# Efficient Zero-Knowledge Argument for Correctness of a Shuffle

Stephanie Bayer Jens Groth

University College London[*]
{s.bayer,j.groth}@cs.ucl.ac.uk

**Abstract**

Mix-nets are used in e-voting schemes and other applications that require anonymity. Shuffles of homomorphic encryptions are often used in the construction of mix-nets. A shuffle permutes and re-encrypts a set of ciphertexts, but as the plaintexts are encrypted it is not possible to verify directly whether the shuffle operation was done correctly or not. Therefore, to prove the correctness of a shuffle it is often necessary to use zero-knowledge arguments.

We propose an honest verifier zero-knowledge argument for the correctness of a shuffle of homomorphic encryptions. The suggested argument has sublinear communication complexity that is much smaller than the size of the shuffle itself. In addition the suggested argument matches the lowest computation cost for the verifier compared to previous work and also has an efficient prover. As a result our scheme is significantly more efficient than previous zero-knowledge schemes in literature.

We give performance measures from an implementation where the correctness of a shuffle of 100,000 ElGamal ciphertexts is proved and verified in around 2 minutes.

**Keywords:** Shuffle, zero-knowledge, ElGamal encryption, mix-net, voting, anonymous broadcast.

## 1 Introduction

A mix-net [Cha81] is a multi-party protocol which is used in e-voting or other applications which require anonymity. It allows a group of senders to input a number of encrypted messages to the mix-net, which then outputs them in random order. It is common to construct mix-nets from shuffles.

Informally, a shuffle of ciphertexts $C_1, \ldots C_N$ is a set of ciphertexts $C'_1, \ldots, C'_N$ with the same plaintexts in permuted order. In our work we will examine shuffle protocols constructed from homomorphic encryption schemes. That means for a given public key $pk$, messages $M_1, M_2$ and randomness $\rho_1, \rho_2$ the encryption function satisfies $\mathcal{E}_{pk}(M_1 M_2; \rho_1 + \rho_2) = \mathcal{E}_{pk}(M_1; \rho_1)\mathcal{E}_{pk}(M_2; \rho_2)$. Thus, we may construct a shuffle of $C_1, \ldots, C_N$ by selecting a permutation $\pi \in \Sigma_N$ and randomizers $\rho_1, \ldots \rho_N$, and calculating $C'_1 = C_{\pi(1)}\mathcal{E}_{pk}(1; \rho_1), \ldots, C'_N = C_{\pi(N)}\mathcal{E}_{pk}(1; \rho_N)$.

A common construction of mix-nets is to let the mix-servers take turns in shuffling the ciphertexts. If the encryption scheme is semantically secure the shuffle $C'_1, \ldots, C'_N$ output by a mix-server does not reveal the permutation or the messages. But this also means that a malicious mix-server in the mix-net could substitute some of the ciphertexts without being detected. In a voting protocol, it could for instance replace all ciphertexts with encrypted votes for candidate X. Therefore, our goal is to construct an interactive argument that makes it possible to verify that the shuffle was done correctly (soundness), but reveals nothing about the permutation or the randomizers used (zero-knowledge).

[*]Both authors are supported by EPSRC grant number EP/G013829/1.

Efficiency is a major concern in arguments for the correctness of a shuffle. In large elections it is realistic to end up shuffling millions of votes. This places considerable strain on the performance of the zero-knowledge argument both in terms of communication and computation. We will construct a zero-knowledge argument for correctness of a shuffle that is highly efficient both in terms of communication and computation.

## 1.1 Related work

The idea of a shuffle was introduced by Chaum [Cha81] but he didn't give any method to guarantee the correctness. Many suggestions had been made how to build mix-nets or prove the correctness of a shuffle since then, but many of these approaches have been partially or fully broken, and the remaining schemes sometimes suffer from other drawbacks. The scheme of Desmedt and Kurosawa [DK00] assumed that only a small number of mix-servers are corrupt. The approach of Jakobson, Juels, and Rivest [JJR02] needed a relatively big number of mix-server to minimize the risk of tampering with messages or compromising privacy of the senders. Peng et al. [PBDV04] restrained the class of possible permutations and also required that a part of the senders are honest. None of these drawbacks are suffered by the shuffle scheme of Wikström [Wik02] and approaches based on zero-knowledge arguments. Since zero-knowledge arguments achieve better efficiency they will be the focus of our paper.

Early contributions using zero-knowledge arguments were made by Sako and Killian [SK95] and Abe [Abe98, Abe99, AH01]. Furukawa and Sako [FS01] and Neff [Nef01, Nef03] proposed the first shuffles for ElGamal encryption with a complexity that depends linearly on the number of ciphertexts.

Furukawa and Sako's approach is based on permutation matrices and has been refined further [Fur05, GL07]. Furukawa, Miyachi, Mori, Obana and Sako [FMM$^+$02] presented an implementation of a shuffle argument based on permutation matrices and tested it on mix-nets handling 100,000 ElGamal ciphertexts. Recently, Furukawa and Sako [FMS10] have reported on another implementation based on elliptic curve groups.

Wikström [Wik09] also used the idea of permutation matrices and suggested a shuffle argument which splits in an offline and online phase. Furthermore, Terelius and Wikström [TW10] constructed conceptually simple shuffle arguments that allowed the restriction of the shuffles to certain classes of permutations. Both protocols are implemented in the Verificatum mix-net library [Wik10].

