^{-n}} \cdot (\boldsymbol{b}^*)^{-x\boldsymbol{y}_z^{n-1}} \cdot h^{-\mu_b} \cdot g'^{\hat{t}}$
    - $P_c = A_L \cdot A_{R,c}^{x} \cdot S_L^{x^2} \cdot S_{R,c}^{x^3} \cdot \boldsymbol{g}^{z^3 \boldsymbol{y}^{-n}} \cdot (\boldsymbol{c}^*)^{-x\boldsymbol{y}_z^{n-1}} \cdot h^{-\mu_c} \cdot g'^{\hat{t}}$
    - $P_d = A_L \cdot A_{R,d}^{x} \cdot S_L^{x^2} \cdot S_{R,d}^{x^3} \cdot \boldsymbol{g}^{z^3 \boldsymbol{y}^{-n}} \cdot (\boldsymbol{d}^*)^{-x\boldsymbol{y}_z^{n-1}} \cdot h^{-\mu_d} \cdot g'^{\hat{t}}$
    - $P_W = A_W \cdot S_W^{x} \cdot \boldsymbol{g}^{z\boldsymbol{y}^{-n} + \boldsymbol{w}^n - \boldsymbol{w}_0^{n-1}} \cdot (\boldsymbol{h}^*)^{-\boldsymbol{y}_z^{n-1}} \cdot h^{-\mu_W} \cdot g'^{\hat{t}_W}$
17. $\boldsymbol{\mathcal{P}} \leftrightarrow \boldsymbol{\mathcal{V}}$ **:** Engage in inner product argument with the following pairings using only a single challenge value in each of the k halving rounds:
    - $(\boldsymbol{G}, \boldsymbol{a}_{IP}), (\boldsymbol{H}, \boldsymbol{b}_{IP})$, commit halves to L_{h_i} and R_{h_i}, aggregate on $P_{i,h}$
    - $(\boldsymbol{G}, \boldsymbol{a}_{IP}), (\boldsymbol{B}, \boldsymbol{b}_{IP})$, commit halves to L_{b_i} and R_{b_i}, aggregate on $P_{i,b}$
    - $(\boldsymbol{G}, \boldsymbol{a}_{IP}), (\boldsymbol{C}, \boldsymbol{b}_{IP})$, commit halves to L_{c_i} and R_{c_i}, aggregate on $P_{i,c}$
    - $(\boldsymbol{G}, \boldsymbol{a}_{IP}), (\boldsymbol{D}, \boldsymbol{b}_{IP})$, commit halves to L_{d_i} and R_{d_i}, aggregate on $P_{i,d}$
    - $(\boldsymbol{G}, \boldsymbol{a}_W), (\boldsymbol{H}, \boldsymbol{b}_W)$, commit halves to L_{W_i} and R_{W_i}, aggregate on $P_{i,W}$

Final checks in the last single-valued round:

18. $\boldsymbol{\mathcal{P}} \rightarrow \boldsymbol{\mathcal{V}}$ **:** $a_0, b_0, a_{0,W}, b_{0,W}$
19. $\boldsymbol{\mathcal{V}}$ **:**
    - $c_0 = a_0 b_0$
    - $P_{0,h} \stackrel{?}{=} G_0^{a_0} \cdot H_0^{b_0} \cdot g'^{c_0}$
    - $P_{0,b} \stackrel{?}{=} G_0^{a_0} \cdot B_0^{b_0} \cdot g'^{c_0}$
    - $P_{0,c} \stackrel{?}{=} G_0^{a_0} \cdot C_0^{b_0} \cdot g'^{c_0}$
    - $P_{0,d} \stackrel{?}{=} G_0^{a_0} \cdot D_0^{b_0} \cdot g'^{c_0}$
    - $c_{0,W} = a_{0,W} b_{0,W}$
    - $P_{0,W} \stackrel{?}{=} G_0^{a_{0,W}} \cdot H_0^{b_{0,W}} \cdot g'^{c_{0,W}}$
20. $\boldsymbol{\mathcal{V}}$ **:** Accept or Reject.

Theorem 2. Under the discrete log hardness assumption, the power argument has perfect completeness, perfect special honest-verifier zero-knowledge, and computational witness-extended simulation.
We prove this theorem in chapter 5.

4.5. Complexity

Complexity of MKS follows directly from the protocol 1.

Proof Size. Space complexity of communication between $\mathcal{P}$ and $\mathcal{V}$ consists of two parts: First, the constraint part (until step 11, ignoring "*"), and second, the inner product part (everything else).

Proof Size – Constraints. The complexity of the constraint part is independent of how correctness of the inner product is checked: The prover sends group elements $V, W, A_W, V_W, A_L, S_L, S_{R,h}, S_{R,b}, S_{R,c}, S_{R,d}, T_3, T_5, T_{1,W}, T_{2,W}$, scalars $\tau_x, \mu_h, \mu_b, \mu_c, \mu_d, t, \tau_{x,W}, \mu_W, t_W$, the verifier sends random challenges (scalars) r, u, y, z, w, x, resulting in a constant 14 group elements and 15 scalars.

Proof Size – Inner Product (Linear). If the specialized inner product protocol is not executed (option "*"), $\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{l}_W, \boldsymbol{r}_W$ need to be communicated to the verifier. Each of them has length n, resulting in a linear proof size of $4 \cdot n$ scalars for the second part. For the whole protocol, 14 group elements and $4n + 15$ scalars are needed.

Proof Size – Inner Product (Logarithmic). If the specialized inner product protocol is used, the vectors of the inner products do not need to be communicated. Instead, in each halving iteration of the inner product protocol, commitments $L_{\{h,b,c,d,W\}_i}$ and $R_{\{h,b,c,d,W\}_i}$ are sent to the verifier to be aggregated on $P_{h,b,c,d,W}$ and one challenge is sent back to the prover. Finally, scalars $a_0, b_0, a_{0,W}, b_{0,W}$ are sent to the verifier for the final checks. This results in a proof size of $5 \cdot 2 \cdot \lceil log_2(n) \rceil$ group elements and $\lceil log_2(n) \rceil + 4$

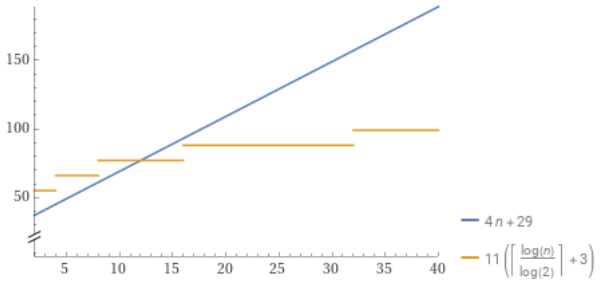

Figure 2.: Plot showing break even point for number of communicated elements at $n = 12$ of log-sized inner product argument with $11\lceil log_2(n)\rceil + 33$ versus the linearly sized with $4n + 29$

scalars for the second part. Altogether, these are $10\lceil log_2(n)\rceil + 14$ group elements and $\lceil log_2(n)\rceil + 19$ scalars.

In elliptic curve cryptography, we can assume the size of a group element to be the same as the size of a scalar to calculate the point at which the specialized inner product argument performs better than the simple linear version: The number of elements exchanged is exactly equal in the case of $n = 12$, meaning the specialized inner product argument gives a smaller proof size for all shuffle sizes n larger than 12.