Neff's approach [Nef01] is based on the invariance of polynomials under permutation of the roots. This idea was picked up by Groth who suggested a perfect honest verifier zero-knowledge protocol [Gro10]. Stamer [Sta05] reported on an implementation of this scheme. Later Groth and Ishai [GI08] proposed the first shuffle argument where the communication complexity is sublinear in the number of ciphertexts.

## 1.2 Our contribution

**Results.** We propose a practical zero-knowledge argument for the correctness of a shuffle. We cover the case of shuffles of ElGamal ciphertexts but it is possible to adapt our argument to other homomorphic cryptosystems as well.

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

The disadvantage of Groth and Ishai's argument compared to the schemes with linear communication was that the prover's computational complexity was on the order of $O(Nm)$ exponentiations. It was therefore only possible to choose small $m$. In comparison, our prover's computational complexity is $O(N \log m)$ exponentiations for constant round arguments and $O(N)$ exponentiations if we allow a logarithmic number of rounds. In practice, we do not need to increase the round complexity until $m$ gets quite large, so the speedup in the prover's computation is significant compared to Groth and Ishai's work and is comparable to the complexity seen in arguments with linear communication. Moreover, the verifier is fast in our argument making the entire process very light from the verifier's point of view.

In Section 6 we report on an implementation of our shuffle argument using shuffles of 100,000 ElGamal ciphertexts. We compare this implementation on the parameter setting for ElGamal encryption used in [FMM$^+$02] and find significant improvements in both communication and computation. We also compare our implementation to the shuffle argument in the Verificatum mix-net [Wik10] and find significant improvements in communication and moderate improvements in computation.

**New techniques.** Groth [Gro09] proposed efficient sublinear size arguments to be used in connections with linear algebra over a finite field. We combine these techniques with Groth and Ishai's sublinear size shuffle argument. The main problem in applying Groth's techniques to shuffling is that they were designed for use in finite fields and not for use with group elements or ciphertexts. It turns out though that the operations are mostly linear and therefore it is possible to carry them out "in the exponent"; somewhat similar to what is often done in threshold cryptography. Using this adaptation we are able to construct an efficient multi-exponentiation argument that a ciphertext $C$ is the product of a set of known ciphertexts $C_1, \ldots, C_N$ raised to a set of hidden committed values $a_1, \ldots, a_N$. This is the main bottleneck in our shuffle argument and therefore gives us a significant performance improvement.

Groth's sublinear size zero-knowledge arguments also suffered from a performance bottleneck in the prover's computation. At some juncture it is necessary to compute the sums of the diagonal strips in a product of two matrices. This problem is made even worse in our setting because when working with group elements we have to compute these sums in the exponents. By adapting techniques for polynomial multiplication such as Toom-Cook [Too00, Coo66] and the Fast Fourier Transform [CT65] we are able to reduce this computation. Moreover, we generalize an interactive technique of Groth [Gro09] to further reduce the prover's computation.

**Structure of the paper.** In Section 2 we will give definitions of key concepts used in the paper. In Section 3 we explain how to construct the full shuffle argument, following the invariance of polynomial roots paradigm of Neff [Nef01]. Section 4 contains a multi-exponentiation argument to hidden committed values, which is a major component in our shuffle argument. We consider this section with the new techniques described above to be our main contribution. In Section 5 we give another major component of our protocol, which is a shuffle argument for hidden committed values. This argument was sketched in Groth [Gro09] but we give all the details and make a few minor improvements. Finally, in Section 6 we report on our implementation and give theoretical and experimental comparisons with other shuffle arguments.

## 2 Preliminaries

In this section we will give the definitions of the key concepts needed. Our protocol works for different types of homomorphic encryption schemes; for example ElGamal encryption. When arguing that a shuffle is correct we will use homomorphic commitment schemes extensively. Finally, we give precise definitions of honest verifier zero knowledge arguments.

### 2.1 Notation

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

We say a function $f : \mathbb{N} \to [0, 1]$ is negligible if $f(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$. We say $1 - f$ is overwhelming if $f$ is negligible. We will give a security parameter $\lambda$ written in unary as input to all parties in our protocols. Intuitively, the higher the security parameter the more secure the protocol. Formally, we define security in the following sections by saying an adversarial algorithm has negligible (in the security parameter) probability of succeeding in its attack.

For vectors of group elements, we write $\vec{x}\vec{y} = (x_1y_1, \ldots, x_ny_n)$ for the entry-wise product and correspondingly $\vec{x}^z = (x_1^z, \ldots, x_n^z)$. We write $\vec{x}_\pi$ if the entries of vector $\vec{x}$ are permuted by the permutation $\pi$, i.e., $\vec{x}_\pi = (x_{\pi(1)}, \ldots, x_{\pi(n)})$. For vectors of field elements, we use the standard inner product $\vec{x} \cdot \vec{y} = \sum_{i=1}^n x_iy_i$.

## 2.2 Homomorphic encryption

Informally, an encryption scheme is homomorphic if for a public key $pk$, messages $M_1, M_2$ and randomness $\rho_1, \rho_2$ the encryption function satisfies $\mathcal{E}_{pk}(M_1M_2; \rho_1 + \rho_2) = \mathcal{E}_{pk}(M_1; \rho_1)\mathcal{E}_{pk}(M_2; \rho_2)$. Our argument works with many different homomorphic encryption schemes where the message space has large prime order $q$, but for notational convenience we will focus just on ElGamal encryption.

Let $\mathbb{G}$ be a cyclic group of large prime order $q$ with generator $G$. The ElGamal encryption scheme [ElG84] in this group works as follows: the secret key $sk = x \in \mathbb{Z}_q^*$ is chosen at random, and the public key $pk$ contains $Y = G^x$. To encrypt a message $M \in \mathbb{G}$ we choose a random $\rho \in \mathbb{Z}_q$ and compute the ciphertext $\mathcal{E}_{pk}(M; \rho) := (G^\rho, Y^\rho M)$ belonging to the ciphertext space $\mathbb{H} = \mathbb{G} \times \mathbb{G}$. To decrypt a ciphertexts $(U, V) \in \mathbb{H}$ we compute $M = VU^{-x}$.

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

$$\begin{aligned} \mathcal{E}_{pk}(M_1M_2; \rho_1 + \rho_2) &= (G^{\rho_1+\rho_2}; Y^{\rho_1+\rho_2}M_1M_2) = (G^{\rho_1}, Y^{\rho_1}M_1)(G^{\rho_2}, Y^{\rho_2}M_2) \\ &= \mathcal{E}_{pk}(M_1; \rho_1)\mathcal{E}_{pk}(M_2; \rho_2). \end{aligned}$$

For $\vec{M} = (M_1, \ldots, M_n)$ and $\vec{\rho} = (\rho_1, \ldots, \rho_n)$ we define $\mathcal{E}_{pk}(\vec{M}; \vec{\rho}) = \big(\mathcal{E}_{pk}(M_1; \rho_1), \ \ldots, \ \mathcal{E}_{pk}(M_m; \rho_m)\big)$. We also define a bilinear map

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

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

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

## 2.3 Homomorphic commitment

We will need a homomorphic commitment scheme in our protocol. A commitment scheme is homomorphic if for a commitment key $ck$, messages $a, b$, and randomizers $r, s$ it holds that $\mathrm{com}_{ck}(a + b; r + s) = \mathrm{com}_{ck}(a; r)\mathrm{com}_{ck}(b; s)$. We also require for our scheme that it is possible to commit to $n$ elements in $\mathbb{Z}_q$, where $q$ is a large prime, at the same time. Many homomorphic commitment schemes with this property can be used, but for convenience we just focus on a generalization of the Pedersen commitment scheme [Ped91].

For a cyclic group $\mathbb{G}$ of large prime order $q$, the general Pedersen commitment scheme works as follows: first the key generation algorithm $\mathcal{K}$ chooses random generators $G_1, \ldots, G_n, H$ of the group $\mathbb{G}$ and sets the commitment key $ck = (\mathbb{G}, G_1, \ldots, G_n, H)$. To commit to $n$ elements $(a_1, \ldots, a_n) \in \mathbb{Z}_q^n$ we pick randomness $r \in \mathbb{Z}_q$ and compute $\mathrm{com}_{ck}(a_1, \ldots, a_n; r) = H^r \prod_{i=1}^n G_i^{a_i}$. We can also commit to less than $n$ elements; this is done by setting the remaining entries to 0. We will always assume that interested parties have verified that commitments belong to the group $\mathbb{G}$.

The commitment is computationally binding under the discrete logarithm assumption, i.e., a non-uniform probabilistic polynomial time adversary has negligible probability of finding two different openings of the same commitment. The commitment scheme is perfectly hiding since the commitment is uniformly distributed in $\mathbb{G}$ no matter what the messages are.

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

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

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

For a matrix $A \in \mathbb{Z}_q^{n \times m}$ with columns $\vec{a}_1, \ldots, \vec{a}_m$ we shorten notation by defining $\mathrm{com}_{ck}(A; \vec{r}) = (\mathrm{com}_{ck}(\vec{a}_1; r_1), \ldots, \mathrm{com}_{ck}(\vec{a}_m; r_m))$. We will also abuse this notation slightly and define the commitment to $\vec{a} \in \mathbb{Z}_q^N$ where $N = mn$ as $\mathrm{com}_{ck}(\vec{a}; \vec{r}) = \big(\mathrm{com}_{ck}(a_1 \ldots a_n; r_1), \ \ldots, \ \mathrm{com}_{ck}(a_{(m-1)n+1} \ldots a_N; r_m)\big)$. Like in the case of encryption, we also define $\vec{c}^{\,\vec{b}} = (c_1, \ldots, c_m)^{(b_1, \ldots, b_m)^T} = \prod_{j=1}^m c_j^{b_j}$ and for a matrix $B$ with columns $\vec{b}_1, \ldots, \vec{b}_m$ we define $\vec{c}^B = \big(\vec{c}^{\,\vec{b}_1}, \ldots, \vec{c}^{\,\vec{b}_m}\big)$. It is useful to observe that the underlying linear algebra behaves nicely, i.e., $\mathrm{com}_{ck}(A; \vec{r})^{\vec{b}} = \mathrm{com}_{ck}(A\vec{b}; \vec{r} \cdot \vec{b})$ and $\mathrm{com}_{ck}(A; \vec{r})^B = \mathrm{com}_{ck}(AB; \vec{r}B)$.

## 2.4 Special honest verifier zero-knowledge argument of knowledge

In the shuffle arguments we consider a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ both of which are probabilistic polynomial time interactive algorithms. We assume the existence of a probabilistic polynomial time setup algorithm $\mathcal{G}$ that when given a security parameter $\lambda$ returns a common reference string $\sigma$.

In our case, the common reference string will be $\sigma = (pk, ck)$, where $pk$ and $ck$ are public keys for the ElGamal encryption scheme and the generalized Pedersen commitment scheme described previously. The encryption scheme and the commitment scheme may use different underlying groups, but we require that they have the same prime order $q$. We will write $\mathbb{G}$ for the group used by the commitment scheme and write $\mathbb{H}$ for the ciphertext space.