Prover Time. The multi-key shuffle protocol involves a constant number of operations linear in the size of the shuffle n. In the inner product protocol, $\lceil log_2(n)\lceil$ steps with $c \cdot n, c \cdot \frac{n}{2}, c \cdot \frac{n}{4}, \cdots, 1$ operations for constant c, summing to a total of at most $c \cdot n \cdot 2$ which is also linear in n, operations are executed. Altogether this results in a prover time in $\mathcal{O}(n)$

Verifier Time. Analog to prover time, verifier time is linear in the size of the shuffle n.

Implementation. The goal of our implementation is not about performance but to demonstrate the zero knowledge protocol for the multi-key verifiable shuffle and to show how the mathematics behind it works in detail. This is achieved by the Mathematica notebook implementation and explicitly shown in the demonstration videos containing

the execution of the notebooks for different shuffle sizes. There are many sanity checks included into the implementation that are not needed in a production application.

Tests. Tests of the implementation were performed on the following hardware:

CPU	Intel Xeon E3-1231 v3 at 3.4 GHz (4 cores, 8 threads, 22nm, Haswell)
RAM	32 GiB DDR3 RAM (800 MHz)
Persistent storage	0.5 TB SATA SSD
Mainboard	Gigabyte H97M-D3H
Graphics card	NVIDIA GeForce RTX 2080 Ti
Operating System	Windows 10 Pro on UEFI

Videos of test runs are attached to this thesis. Tests have been run for shuffle sizes from 2 up to 9. The system heavily slows down when arrays of size 16 get involved. This problem has not been investigated because it is seen only as a minor limitation for a notebook based proof-of-concept implementation in Mathematica which is not known to be optimized for cryptography. Runtimes were measured in interactive mode with log-sized inner product protocol enabled, beginning with the start of the verifier component and ending with successful verification and inspection:

shuffle size n	2	3	4	5	6	7	8	9
execution time [s]	21	27	30	36	38	41	46	183
group elements	24	34	34	44	44	44	44	54
scalars	20	21	21	22	22	22	22	23
group elements (linear)	14	14	14	14	14	14	14	14
scalars (linear)	23	27	31	35	39	43	47	51

Runtimes include calculations needed for sanity checks. The implementation has not been optimized for efficiency but for being easier to read, debug and extend.

4.6. Secure Setup of Generators for Commitments

The algorithm $\mathsf{Setup}(1^\lambda)$ outputs public parameters $\mathbb{G}, q, g, h, \boldsymbol{g}, g'$ where $\mathbb{G}$ is a field of prime order q, g usually is a commonly known generator, and $h, \boldsymbol{g}, g'$ are chosen as *NUMS* ("nothing up my sleeve") generators such that none of the discrete log relations between g, h, g' and any element of $\boldsymbol{g}$ is known. These *NUMS* generators are created by selecting group elements with a cryptographic pseudorandom number generator that is initialized with a fixed seed value. Everyone can confirm the generators were created honestly (i.e. such that no discrete logarithm relation between them is known) by executing the pseudorandom setup algorithm of the *NUMS* generators by himself.

4.7. Fiat-Shamir Transform

Every message exchanged between prover and verifier has to be included in the Fiat-Shamir transform. To avoid replay attacks, the identity (e.g. public key) of the prover is added to the Fiat-Shamir transform. To avoid tampering with setup configuration, the configuration details from the setup of $\mathbb{G}$, communication setup, interactivity properties and the size of the shuffle n are added to the Fiat-Shamir transform.

As the *NUMS* group elements h, g' and $\boldsymbol{g}$ are pseudo-randomly generated by the Setup algorithm, the pseudo-random number generator specification and its seed are also included in the Fiat-Shamir transform to not only explicitly bind the prover to those *NUMS* generators but moreover ensure verifiability of their *NUMS* characteristic.

Calling the list of these variables $\mathsf{initvars}$, the Fiat-Shamir transform of the interactive shuffle protocol generates challenge values as follows:

- $\mathsf{initvars}, V, W, A_W, S_W \rightarrow r$
- $r \rightarrow u$
- $u, A_L, S_L, S_{R,h}, S_{R,b}, S_{R,c}, S_{R,d} \rightarrow y$

- $y \rightarrow z$
- $z \rightarrow w$
- $w, T_3, T_5, T_{1,W}, T_{2,W} \rightarrow x$
- $x, \tau_x, \mu_h, \mu_b, \mu_c, \mu_d, \hat{t}, \tau_{x,W}, \mu_W, \hat{t_W}, L_{k,\{h,b,c,d,W\}}, R_{k,\{h,b,c,d,W\}} \rightarrow u_k$
- for $i \in \{k-1, \cdots, 1\} : u_{i+1}, L_{i,\{h,b,c,d,W\}}, R_{i,\{h,b,c,d,W\}} \rightarrow u_i$
- $u_1, a_0, b_0, a_{0,W}, b_{0,W}$: No further challenge generated.

Security of the protocol would be compromised if one of the variables on the left side would be omitted from the Fiat-Shamir transform. It would become vulnerable to a *Frozen Heart Attack*. In the worst case this attack enables the prover to forge arbitrary proofs. We look into this kind of attack in more detail in section 6.3.

The Fiat-Shamir transform is implemented using the SHA256 hash function which is mapped to multiple parts of the input values in case of group size $q >= 2^{256}$. The Fiat-Shamir transform is implemented in a small library that ensures each challenge (right side) gets at least as much entropy from the input (left side) as its bit size by keeping track of the variables that are added to the transform. This shall help prevent loss of the prover's privacy by otherwise unrecognized bugs in the implementation.

4.8. Yet no Masked Single-Key Shuffles from MKS

Hoffmann et al. achieved efficient arguments of logarithmic size for shuffles of ElGamal ciphertexts in 2019 [6] by optimizing Bayer and Groth's [11] shuffle proof of size $\sqrt{n}$ with their own sub protocols (see section 1.2.4 and appendix C in [6]).

With Bulletproofs [5] shuffles of logarithmic size on commitments can be implemented by sorting circuits, but these do not work for ElGamal ciphertexts. While we have no proof that Bulletproofs can not be modified such that they can take ElGamal ciphertexts as inputs, we can neither find such a modification in scientific literature nor have we found an obvious modification on our own.

Remark: Hoffmann et al. state that they do not see how Bulletproofs could be transformed to give a log-sized shuffle on ElGamal ciphertexts in [6], appendix C, paragraph 4.

The extensions to the original Bulletproofs protocols presented in [1] and the specific use case of a multi-key shuffle raise the question if logarithmic sized shuffles of ElGamal ciphertexts are enabled with this. It is true that the proof of shuffle on commitments can be optimized using the proposed arithmetic circuits for randomized arguments (argument size still remains $\mathcal{O}(\log n)$), see [1], section 4.3, "Comparison with Bulletproofs". Unfortunately, we nevertheless failed to transform the multi-key shuffle into an efficient masked single-key shuffle by naive approaches – the terms "mask" and "shift" can be used interchangeably:

A masked single-key shuffle requires one mask value for each ciphertext to be shuffled instead of the single shift value s that is sufficient for the multi-key shuffle. The first problem is that it is not obvious how a masked single-key shuffle could be realized without the shuffler communicating commitments to all secret masking values to the verifier which makes the argument linear in size. Secondly, as the masking values need to be sampled independently of each other, the single key shift s in MKS would need to be replaced by a vector of masking values. Unfortunately, steps 4, 7 and 8 of the MKS protocol can not be naively transformed in a way such that they incorporate a vector of masking values instead of a single key shift.