The setup algorithm can also return some side-information that may be used by an adversary; however, we require that even with this side-information the commitment scheme should remain computationally binding. The side-information models that the keys may be set up using some multi-party computation protocol that leaks some information, the adversary may see some decryptions or even learn the decryption key, etc. Our protocol for verifying the correctness of a shuffle is secure in the presence of such leaks as long as the commitment scheme is computationally binding.

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

$$L_\sigma := \{x \mid \exists w : (\sigma, x, w) \in R\}$$

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

The public transcript produced by $\mathcal{P}$ and $\mathcal{V}$ when interacting on inputs $s$ and $t$ is denoted by $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t) \rangle$. The last part of the transcript is either accept or reject from the verifier. We write $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle = b, \ b \in \{0, 1\}$ for rejection or acceptance.

**Definition 1.** The triple $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is called an *argument* for a relation $R$ with perfect completeness if for all non-uniform polynomial time interactive adversaries $\mathcal{A}$ we have:
*Perfect completeness:*

$$\Pr[(\sigma, \text{hist}) \leftarrow \mathcal{G}(1^\lambda); (x, w) \leftarrow \mathcal{A}(\sigma, \text{hist}) : (\sigma, x, w) \notin R \text{ or } \langle \mathcal{P}(\sigma, x, w), \mathcal{V}(\sigma, x)\rangle = 1] = 1$$

*Computational soundness:*

$$\Pr[(\sigma, \text{hist}) \leftarrow \mathcal{G}(1^\lambda); x \leftarrow \mathcal{A}(\sigma, \text{hist}) : x \notin L_\sigma \text{ and } \langle \mathcal{A}, \mathcal{V}(\sigma, x)\rangle = 1] \approx 0$$

**Definition 2.** An argument $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is called *public coin* if the verifier chooses his messages uniformly at random and independently of the messages sent by the prover, i.e., the challenges correspond to the verifier's randomness $\rho$.

**Definition 3.** A public coin argument $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is called a *perfect special honest verifier zero knowledge (SHVZK)* argument for $R$ with common reference string generator $\mathcal{G}$ if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

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

To construct a fully zero-knowledge argument secure against *arbitrary* verifiers in the common reference string model one can first construct a SHVZK argument and then convert it into a fully zero-knowledge argument [Gro04, GMY06]. This conversion has constant additive overhead, so it is very efficient and allows us to focus on the simpler problem of getting SHVZK against honest verifiers.

To define an argument of knowledge we follow the approach of Groth and Ishai [GI08] and do it through witness-extended emulation first introduced by Lindell [Lin03]. This definition informally says that given an adversary that produces an acceptable argument with some probability, there exist an emulator that produces a similar argument with the same probability and at the same time provides a witness $w$.

**Definition 4.** A public coin argument $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ has witness extended emulation if for all deterministic polynomial time $\mathcal{P}^*$ there exists an expected polynomial time emulator $\mathcal{X}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} &\Pr[(\sigma, \text{hist}) \leftarrow \mathcal{G}(1^\lambda); (x, s) \leftarrow \mathcal{A}(\sigma, \text{hist}); tr \leftarrow \langle \mathcal{P}^*(\sigma, x, s), \mathcal{V}(\sigma, x)\rangle : \mathcal{A}(tr) = 1] \\ \approx &\Pr[(\sigma, \text{hist}) \leftarrow \mathcal{G}(1^\lambda); (x, s) \leftarrow \mathcal{A}(\sigma, \text{hist}); (tr, w) \leftarrow \mathcal{X}^{\langle \mathcal{P}^*(\sigma, x, s), \mathcal{V}(\sigma, x)\rangle}(\sigma, x, \rho) : \\ &\quad \mathcal{A}(tr) = 1 \text{ and if } tr \text{ is accepting then } (\sigma, x, w) \in R]. \end{aligned}$$

In the definition, $s$ can be interpreted as the state of $\mathcal{P}^*$, including the randomness. So whenever $\mathcal{P}^*$ is able to make a convincing argument when in state $s$, the emulator can extract a witness at the same time giving us an argument of knowledge. This definition automatically implies soundness.

## 2.5 The Fiat-Shamir heuristic

In the Fiat-Shamir heuristic the prover computes the public-coin challenges with a cryptographic hash-function instead of interacting with a verifier. This makes it possible for the prover to compute the entire argument without any interaction, i.e., it is a non-interactive argument.

Non-interactivity is desirable in many applications. Consider for instance an election, where the authorities want to convince independent verifiers that the tally is correct. Using non-interactive arguments the authorities only need to compute the argument once and then they can send the same non-interactive argument to all the verifiers. If the argument was interactive on the other hand, they would have to interact with each verifier, which would increase the complexity of the protocol.

The Fiat-Shamir heuristic is known to be secure in the random oracle model, where the cryptographic hash-function is modeled as a random oracle that returns a uniformly random answer to inputs it has not seen before. Those who are satisfied with a security proof in the random oracle model can therefore obtain a significant saving in interaction and in the prover's computation. For the sake of generality, we will describe the interactive protocols though.

### 2.6 The Schwartz-Zippel lemma

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

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

Given two multi-variate polynomials $p_1$ and $p_2$ we can test whether $p_1(x_1,\ldots,x_n)-p_2(x_1,\ldots,x_n)=0$ for random $x_1,\ldots,x_n \leftarrow \mathbb{Z}_q^*$ or not. This equation will always hold if $p_1=p_2$, whereas if $p_1 \neq p_2$ the probability that the test pass is only $\frac{\max(d_1,d_2)}{q-1}$.