We must conclude that we are not able to implement a masked single-key shuffle for ElGamal ciphertexts using the Bulletproofs paradigm and the multi-key shuffle.

5. Security Proofs for the Power Relation Argument

Bünz et al. have already proved security of the multi-key shuffle argument in appendices B and C of [1] assuming existence of an equally secure power relation argument. We defined security properties that such a power relation argument needs to fulfill in section 3.3 and proposed a protocol based on established techniques in zero-knowledge cryptography. Formal security proofs for this protocol complete the mathematical part of this thesis.

Security of PR and $\mathsf{PR}_{\mathsf{MKS}}$ requires

- privacy (Theorem 1, section 5.1)
- perfect completeness (section 5.4)
- subversion perfect special honest-verifier zero-knowledge (section 5.5)
- computational witness-extended emulation (section 5.6)

as stated in Theorem 2.

Proofs for similar Theorems 1 and 2 presented in [1] are correct for the complete MKS protocol 1.

5.1. Theorem 1: Privacy of PR – Indistinguishability under Chosen Plaintext Attack

Working with the definition of privacy (3.6) by the security game, we see that Pedersen commitments V_0, W_0 and V_1, W_1 are perfectly hiding (3.3). This means they can not be distinguished from each other, so they truly are private.

More formally, given g and h are generators for $\mathbb{G}$, looking at the sets of possible values for those commitments $\mathbb{V}_0, \mathbb{W}_0, \mathbb{V}_1, \mathbb{W}_1$, using notation for sets similar to vectors, note that

$$\begin{aligned}
\mathbb{V}_0 &= g^{s_0} h^{\mathbb{Z}_q^{[\sigma_{V_0}]}} = g^{s_0} \cdot \mathbb{G} = \mathbb{G} \\
&= g^{s_1} \cdot \mathbb{G} = g^{s_1} h^{\mathbb{Z}_q^{[\sigma_{V_1}]}} \\
&= \mathbb{V}_1 \wedge \\
\mathbb{W}_0 &= g^{s_0^n} h^{\mathbb{Z}_q^{[\sigma_{W_0}]}} = g^{s_0^n} \cdot \mathbb{G} = \mathbb{G} \\
&= g^{s_1^n} \cdot \mathbb{G} = g^{s_1^n} h^{\mathbb{Z}_q^{[\sigma_{W_1}]}} \\
&= \mathbb{W}_1
\end{aligned}$$

Distribution over $\mathbb{Z}_q$ for all of $\sigma_{\{V,W\}_{\{0,1\}}}$ is uniform, so distribution over $\mathbb{G}$ is also uniform for all of $\{V, W\}_{\{0,1\}}$ which makes them indistinguishable. Any adversary is left to guess index b by chance, meaning advantage $\mathsf{Adv}_{\mathcal{A}}^{\text{IND-CPA}_{\mathsf{PR}}}(\lambda)$ is negligible. In fact, the advantage is zero, giving perfect privacy for s.

5.2. Theorem 1 continued: Privacy of $\mathsf{PR}_{\mathsf{MKS}}$

To make PR interoperate with MKS, the blinding factor σ_V needs to be removed from V (i.e. set $\sigma_V = 0$, thus $V = g^s$; also remove σ_W from W by $\sigma_W = 0$, $W = g^{s^n}$). Otherwise, the users of the b-directional mix net provided by MKS can not get to know the value of g^s which they need to be able to decrypt their ciphertexts.

In MKS, s is randomly sampled by the shuffler, so it is no longer possible to adversarially choose some s in $\mathsf{PR}_{\mathsf{MKS}}$. In this case, privacy means that it is not possible to calculate s from n, V and W which can be reduced to the discrete logarithm problem. As we assume the discrete logarithm problem to be hard, privacy of $\mathsf{PR}_{\mathsf{MKS}}$ has been proved.

5.3. Theorem 2: Security.

We prove security of PR and $\mathsf{PR}_{\mathsf{MKS}}$ in the following three sections:

5.4. Security: Perfect Completeness

According to the definition of perfect completeness 3.3, given $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$, we first need to show "*correct statement* $\Rightarrow$ $\mathcal{V}$ *accepts*":

For all statements u where $s_n = s^n$ in $W = g^{s_n} \cdot h^{\sigma_W}$, and for all possible witnesses w for statement u, it has to be proved: $\mathcal{V}$ *accepts the argument made by interaction of* $\mathcal{P}(\sigma, u, w)$ *and* $\mathcal{V}(\sigma, u)$.
When $\mathcal{P}$ and $\mathcal{V}$ are executing the PR protocol 1, for all possible randomizers σ_V, σ_W, α_W, β_W, $\boldsymbol{s}_{L,W}$, $\boldsymbol{s}_{R,W}$, $\tau_{1,W}$, $\tau_{2,W}$ of (Pedersen) commitments V, W, A_W, S_W, $T_{1,W}$, $T_{2,W}$, $\hat{t_W}$ that $\mathcal{P}$ sends to $\mathcal{V}$, and all possible random challenges y, z, w and x of $\mathcal{V}$, the verifier's checks (steps 13. and 21.) pass for PR (or $\mathsf{PR}_{\mathsf{MKS}}$). Evaluating the protocol, plugging in the definitions of the commitments in $\mathcal{V}$'s checks, and simplifying the resulting statements shows that indeed, $\mathcal{V}$ *accepts the argument made by interaction of* $\mathcal{P}(\sigma, u, w)$ *and* $\mathcal{V}(\sigma, u)$.

And second, we show "*correct statement* $\Leftarrow$ $\mathcal{V}$ *accepts*" which is equivalent to "*incorrect statement* $\Rightarrow$ $\mathcal{V}$ *does not accept*":

For all statements u where $s_n \neq s^n$ in $W = g^{s_n} \cdot h^{\sigma_W}$, and for all possible witnesses w for statement u, it has to be proved: $\mathcal{V}$ *does not accept the argument made by interaction*

of $\mathcal{P}(\sigma, u, w)$ *and* $\mathcal{V}(\sigma, u)$.

Analog to the first part, for all possible randomizers and random challenges, evaluate the protocol and simplify the verifier's checks. The check on $g^{\hat{tW}} h^{\tau_{x,W}}$ is then transformed into $g^{s^n} \stackrel{?}{=} g^{s_n}$ which fails because of the precondition $s_n \neq s^n$.

5.5. Security: Subversion Perfect Special Honest-Verifier Zero-Knowledge

Security from subversion of the CRS for PR and $\mathsf{PR}_{\mathsf{MKS}}$ is equivalent to security of MKS, proved in [1], appendix A. For PR, $\boldsymbol{h}$ can be calculated as *NUMS* generators in Setup which fully removes user generated input from the power relation protocol, hence the CRS check is no longer necessary.

We can now assume the CRS to be secure, thus we are left to show perfect special honest-verifier zero-knowledge.

To prove SHVZK, let an efficient simulator $\mathcal{S}$ output messages with the same probability distribution like the messages that are generated by $\mathcal{P}$ when it interacts with $\mathcal{V}$. $\mathcal{P}$ produces A_W, S_W, $T_{1,W}$, $T_{2,W}$, $\boldsymbol{l}_W$, $\boldsymbol{r}_W$, $\hat{tW}$, $\tau_{x,W}$ and μ_W which are all randomized uniformly over $\mathbb{G}$, $\mathbb{Z}_q^n$ or $\mathbb{Z}_q$. It is therefore sufficient for $\mathcal{S}$ to output messages that are distributed uniformly at random. $\mathcal{S}$ could also choose to simulate the protocol for some random s' to get the same distribution. In both cases, $\mathcal{S}$ is trivially efficient and therefore PR and $\mathsf{PR}_{\mathsf{MKS}}$ is SHVZK.