## 3 Shuffle Argument

We will give an argument of knowledge of a permutation $\pi \in \Sigma_N$ and randomness $\{\rho_i\}_{i=1}^N$ such that for given ciphertexts $\{C_i\}_{i=1}^N, \{C'_i\}_{i=1}^N$ we have $C'_i = C_{\pi(i)}\mathcal{E}_{pk}(1;\rho_i)$. The shuffle argument combines a multi-exponentiation argument, which allows us to prove that the product of a set of ciphertexts raised to a set of committed exponents yields a particular ciphertext, and a product argument, which allows us to prove that a set of committed values has a particular product. The multi-exponentiation argument is given in Section 4 and the product argument is given in Section 5. In this section, we will give an overview of the protocol and explain how a multi-exponentiation argument can be combined with a product argument to yield an argument for the correctness of a shuffle.

The first step for the prover is to commit to the permutation. This is done by committing to $\pi(1),\ldots,\pi(N)$. The prover will now receive a challenge $x$ and commit to $x^{\pi(1)},\ldots,x^{\pi(N)}$. The prover will give an argument of knowledge of openings of the commitments to permutations of respectively $1,\ldots,N$ and $x^1,\ldots,x^N$ and demonstrate that the same permutation has been used in both cases. This means the prover has a commitment to $x^1,\ldots,x^N$ permuted in an order that was fixed before the prover saw $x$.

To check that the same permutation has been used in both commitments the verifier sends random challenges $y$ and $z$. By using the homomorphic properties of the commitment scheme the prover can in a verifiable manner compute commitments to $d_1 - z = y\pi(1)+x^{\pi(1)}-z,\ldots,d_N - z = y\pi(N)+x^{\pi(N)}-z$. Using the product argument from Section 5 the prover shows that $\prod_{i=1}^N(d_i - z) = \prod_{i=1}^N(yi + x^i - z)$. Observe that we have two identical degree $N$ polynomials in $z$ since the only difference is that the roots have been permuted. The verifier does not know a priori that the two polynomials are identical but can by the Schwartz-Zippel lemma deduce that the prover has negligible chance over the choice of $z$ of making a convincing argument unless indeed there is a permutation $\pi$ such that $d_1 = y\pi(1)+x^{\pi(1)},\ldots,d_N = y\pi(N)+x^{\pi(N)}$. Furthermore, there is negligible probability over the choice of $y$ of this being true unless the first commitment contains $\pi(1),\ldots,\pi(N)$ and the second commitment contains $x^{\pi(1)},\ldots,x^{\pi(N)}$.

The prover now has commitments to $x^{\pi(1)},\ldots,x^{\pi(N)}$ and uses the multi-exponentiation argument from Section 4 to demonstrate that there exists a $\rho$ such that $\prod_{i=1}^N C_i^{x^i} = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^N (C_i')^{x^{\pi(i)}}$. The verifier does not see the committed values and therefore does not learn what the permutation is. However, from the homomorphic properties of the encryption scheme the verifier can deduce $\prod_{i=1}^N M_i^{x^i} = \prod_{i=1}^N (M_i')^{x^{\pi(i)}}$ for some permutation $\pi$ that was chosen before the challenge $x$ was sent to the prover. Taking discrete logarithms we have the polynomial identity $\sum_{i=1}^N \log(M_i)x^i = \sum_{i=1}^N \log(M'_{\pi^{-1}(i)})x^i$. There is negligible probability over the choice of $x$ of this equality holding true unless $M_1' = M_{\pi(1)},\ldots,M_N' = M_{\pi(N)}$. This shows that we have a correct shuffle.

**Common reference string:** $pk, ck$.

**Statement:** $\vec{C},\vec{C'} \in \mathbb{H}^N$ with $N = mn$.

**Prover's witness:** $\pi \in \Sigma_N$ and $\vec{\rho} \in \mathbb{Z}_q^N$ such that $\vec{C'} = \mathcal{E}_{pk}(\vec{1};\vec{\rho})\vec{C}_\pi$.

**Initial message:** Pick $\vec{r} \leftarrow \mathbb{Z}_q^m$, set $\vec{a} = \{\pi(i)\}_{i=1}^N$ and compute $\vec{c}_A = \mathrm{com}_{ck}(\vec{a};\vec{r})$.

Send: $\vec{c}_A$

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

**Answer** Pick $\vec{s} \in \mathbb{Z}_q^m$, set $\vec{b} = \{x^{\pi(i)}\}_{i=1}^N$ and compute $\vec{c}_B = \mathrm{com}_{ck}(\vec{b};\vec{s})$.

Send: $\vec{c}_B$

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

**Answer:** Define $\vec{c}_{-z} = \mathrm{com}_{ck}(-z,\ldots,-z;\vec{0})$ and $\vec{c}_D = \vec{c}_A^{\,y}\vec{c}_B$. Compute $\vec{d} = y\vec{a} + \vec{b}$ and $\vec{t} = y\vec{r} + \vec{s}$. Engage in a product argument as described in Section 5 of openings $d_1 - z,\ldots,d_N - z$ and $\vec{t}$ such that

$$\vec{c}_D\vec{c}_{-z} = \mathrm{com}_{ck}(\vec{d} - \vec{z};\vec{t}) \qquad \text{and} \qquad \prod_{i=1}^N (d_i - z) = \prod_{i=1}^N (yi + x^i - z).$$

Compute $\rho = -\vec{\rho}\cdot\vec{b}$ and set $\vec{x} = (x, x^2,\ldots,x^N)^T$. Engage in a multi-exponentiation argument as described in Section 4 of $\vec{b}$, $\vec{s}$ and $\rho$ such that

$$\vec{C}^{\vec{x}} = \mathcal{E}_{pk}(1;\rho)\vec{C'}^{\vec{b}} \qquad \text{and} \qquad \vec{c}_B = \mathrm{com}_{ck}(\vec{b};\vec{s})$$

The two arguments can be run in parallel. Furthermore, the multi-exponentiation argument can be started already in round 3 after the computation of the commitments $\vec{c}_B$.