The inner product protocol is SHVZK by a similar argument.

5.6. Security: Computational Witness-Extended Emulation

Computational witness-extended emulation means soundness of PR and $\mathsf{PR}_{\mathsf{MKS}}$, i.e. the prover can not produce an argument that is accepted by the verifier without knowing a

witness.

The general forking lemma by Bootle et al. [26] states that witness-extended emulation can be proved by finding an algorithm that extracts a witness from any $(n_1 \times n_2 \times \cdot \times n_f)$-tree of transcripts which are accepted by $\mathcal{V}$. The extraction algorithm suffices to succeed with overwhelming probability to prove *computational* witness-extended emulation in Bünz et al.'s relaxed forking lemma stated and proved in [1], appendix D.3, theorem 4.

Overview. To construct an extractor $\mathcal{E}$ for all values of the witness, we work from the end of the argument towards the beginning where s, σ_V and σ_W can finally be calculated. We let $\mathcal{E}$ fork at each random challenge that is generated by the verifier, as often as it is needed to get linear equation systems which allow us to compute the next level of witness values.

For the inner product protocol, there exists an efficient extractor $\mathcal{E}_{\mathsf{InnerProduct}}$ for witnesses $\boldsymbol{l}_W$, $\boldsymbol{r}_W$ and $\hat{t_W}$ such that $\mathcal{V}$'s checks pass, as proved in [5], appendix C. Hence, it suffices to construct $\mathcal{E}$ starting from $\boldsymbol{l}_W$, $\boldsymbol{r}_W$ and $\hat{t_W}$. *For simplicity of notation we omit the W subscripts in the extraction algorithm.*

Extraction Algorithm. Statement V, W and $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$ are given as well as $\boldsymbol{l}_W$, $\boldsymbol{r}_W$ and $\hat{t_W}$ from $\mathcal{E}_{\mathsf{InnerProduct}}$. $\mathcal{E}$ is left to extract α, β, $\boldsymbol{a}_L$, $\boldsymbol{a}_R$, $\boldsymbol{s}_L$, $\boldsymbol{s}_R$, t_1, t_2, τ_1, τ_2, σ_V, σ_W and s. At every time it knows all random challenges and commitments of some valid transcript that he is allowed to rewind according to the forking lemma.

First forking layer:

- rewinding point: at challenge x
- size: 3
- challenge values: x_i, $i \in \{1, 2, 3\}$
- extract: α, β, $\boldsymbol{a}_L$, $\boldsymbol{a}_R$, $\boldsymbol{s}_L$, $\boldsymbol{s}_R$, t_1, t_2, τ_1, τ_2
- implicitly forked variables: $\boldsymbol{l}$, $\boldsymbol{r}$, μ, $\hat{t}$, τ_x

Let $\sim_{\dots}$ (e.g. $\sim_{yzw}$) denote some constant depending on the indexed variables. The value of that constant is not further relevant for this proof. Then, matching exponents in $h^{\mu_i}\boldsymbol{g}^{\boldsymbol{l}_i}(\boldsymbol{h}^*)^{\boldsymbol{r}_i} = AS^{x_i}g^{\sim_{yzw}}(\boldsymbol{h}^*)^{\sim_{yz}}$ yields:

$$\begin{aligned}
\boldsymbol{g} &: \boldsymbol{l}_i = \boldsymbol{a}_L + \boldsymbol{s}_L x_i + \sim_{yzw} \\
\boldsymbol{h} &: \boldsymbol{r}_i = (\boldsymbol{a}_R + \boldsymbol{s}_R x_i) \circ \boldsymbol{y}^n + \sim_{yz} \\
h &: \mu_i = \alpha + \beta x_i
\end{aligned}$$

Let $\boldsymbol{\mu} = \binom{\mu_1}{\mu_2}$, $[M_\mu] = \left(\begin{smallmatrix} 1 & x_1 \\ 1 & x_2 \end{smallmatrix}\right)$, $\boldsymbol{v} = \binom{\alpha}{\beta}$. Solving the linear equation system $[M_\mu]\boldsymbol{\mu} = \boldsymbol{v}$ extracts α and β: $\binom{\alpha}{\beta} = [M_\mu]^{-1}\boldsymbol{\mu}$.

If values from the third transcript x_3 and μ_3 were included into this equation system, with overwhelming probability, it would still have a single solution. With negligible probability it has no solution, meaning a discrete logarithm relation $\boldsymbol{g}^{\boldsymbol{a}_g}\boldsymbol{h}^{\boldsymbol{a}_h} = \boldsymbol{g}^{\boldsymbol{a}'_g}\boldsymbol{h}'^{a_h}, a_g||a_h \neq a'_g||a'_h$ has been found and $\mathcal{E}$ fails.

In a similar way, we define linear equation systems for all $j \in \{1, \dots, n\}$ to extract $\boldsymbol{a}_L$, $\boldsymbol{a}_R$, $\boldsymbol{s}_L$, $\boldsymbol{s}_R$:

$$\begin{aligned}
\begin{pmatrix} a_{L_j} \\ s_{L_j} \end{pmatrix} &= \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix}^{-1} \begin{pmatrix} l_{1_j} - \sim_{yzw_j} \\ l_{2_j} - \sim_{yzw_j} \end{pmatrix} \\
\begin{pmatrix} a_{R_j} \\ s_{R_j} \end{pmatrix} &= \left(y^{j-1} \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix} \right)^{-1} \begin{pmatrix} r_{1_j} - \sim_{yz_j} \\ r_{2_j} - \sim_{yz_j} \end{pmatrix}
\end{aligned}$$

Again, a non-trivial discrete logarithm relation between $\boldsymbol{g}$ and $\boldsymbol{h}$ has been found and $\mathcal{E}$ fails if the linear equation system which includes the third transcript has no solution, which happens with negligible probability.

To extract coefficients t_0, t_1, t_2 of $t(X)$ and corresponding blinding values τ_0, τ_1, τ_2, where τ_0 is a virtual blinding value dependent on σ_V and σ_W, we use two linear equation systems

of degree three based on matching exponents in $g^{\hat{t}}h^{\tau_x} = V^{\sim_{yzw}}W^{\sim_y}g^{\delta_{yzw}}T_1^x T_2^{x^2}$:

$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{pmatrix}^{-1} \begin{pmatrix} \hat{t}_1 \\ \hat{t}_2 \\ \hat{t}_3 \end{pmatrix}$$
$$\begin{pmatrix} \tau_0 \\ \tau_1 \\ \tau_2 \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{pmatrix}^{-1} \begin{pmatrix} \tau_{x_1} \\ \tau_{x_2} \\ \tau_{x_3} \end{pmatrix}$$

With negligible probability, the linear equation system has no solution which implies a non-trivial discrete logarithm relation between g and h is found and $\mathcal{E}$ fails.

Second forking layer:

- rewinding point: at challenge z
- size: 2
- challenge values: z_i, $i \in \{1, 2\}$
- extract: s, σ_V, σ_W
- additional implicitly forked variables: t_0, τ_0

We extract s and s^n:

$$t_{0_i} = s\left((wy)^{n-1} + nz_i\right) + s^n y^{n-1} + \delta_{yz_i w} \Rightarrow$$
$$\begin{pmatrix} s \\ s^n \end{pmatrix} = \begin{pmatrix} (wy)^{n-1} + nz_1 & y^{n-1} \\ (wy)^{n-1} + nz_2 & y^{n-1} \end{pmatrix}^{-1} \begin{pmatrix} t_{0_1} - \delta_{yz_1 w} \\ t_{0_2} - \delta_{yz_2 w} \end{pmatrix}$$

Similarly we extract σ_V and σ_W:

$$\tau_{0_i} = \sigma_V \left((wy)^{n-1} + nz_i\right) + \sigma_W y^{n-1} \Rightarrow$$
$$\begin{pmatrix} \sigma_V \\ \sigma_W \end{pmatrix} = \begin{pmatrix} (wy)^{n-1} + nz_1 & y^{n-1} \\ (wy)^{n-1} + nz_2 & y^{n-1} \end{pmatrix}^{-1} \begin{pmatrix} \tau_{0_1} \\ \tau_{0_2} \end{pmatrix}$$

With negligible probability, the linear equation systems have no solutions, meaning a discrete logarithm relation for g and h has been found, which lets $\mathcal{E}$ fail.

Now, having found all witness values, $\mathcal{E}$ is left to extract the linear constraints on those values. Since the polynomial $t(X)$ has already been extracted, matching of coefficients on degree zero shows that for all y, z, w

$$\begin{aligned}
&\text{extracted } t_0 = \text{unblinded polynomial } t(0) \\
\Leftrightarrow\ & s\left((wy)^{n-1} + nz\right) + s^n y^{n-1} + \delta_{yzw} = \left\langle \boldsymbol{a}_L + z\boldsymbol{y}^{-n} + \boldsymbol{w}^n - \boldsymbol{w}_0^{n-1}, \boldsymbol{a}_R \circ \boldsymbol{y}^n - \boldsymbol{y}_z^{n-1} \right\rangle \\
\Leftrightarrow\ & s\left((wy)^{n-1} + nz\right) + s^n y^{n-1} \\
& = \left\langle \boldsymbol{a}_L + z\boldsymbol{y}^{-n} + \boldsymbol{w}^n - \boldsymbol{w}_0^{n-1}, \boldsymbol{a}_R \circ \boldsymbol{y}^n - \boldsymbol{y}_z^{n-1} \right\rangle - \delta_{yzw} \\
& = \left[a_{L_n} a_{R_n} y^{n-1}\right]_1 + \left[\sum_{i=1}^{n-1} y^{i-1}\left(a_{L_i} a_{R_i} - a_{L_{i+1}}\right)\right]_2 - z + \left[z \sum_{i=1}^{n} a_{R_i}\right]_3 - z(z + (n-1)y^{-1}) \\
& + \left[(wy)^{n-1} a_{R_n}\right]_4 + \left[\sum_{i=1}^{n-1} (wy)^{i-1}\left(a_{R_i} - a_{R_{i+1}}\right)\right]_5 - z + (y^{-1} + w)\sum_{i=0}^{n-2} (wy)^i - \delta_{yzw} \\
& = [\cdot]_1 + [\cdot]_2 + [\cdot]_3 + [\cdot]_4 + [\cdot]_5 \\
\Leftrightarrow\ & s^n y^{n-1} = [\cdot]_1 \wedge \\
& 0 = [\cdot]_2 \wedge \\
& snz = [\cdot]_3 \wedge \\
& s(wy)^{n-1} = [\cdot]_4 \\
& \wedge\ 0 = [\cdot]_5
\end{aligned}$$

where brackets 1 to 5 correspond to the following linear constraints on $\boldsymbol{a}_L$ and $\boldsymbol{a}_R$:

$$
\begin{aligned}
a_{L_n} a_{R_n} &= s^n \\
\forall i \in \{1, \ldots, n-1\} : a_{L_i} a_{R_i} &= a_{L_{i+1}} \\
mean(\boldsymbol{a}_R) &= s \\
a_{R_n} &= s \\
\forall i \in \{1, \ldots, n-1\} : a_{R_i} &= a_{R_{i+1}}
\end{aligned}
$$

which are the constraints presented in section 4.3. (the 3rd $mean()$ constraint can be ignored because it follows from the 4th and 5th constraint).

$\mathcal{E}$ extracts constraints enforcing $\boldsymbol{a}_R = (s, s, \ldots, s)$ from $[\cdot]_4$ and $[\cdot]_5$ by forking n times on challenge w (third forking layer). $\mathcal{E}$ fails by finding a discrete logarithm relation between g and h if the linear equation system does not have a solution with negligible probability.

Finally, $\mathcal{E}$ forks n times on y to extract constraints enforcing $\boldsymbol{a}_L = (1, s, s^2, \ldots, s^{n-1})$ or fail similar to the previous step with negligible probability.

This marks the end of the extraction algorithm $\mathcal{E}$ that successfully recovered a valid witness and its linear constraints. $\mathcal{E}$ forked $n \cdot n \cdot 2 \cdot 3 = 6n^2$ times and on each forking it executed a polynomial time algorithm. By the weakened forking lemma this proves that argument PR has computational witness-extended emulation. $\mathsf{PR}_{\mathsf{MKS}}$ is a special case of PR and therefore inherits the property of computational witness-extended emulation.

■

6. Security

This section focuses on security of implementation, common security pitfalls in zero-knowledge cryptography and how to exploit them. If any of the assumptions explained in 3.2 does not hold, security as defined in 3.6 and proved in 5 is lost.

6.1. Assumptions and Keylengths

Discrete Logarithm Assumption. It is an unsolved problem in computer science if the discrete logarithm modulo a prime number can be computed in polynomial time in the number of digits of the modulus. The most efficient methods to calculate the discrete logarithm are index calculus (idea by Western and Miller [27] optimized and presented in today's form by Adleman [28]), adaptations of the number field sieve [29, 30] and the function field sieve [31].

These efficient algorithms are not applicable to elliptic curves. The exponential Pollard-Rho algorithm [32] is the fastest algorithm known to calculate the elliptic curve discrete logarithm in general. Not relevant for our implementation, but on groups with smooth order the specialized Pohlig-Hellman algorithm [33] can be used to significantly speed up Pollard-Rho.

In practice it is assumed that the elliptic curve discrete logarithm is hard to compute when group size is large enough.

Elliptic Curve Cryptography. The main focus of this thesis lies on the MKS and PR protocols which abstract from the choice of the finite prime order group $\mathbb{G}$. *SafeCurves* [16] by Bernstein et al. makes very restrictive recommendations on which elliptic curves should be used for most secure cryptography. There, no common elliptic curve with Weierstrass representation is seen to fulfill the highest security standards, partially because of difficulty of mitigating side-channel attacks based on addition laws which are not well-suited to be implemented in a fixed number of CPU cycles with Weierstrass curve representations. According to *SafeCurves*, Edwards and Montgomery representations are to be preferred for maximum security, often without efficiency losses.

Elliptic curve cryptography is still being actively researched and real-world applications use many different elliptic curves [21, 18, 34, 17] which may not fulfill the highest standards according to *SafeCurves* from case to case. For example, Bitcoin uses Weierstrass curve secp256k1 which we consider sufficient for the purpose of a proof-of-concept implementation.