**Verification:** The verifier checks $\vec{c}_A, \vec{c}_B \in \mathbb{G}^m$ and computes $\vec{c}_{-z}, \vec{c}_D$ as described above and computes $\prod_{i=1}^N (yi + x^i - z)$ and $\vec{C}^{\vec{x}}$. The verifier accepts if the product argument and the multi-exponentiation argument both are valid.

**Theorem 5.** *The protocol is a public coin perfect SHVZK argument of knowledge of $\pi \in \Sigma_N$ and $\vec{\rho} \in \mathbb{Z}_q^N$ such that $\vec{C'} = \mathcal{E}_{pk}(\vec{1};\vec{\rho})\vec{C}_\pi$.*

*Proof.* Let us first argue that we have perfect completeness. Note that $d_i = y\pi(i) + x^{\pi(i)}$ so we have

$$\prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(y\pi(i) + x^{\pi(i)} - z) = \prod_{i=1}^{N}(yi + x^i - z).$$

Also, we have with $C'_i = \mathcal{E}_{pk}(1;\rho_i), b_i = x^{\pi(i)}$ and $\rho = -\vec{\rho}\cdot\vec{b}$ that

$$\mathcal{E}_{pk}(1;\rho)\vec{C'}^{\vec{b}} = \mathcal{E}_{pk}(\vec{1};-\vec{\rho})^{\vec{b}}\left(\mathcal{E}_{pk}(\vec{1},\vec{\rho})\vec{C}_\pi\right)^{\vec{b}} = \vec{C}_\pi^{\vec{b}} = \prod_{i=1}^{N} C_{\pi(i)}^{x^{\pi(i)}} = \vec{C}^{\vec{x}}.$$

Perfect completeness now follows from the perfect completeness of the product argument and the perfect completeness of the multi-exponentiation argument.

Perfect SHVZK follows from the fact that the commitments are perfectly hiding and the underlying arguments are perfect SHVZK. To simulate the entire argument we can pick random commitments $\vec{c}_A, \vec{c}_B \leftarrow \mathrm{com}_{ck}(0,\ldots,0)$ which can be done without knowing the witness for the correctness of the shuffle. The simulator then runs perfect SHVZK simulations of the product and multi-exponentiation arguments.

Finally, we have to show that we have witness-extended emulation. In the witness-extended emulation we run the prover and verifier $\langle\mathcal{P}^*,\mathcal{V}\rangle$ with random challenges $x, y, z \in \mathbb{Z}_q^*$ and random challenges for the product and multi-exponentiation arguments. If we get an acceptable argument we have to extract a witness. We start by rewinding and running the witness-extended emulators for the product and the multi-exponentiation arguments. This gives us openings of $\vec{c}_B$ and $\vec{c}_D$. Since $\vec{c}_D = \vec{c}_A^{\,y}\vec{c}_B$ this allows us to compute openings $\vec{a}, \vec{r}$ of $\vec{c}_A$.

We will now argue that with overwhelming probability the extracted openings of $\vec{c}_A$ must be of the form $\vec{a} = \{\pi(i)\}_{i=1}^N$ for some permutation $\pi \in \Sigma_N$. Consider the situation after round 3 where the prover has sent $\vec{c}_B$. The witness-extended emulation of the multi-exponentiation argument gives us the opening $\vec{b}, \vec{s}$ of $\vec{c}_B$ showing that the opening of $\vec{c}_D$ must be of the form $\vec{d} = y\vec{a} + \vec{b}$. The product argument shows that $\prod_{i=1}^N(d_i - z) = \prod_{i=1}^N(yi + x^i - z)$. This has negligible probability over $z$ in succeeding unless there exists a permutation $\pi$ such that $d_i = \pi(i) + x^{\pi(i)}$. This shows $ya_i + b_i = y\pi(i) + x^{\pi(i)}$, which has negligible probability over $y$ of being true unless $a_i = \pi(i)$. Furthermore, we then see that $b_i = x^{\pi(i)}$.

We now rewind and run the argument until we have extracted witnesses for the multi-exponentiation argument for $N$ random choices of $x$. Each choice $x_j$ gives us a witness containing $\rho^{(j)}$ such that

$$\vec{C}^{\vec{x}_j} = \mathcal{E}_{pk}(1;\rho^{(j)})\prod_{i=1}^{N}(C'_i)^{x_j^{\pi(i)}} = \mathcal{E}_{pk}(1;\rho^{(j)})\vec{C'}^{\vec{x}_j}_{\pi^{-1}}.$$

The $N \times N$ matrix

$$X = \begin{pmatrix} x_1^1 & \ldots & x_N^1 \\ \vdots & & \vdots \\ x_1^N & \ldots & x_N^N \end{pmatrix}$$

can be viewed as a submatrix of a transposed Vandermonde matrix. If $x_1, \ldots, x_N$ are different then $X$ is invertible. We now have

$$\vec{C} = (\vec{C}^X)^{X^{-1}} = \left(\mathcal{E}_{pk}(\vec{1};\vec{\rho})\vec{C'}^{X}_{\pi^{-1}}\right)^{X^{-1}} = \mathcal{E}_{pk}(\vec{1};\vec{\rho}X^{-1})\vec{C'}_{\pi^{-1}}.$$