For real-world implementations of MKS, security hardened libraries should be used. It is also necessary to choose secure and efficient elliptic curve point encodings and always verify validity of points received from possibly insecure sources. For example, to handle the cofactor of Curve25519 (which has Montgomery representation), dalek's Bulletproofs implementation [35] uses the Ristretto encoding [19] (updated in 2023) which is based on Hamburg's 2015 Decaf point compression [20].

In mathematics, linear equation systems are understood exhaustively and most open questions on quadratic forms are answered since their classification (Hilbert's 11th problem) has been solved in the early 20th century mainly by Minkowski and Hasse. Cubic forms can not be said to be understood as many questions that are answered for lower degree forms remain open. While cubic equations (unary cubic forms) were solved by Cardano in 1545 by the formula named after him and the later invention of complex numbers, many basic questions remain open for elliptic curves (binary cubic forms). As active research on elliptic curves is ongoing, steady mathematical progress or sudden

breakthroughs on understanding elliptic curves could possibly weaken or break the security of curves that are assumed to be secure under today's security assumptions (e.g. a sub-exponential algorithm might be discovered).

Schnorr Groups. When instead of elliptic curves using a Schnorr group [4] as cryptographic primitive, while still not proved, the mathematical reasons for its security are understood much better. For randomly seeded (i.e. not chosen to have a special form) prime numbers p and q satisfying $p = h \cdot q + 1, h \in \mathbb{Z}$, choose some $g \in \mathbb{Z}_p^*$ ($\mathbb{Z}_p^*$ is the finite cyclical multiplicative group of order p) such that $\exists b \in \mathbb{Z}_p^* : e^h \equiv g \mod p$. Then, $\mathbb{G} \equiv g^{\mathbb{Z}_q} \mod p$ is a finite cyclical subgroup of $\mathbb{Z}_p^*$ of prime order q. Equivalent to elliptic curve cryptography, q needs to be chosen large enough to withstand the Pollard-Rho algorithm, but in addition, prime modulus p also needs to be large enough to be secure from number field sieve attacks which is a sub-exponential algorithm.

Practical recommendations by German BSI (Bundesamt für Sicherheit in der Informationstechnik) [36] on keylengths are 256 bits for q and 256 bits for prime modulus p of elliptic curve or 3200 bits for prime modulus p of the Schnorr group which we comply with in the implementation. Other sources [21, 37] recommend similar keylengths.

6.2. Quantum Security

Both Bulletproofs and multi-key shuffles rely on the hardness of computing a discrete logarithm relation. Based on Shor's algorithm for efficient factorization of integers [38], Proos et al. introduced a quantum polynomial algorithm that solves the discrete logarithm problem [39]. This breaks a central security assumption of the multi-key shuffle and leaves it vulnerable to attacks with quantum computers of sufficient size.

The number of quantum bits needed for this algorithm to break cryptosystems with keylengths considered secure today is much higher by orders of magnitude than what has been achieved by any publicly known quantum computers so far.

For today's security standards a quantum computer with a number of error corrected qubits in the magnitude of 1.000 to 10.000 would be needed to break today's cryptography based on elliptic curves [40] or prime factorization [41] [42]. Furthermore, there exist special prime numbers that require quantum computers with perfect error correction such that Shor's algorithm could break composites of them [43]. While some pre-selected numbers have been factorized by quantum algorithms adapted for those specific numbers, up until today, no quantum computer is known to have ever run a full implementation of Shor's algorithm successfully [44] [45].

Therefore, while the multi-key shuffle does not have mathematical post-quantum security, this does not impair its security in real-world applications at the moment. It is safe to say that quantum vulnerability can not be expected in the near future.

6.3. Implementation Issues

Weak security parameter. In practice, choosing correct security parameters can be challenging. Often, security of cryptanalysis has to be traded off against performance and hardware limitations. If keylengths are chosen too short, a prover with large enough computational resources may be able to find a discrete logarithm relation between generators. We look at the consequences in the following paragraphs.

Discrete Logarithm Relation – Incorrect Sampling of Generators for Pedersen Commitments. If the generators for Pedersen commitments are not sampled at random and independent of each other but instead are based on a common generator (which is a wrong implementation of Setup), a discrete logarithm relation between the generators is known.

As a consequence, all Pedersen commitments lose their binding property and a malicious prover may exchange underlying values dependent on which party she communicates with. Especially, she might use fraudulent ones for transactions but reveal honestly looking values on inspection which perfectly hides the fraudulent behaviour for ever.

Weak Random Number Generators. Should the prover use a weak random number generator to sample blinding values, the verifier is enabled to efficiently guess or calculate the prover's secret blinding values, then unblind her commitments and might guess her secret from a set of probable or possible values.

Should the verifier use a weak random number generator to sample challenges, the prover is enabled to efficiently guess them before she sends any commitments to the verifier. With the foreknowledge of the challenges she could then simulate an argument with witness w' based on a permutation π' that she did not apply and then extract a correct witness $\hat{w}$ from the transcript using the extractor shown in the proof of computational witness-extended emulation. She then starts interacting with the verifier which will produce the transcript she already prepared. On inspection, she can reveal $\hat{w}$ with some permutation $\hat{\pi}$ despite her in fact applying π'.

Fiat-Shamir Transform and Frozen-Heart Attacks. If the implementation of the Fiat-Shamir transform misses including one of the inputs to the argument or one of the messages exchanged between prover and verifier, the protocol becomes vulnerable to forged proofs. For example, if a transcript omits V, the prover can first simulate the argument for an arbitrary s' and then extract a valid witness from the transcript with some $\hat{V} = g^{\hat{s}}$. The fraud scheme works the same as if the verifier used a weak random number generator.

This attack is called a "frozen heart attack" because the Fiat-Shamir transform, which can be considered to be at the heart of non-interactive proof systems, is frozen by holding the prover's commitments constant.

The frozen heart attack has been first described after it was found that the first version of Bulletproofs [5] forgot to include V in the Fiat-Shamir transform. Production implementations with this vulnerability existed which lead to major attention in the field [46].

6.4. Practical Considerations

User-Shuffler Collusion. To enable an untrusted setup of the common reference string, it must be assumed that there is no collusion between user and shuffler. If this assumption is violated, the shuffler can work together with users to get to know a logarithm relation between generators for Pedersen commitments with all fatal consequences described above.

Number of Concatenated Mix nets. The depth of the concatenated mix nets as well as the corresponding shufflers having contrary interests is vital for the security of the complete mix net. While it is sufficient that only one of all the mix nets shufflers executes a truly random shuffle for the whole mix net, if all shufflers are acting dishonest, privacy of all users of the system is compromised. In addition, as there must not be a common interest between users and shufflers in using the mix net, a suitable system of incentives should be considered for real-world instances of a MKS network.

7. Implementation

Running the implementation with the trial version of Mathematica is possible. When using a single machine, the notebooks can be run in non-interactive mode; With multiple machines, the whole feature set of the software can be used, including interactive mode.

7.1. Previous Work

This master's thesis is not a standalone project. The implementation is based on previous projects on zero-knowledge cryptography: First, my 2023 study project "Attacking Bulletproofs: Exploiting Common Security Pitfalls in Zero-Knowledge Cryptography" which implemented the Bulletproofs range proof protocol by Bünz et al. [5] with Mathematica notebooks and introduced a routing server. Second, my 2023 Lab Course Project "Implementing a Router and Inspector for Testing Cryptographic Protocols" which is a major improvement on the study project's framework and made the communication server as well as underlying Mathematica libraries more feature-rich and robust. The implementation of the multi-key shuffle uses almost the complete feature set which has been implemented in the Lab Course project.

7.2. Implementation Goals

The implementation was used as a starting challenge to get to understand all aspects of the multi-key shuffle and verify correctness of the protocol and possible use cases of the

multi-key shuffle. Then, the power relation protocol was mathematically developed and integrated into this implementation.

Based on the knowledge gained, formal proofs of the power relation protocol were developed. The notebooks are easy to debug, contain sanity checks where possible and are developed to be extendable.

The proof-of-concept implementation can be considered as a starting point for more efficient implementations.

7.3. System Architecture and Components

Figure 3 illustrates the system's architecture and components. In detail, the multi-key shuffle involves three parties:

Users. Multiple users (with public keys $\boldsymbol{d}$) create ElGamal ciphertexts $(\boldsymbol{b}, \boldsymbol{c})$ which they want to be sent anonymously to other users (with public keys $\boldsymbol{h}$). They send those messages to a shuffler or a network of shufflers which sends them back a multi-key shuffle of the original messages. The users identify and decrypt their messages and can create reply messages.

Again, it must be assumed that there is no collusion between users and the shuffler(s).

Prover/Shuffler. On input $(\boldsymbol{h}, (\boldsymbol{b}, \boldsymbol{c}), \boldsymbol{d})$, the shuffler performs a multi-key shuffle $(\boldsymbol{h}', (\boldsymbol{b}', \boldsymbol{c}'), \boldsymbol{d}')$ and sends it back to the users (or to the next shuffler in a mix-net; this kind of multi-stage shuffling has not been implemented). The shuffler creates a zero-knowledge proof of the correctness of his multi-key shuffle, either with the help of an interactive verifier or by proving himself with random challenges generated by the Fiat-Shamir transform.

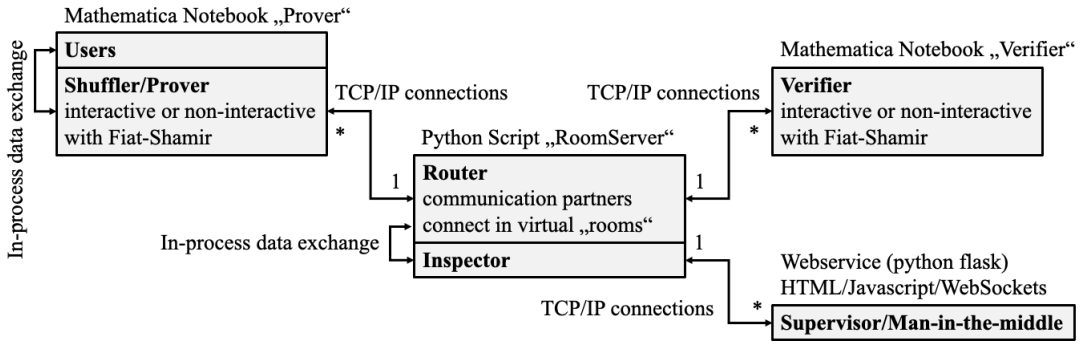

Figure 3.: System components

Verifier. The verifier sends random challenges to the prover to help her argue the correctness of the multi-key shuffle. The verifier verifies the proof generated by the shuffler; With Fiat-Shamir transform applied, she verifies proofs at an arbitrary time.

For simplicity of implementation, the role of the users has been integrated into the shuffler's/prover's notebook. It would be possible to create a separate notebook which would be executed by each user separately. To achieve this, the communication library and the communication server script needed to be extended to incorporate a user role.

Another component is introduced to handle communication between prover and verifier:

Communication. Messages are routed over a server component to achieve strong separation between the parties and showcase real-world communication scenarios. It is implemented in the python scripting language and exposes two services on two different ports: First, the router component which the parties use to communicate with each other in virtual *rooms*, and second, an insight component which acts as a man-in-the-middle and publishes all messages that are exchanged via the router to a web service. That insight web service provides real-time supervision of protocol execution and allows the communicating parties to publish messages for debugging or additional logging.

7.4. Features, Configuration Options and Programming Paradigms

To summarize and be complete, the implementation of the prover, users and verifier with Mathematica notebooks which communicate via a python server, leveraging networking and web technologies, provides the following feature set:

- state machine to allow for re-winding the protocol to different stages, achieving easy restarting, debugging and development,
- single kernel configuration option to allow running the notebooks on the free version of Mathematica,
- setup of trapdoor function (Elliptic Curve Cryptography or Schnorr Groups),
- *NUMS* generators for commitments $\mathsf{Setup}(1^\lambda) \to (\mathbb{G}, q, g, h, \boldsymbol{g})$,
- event handler, i.e. for verification results, inspection requests and error messages,
- multi-key shuffle proof with power relation protocol,
- options to run with or without inner product protocol,
- option to run in interactive mode or with Fiat-Shamir enabled,
- sanity checks for prover,
- detailed printouts of all variables and formulas used by prover respectively verifier,
- proof verification,
- inspection of prover's commitments,
- configuration of random number generator's method,
- tracking of communication cost,
- tracking of entropy to expose possible security vulnerabilities on the implementation of Fiat-Shamir,
- Fiat-Shamir state tracking and printouts,
- real-time updates on the protocol state and variables exchanged over the router server via websockets.

8. Conclusions and Future Work

To the best of our knowledge this thesis is the first publication about the completed multi-key shuffle argument and also its first implementation. The multi-key shuffle protocol is ready to be used for real world applications for privacy in communication.

The Bulletproofs paradigm again proves to be a powerful tool for developing zero-knowledge protocols. Toolchains for the development of complex zero-knowledge protocols could increase speed of implementation, mathematical verification and security of cryptographic software dramatically. A standardized language and further formalization of zero-knowledge protocols could help develop systems to automatically proof their correctness and security properties.

The masked single-key shuffle for mental card games remains an unsolved problem for Bulletproof protocols.

Regarding the power relation protocol, prover and verifier times are linear in the absolute value of the exponent which is sufficient for the use case of the multi-key verifiable shuffle. However, when used as a standalone protocol, the power relation protocol can not handle large numbers efficiently. It may be possible to further reduce the size of the power relation argument by working on a binary representation of the exponent n, which would result in prover and verifier times in $\mathcal{O}(\log n)$ and an argument size in $\mathcal{O}(\log \log n)$.

9. Acknowledgments

I would like to thank

- Prof. Dr. Schindelhauer for his introduction to cryptography, for his individual support, for giving me the freedom to choose the topics I most wanted to work on, and for his trust in me to tackle problems in a field I have just began to explore,
- Prof. Dr. Jörn Müller-Quade and Dr. Willi Geiselmann for evaluating my master's thesis as second examiner in the short term,
- Sneha Mohanty for the organization and support,
- My family and friends for their feedback and advice,
- My fiancé for her exceptional support and companionship during my study period.

Bibliography

[1] B. Bünz, M. Raykova, and J. Sarathy, "Efficient multi-key verifiable shuffles from short arguments for randomized algorithms," 2023. https://scholar.harvard.edu/files/jsarathy/files/bp_shuffle_draft_4.pdf, Accessed 2023-07-26.

[2] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems, 17th," in *Annual ACM Symp. on Theory of Computing*, vol. 10, 1985.

[3] J. Kilian, "A note on efficient zero-knowledge proofs and arguments," in *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pp. 723–732, 1992.