This gives us a permutation $\pi$ and $\vec{\rho}\,' = (-\vec{\rho}X^{-1})_\pi$ such that $\vec{C'} = \mathcal{E}_{pk}(\vec{1};\vec{\rho}\,')\vec{C}_\pi$. $\square$

## 4 Multi-exponentiation Argument

Given ciphertexts $C_{11}, \ldots, C_{mn}$ and $C$ we will give in this section an argument of knowledge of openings of commitments $\vec{c}_A$ to $A = \{a_{ij}\}_{i,j=1}^{n,m}$ such that

$$C = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^{m} \vec{C}_i^{\,\vec{a}_i} \qquad \text{and} \qquad \vec{c}_A = \mathrm{com}_{ck}(A;\vec{r})$$

where $\vec{C}_i = (C_{i1}, \ldots, C_{in})$ and $\vec{a}_j = (a_{1j}, \ldots, a_{nj})^T$.

To explain the idea in the protocol let us for simplicity assume $\rho = 0$ and the prover knows the openings of $\vec{c}_A$, and leave the question of SHVZK for later. In other words, we will for now just explain how to convince the verifier in a communication-efficient manner that $C = \prod_{i=1}^{m} \vec{C}_i^{\,\vec{a}_i}$. The prover can calculate the ciphertexts

$$E_k = \prod_{\substack{1\le i,j\le m\\ j=(k-m)+i}} \vec{C}_i^{\,\vec{a}_j},$$

where $E_m = C$. To visualize this consider the following matrix

$$\begin{array}{cc}
 & \begin{pmatrix} \vec{a}_1 & \quad\cdots\quad & \vec{a}_m \end{pmatrix} \\
\begin{pmatrix} \vec{C}_1 \\ \vec{C}_2 \\ \vdots \\ \vec{C}_m \end{pmatrix} & \begin{pmatrix} \vec{C}_1^{\,\vec{a}_1} & & \ddots & \vec{C}_1^{\,\vec{a}_m} \\ \vec{C}_2^{\,\vec{a}_1} & & \ddots & \vec{C}_2^{\,\vec{a}_m} \\ \ddots & \ddots & \ddots & \ddots \\ \vec{C}_m^{\,\vec{a}_1} & & \ddots & \vec{C}_m^{\,\vec{a}_m} \end{pmatrix} \begin{matrix} \\ E_{2m-1} \\ \vdots \\ E_{m+1} \end{matrix} \\
 & \begin{matrix} E_1 & \quad\cdots\quad & E_{m-1} & \quad E_m \end{matrix}
\end{array}$$

The prover sends the ciphertexts $E_1, \ldots, E_{2m-1}$ to the verifier. The ciphertext $C = E_m$ is the product of the main diagonal and the other $E_k$'s are the products of the other diagonals. The prover will use a batch-proof to simultaneously convince the verifier that all the diagonal products give their corresponding $E_k$.

The verifier selects a challenge $x \leftarrow \mathbb{Z}_q^*$ at random. The prover sets $\vec{x} = (x, x^2, \ldots, x^m)^T$ opens $\vec{c}_A^{\,\vec{x}}$ to $\vec{a} = \sum_{j=1}^{m} x^j \vec{a}_j$ and the verifier checks

$$C^{x^m} \prod_{\substack{k=1\\ k\neq m}}^{2m-1} E_k^{x^k} = \prod_{i=1}^{m} \vec{C}_i^{\,(x^{m-i}\vec{a})}.$$

Since $x$ is chosen at random, the prover has negligible probability of convincing the verifier unless the $x^k$-related terms match on each side of the equality for all $k$. In particular, since $\vec{a} = \sum_{j=1}^{m} x^j \vec{a}_j$ the $x^m$-related terms give us

$$C^{x^m} = \prod_{i=1}^{m} \vec{C}_i^{\;x^{m-i}\sum_{\substack{1\le j\le m\\ m=m-i+j}} x^j \vec{a}_j} = \left(\prod_{i=1}^{m} \vec{C}_i^{\,\vec{a}_i}\right)^{x^m}$$

and allow the verifier to conclude $C = \prod_{i=1}^{m} \vec{C}_i^{\,\vec{a}_i}$.

Finally, to make the argument honest verifier zero-knowledge we have to avoid leaking information about the exponent vectors $\vec{a}_1, \ldots, \vec{a}_m$. The prover therefore commits to a random vector $\vec{a}_0 \leftarrow \mathbb{Z}_q^n$ and after she sees the challenge $x$ she reveals $\vec{a} = \vec{a}_0 + \sum_{j=1}^{m} x^j \vec{a}_j$. Since $\vec{a}_0$ is chosen at random this vector does not leak any information about the exponents.

Another possible source of leakage is the products of the diagonals. The prover will therefore randomize each $E_k$ by multiplying it with a random ciphertext $\mathcal{E}_{pk}(G^{b_k};\tau_k)$. Now each $E_k$ is a uniformly random group element in $\mathbb{H}$ and will therefore not leak information about the exponents. Of course, this would make it possible to encrypt anything in the $E_k$ and allow cheating. To get around this problem the prover has to commit to the $b_k$'s used in the random encryptions and the verifier will check that the prover uses $b_m = 0$. The full argument that also covers the case $\rho \neq 0$ can be found below.