[4] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of cryptology*, vol. 4, pp. 161–174, 1991.

[5] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE symposium on security and privacy (SP)*, pp. 315–334, IEEE, 2018.

[6] M. Hoffmann, M. Klooß, and A. Rupp, "Efficient zero-knowledge arguments in the discrete log setting, revisited," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2093–2110, 2019.

[7] L. Eagen, S. Kanjalkar, T. Ruffling, and J. Nick, "Bulletproofs++," *Cryptology ePrint Archive*, 2022.

[8] C. Schindelhauer, "A toolbox for mental card games," *Medizinische Universität Lübeck*, 1998.

[9] A. Barnett and N. P. Smart, "Mental poker revisited," in *Cryptography and Coding: 9th IMA International Conference, Cirencester, UK, December 16-18, 2003. Proceedings 9*, pp. 370–383, Springer, 2003.

[10] H. Stamer, "Efficient electronic gambling: an extended implementation of the toolbox for mental card games," in *WEWoRC 2005–Western European Workshop on Research in Cryptology*, Gesellschaft für Informatik eV, 2005.

[11] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle," in *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 263–280, 2012.

[12] J. Groth, "Linear algebra with sub-linear zero-knowledge arguments," in *Advances in Cryptology-CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pp. 192–208, 2009.

[13] J. Groth, "A verifiable secret shuffle of homomorphic encryptions," *Journal of Cryptology*, vol. 23, no. 4, 2010.

[14] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[15] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*, pp. 186–194, Springer, 1986.

[16] D. J. Bernstein and T. Lange, "Safecurves: choosing safe curves for elliptic-curve cryptography." https://safecurves.cr.yp.to, Accessed 2023-07-23.

[17] D. J. Bernstein, C. Chuengsatiansup, and T. Lange, "Curve41417: Karatsuba revisited," in *Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16*, pp. 316–334, 2014.

[18] M. Lochter and J. Merkle, "Elliptic curve cryptography (ecc) brainpool standard curves and curve generation," RFC 5639, 2010.

[19] H. de Valence, J. Grigg, M. Hamburg, I. Lovecruft, G. Tankersley, and F. Valsorda, "Internet draft for ristretto group," 2023. https://datatracker.ietf.org/doc/draft-irtf-cfrg-ristretto255-decaf448/, Accessed 2023-09-07.

[20] M. Hamburg, "Decaf: Eliminating cofactors through point compression," in *Advances in Cryptology–CRYPTO 2015*, pp. 705–723, Springer Berlin Heidelberg, 2015.

[21] D. Brown, "Standards for efficient cryptography 2 (sec 2)," *Certicom Research*, vol. 2, no. Sec 2, 2010.

[22] T. Kivinen and M. Kojo, "More modular exponential (modp) diffie-hellman groups for internet key exchange (ike)," RFC 3526, 2003. https://datatracker.ietf.org/doc/rfc3526/, Accessed 2023-07-23.

[23] R. A. DeMillo and R. J. Lipton, "A probabilistic remark on algebraic program testing," *Inf. Process. Lett.*, vol. 7, no. 4, pp. 193–195, 1978.

[24] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *International symposium on symbolic and algebraic manipulation*, pp. 216–226, Springer, 1979.

[25] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.

[26] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting," in *Advances in*

Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35, pp. 327–357, Springer, 2016.

[27] A. E. Western and J. C. Miller, "Tables of indices and primitive roots," *Royal Society, Mathematical Tables: Volume 9, Indices and Primitive Roots*, 1968.

[28] L. Adleman, "A subexponential algorithm for the discrete logarithm problem with applications to cryptography," in *20th Annual Symposium on Foundations of Computer Science (SFCS 1979)*, pp. 55–60, IEEE Computer Society, 1979.

[29] A. K. Lenstra, W. Hendrik Jr, *et al.*, *The development of the number field sieve*, vol. 1554. Springer Science & Business Media, 1993.

[30] C. Pomerance, "A tale of two sieves," *Notices of the American Mathematical Society*, vol. 43, no. 12, pp. 1473–1485, 1996.

[31] L. M. Adleman and M.-D. A. Huang, "Function field sieve method for discrete logarithms over finite fields," *Information and Computation*, vol. 151, no. 1-2, pp. 5–16, 1999.

[32] J. M. Pollard, "Monte carlo methods for index computation ( )," *Mathematics of computation*, vol. 32, no. 143, pp. 918–924, 1978.

[33] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms gf(p) and its cryptographic significance," *IEEE Transactions on Information Theory*, vol. 24, no. 1, p. 106–110, 1978.

[34] A. Langley, M. Hamburg, and S. Turner, "Elliptic curves for security," RFC 7748, 2016. https://datatracker.ietf.org/doc/rfc7748/, Accessed 2023-12-11.

[35] C. Yun, H. de Valence, O. Andreev, *et al.*, "Bulletproofs," 2018.

[36] B. für Sicherheit in der Informationstechnik, "Bsi - technical guideline, cryptographic mechanisms: Recommendations and key lengths," 2023.

[37] D. Giry, "Cryptographic key length recommendation (nist recommendations 2020)," 2020. https://www.keylength.com/en/4/, Accessed 2023-07-23.

[38] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.

[39] J. Proos and C. Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," *arXiv preprint quant-ph/0301141*, 2003.

[40] M. Roetteler, M. Naehrig, K. M. Svore, and K. Lauter, "Quantum resource estimates for computing elliptic curve discrete logarithms," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pp. 241–270, Springer, 2017.

[41] E. Gerjuoy, "Shor's factoring algorithm and modern cryptography. an illustration of the capabilities inherent in quantum computers," *American journal of physics*, vol. 73, no. 6, pp. 521–540, 2005.

[42] J. Fujitsu Limited, Tokyo, "Fujitsu quantum simulator assesses vulnerability of rsa cryptosystem to potential quantum computer cryptography threat." https://www.fujitsu.com/global/about/resources/news/press-releases/2023/0123-01.html, Accessed 2023-12-10.

[43] J.-Y. Cai, "Shor's algorithm does not factor large integers in the presence of noise," *arXiv preprint arXiv:2306.10072*, 2023.

[44] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, no. 6866, pp. 883–887, 2001.

[45] M. Amico, Z. H. Saleem, and M. Kumph, "Experimental study of shor's factoring algorithm using the ibm q experience," *Physical Review A*, vol. 100, no. 1, p. 012305, 2019.

[46] J. Miller, "The frozen heart vulnerability in bulletproofs (blog post)." https://blog.trailofbits.com/2022/04/15/the-frozen-heart-vulnerability-in-bulletproofs/, Accessed 2023-07-23.

A. Appendix: File Tree of Attachments

```
/Implementation
    /MultiKeyShuffle-Prover.nb
    /MultiKeyShuffle-Verifier.nb
    /README.txt
    /Test Runs
        /MKS-2.mp4
        /MKS-3.mp4
        /MKS-4.mp4
        /MKS-5.mp4
        /MKS-6.mp4
        /MKS-7.mp4
        /MKS-8.mp4
        /MKS-9.mp4
        /MKS-16-very-slow.mp4
    /Transcripts
        /MultiKeyShuffle-Prover_evaluated.nb
        /MultiKeyShuffle-Prover_evaluated.pdf
        /MultiKeyShuffle-Verifier_evaluated.nb
        /MultiKeyShuffle-Verifier_evaluated.pdf
```