**Common reference string:** $pk, ck$.

**Statement:** $\vec{C}_1, \ldots, \vec{C}_m \in \mathbb{H}^n$ , $C \in \mathbb{H}$ and $\vec{c}_A \in \mathbb{G}^m$

**Prover's witness:** $A = \{\vec{a}_j\}_{j=1}^m \in \mathbb{Z}_q^{n\times m}$, $\vec{r} \in \mathbb{Z}_q^m$ and $\rho \in \mathbb{Z}_q$ such that

$$C = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^{m} \vec{C}_i^{\,\vec{a}_i} \qquad \text{and} \qquad \vec{c}_A = \mathrm{com}_{ck}(A;\vec{r})$$

**Initial message:** Pick $\vec{a}_0 \leftarrow \mathbb{Z}_q^n, r_0 \leftarrow \mathbb{Z}_q$ and $b_0, s_0, \tau_0 \ldots, b_{2m-1}, s_{2m-1}, \tau_{2m-1} \leftarrow \mathbb{Z}_q$ and set $b_m = 0, s_m = 0, \tau_m = \rho$. Compute for $k = 0, \ldots, 2m-1$

$$c_{A_0} = \mathrm{com}_{ck}(\vec{a}_0; r_0) \qquad c_{B_k} = \mathrm{com}_{ck}(b_k; s_k) \qquad E_k = \mathcal{E}_{pk}(G^{b_k};\tau_k) \prod_{\substack{i=1,j=0\\ j=(k-m)+i}}^{m,m} \vec{C}_i^{\,\vec{a}_j},$$

Send: $c_{A_0}, \{c_{B_k}\}_{k=0}^{2m-1}, \{E_k\}_{k=0}^{2m-1}$.

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

**Answer:** Set $\vec{x} = (x, x^2, \ldots, x^m)^T$ and compute

$$\vec{a} = \vec{a}_0 + A\vec{x} \qquad r = r_0 + \vec{r}\cdot\vec{x} \qquad b = b_0 + \sum_{k=1}^{2m-1} b_k x^k \qquad s = s_0 + \sum_{k=1}^{2m-1} s_k x^k \qquad \tau = \tau_0 + \sum_{k=1}^{2m-1} \tau_k x^k$$

Send: $\vec{a}, r, b, s, \tau$.

**Verification:** Check $c_{A_0}, c_{B_0}, \ldots, c_{B_{2m-1}} \in \mathbb{G}$ and $E_0, \ldots, E_{2m-1} \in \mathbb{H}$ and $\vec{a} \in \mathbb{Z}_q^n$ and $r, b, s, \tau \in \mathbb{Z}_q$ and accept if $c_{B_m} = \mathrm{com}_{ck}(0;0)$ and $E_m = C$ and

$$c_{A_0}\vec{c}_A^{\,\vec{x}} = \mathrm{com}_{ck}(\vec{a};r) \qquad c_{B_0}\prod_{k=1}^{2m-1} c_{B_k}^{x^k} = \mathrm{com}_{ck}(b;s) \qquad E_0 \prod_{k=1}^{2m-1} E_k^{x^k} = \mathcal{E}_{pk}(G^b;\tau)\prod_{i=1}^{m} \vec{C}_i^{\,x^{m-i}\vec{a}}.$$

**Theorem 6.** *The protocol above is a public coin perfect SHVZK argument of knowledge of openings $\vec{a}_1, \ldots, \vec{a}_m, \vec{r}$ and randomness $\rho$ such that $C = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^m \vec{C}_i^{\,\vec{a}_i}$.*

*Proof.* It follows by direct verification that $E_m = C$ and $c_{A_0}\vec{c}_A^{\,\vec{x}} = \mathrm{com}_{ck}(\vec{a};r)$ and $c_{B_0}\prod_{k=1}^{2m-1} c_{B_k}^{x^k} =$

$\mathrm{com}_{ck}(b;s)$. Perfect completeness now follows from

$$\begin{aligned}
E_0 \prod_{k=1}^{2m-1} E_k^{x^k} &= \prod_{k=0}^{2m-1} \left( \mathcal{E}_{pk}(G^{b_k};\tau_k) \prod_{\substack{i=1,j=0\\ j=(k-m)+i}}^{m,m} \vec{C}_i^{\,\vec{a}_j} \right)^{x^k} \\
&= \mathcal{E}_{pk}\big(G^{\sum_{k=0}^{2m-1} b_k x^k}; \sum_{k=0}^{2m-1} \tau_k x^k\big) \prod_{k=0}^{2m-1} \prod_{\substack{i=1,j=0\\ j=(k-m)+i}}^{m,m} \vec{C}_i^{\,x^k \vec{a}_j} \\
&= \mathcal{E}_{pk}(G^b;\tau) \prod_{i=1}^{m} \prod_{j=0}^{m} \vec{C}_i^{\,x^{m-i+j}\vec{a}_j} \\
&= \mathcal{E}_{pk}(G^b;\tau) \prod_{i=1}^{m} \vec{C}_i^{\,x^{m-i}\sum_{j=0}^{m} x^j \vec{a}_j} = \mathcal{E}_{pk}(G^b;\tau) \prod_{i=1}^{m} \vec{C}_i^{\,x^{m-i}\vec{a}}.
\end{aligned}$$

Now we will prove that we have perfect SHVZK. On challenge $x$ the simulator picks $\vec{a} \leftarrow \mathbb{Z}_q^n$ and $r \leftarrow \mathbb{Z}_q$ at random and sets $c_{A_0} = \mathrm{com}_{ck}(\vec{a};r) \cdot \vec{c}_A^{\,-\vec{x}}$. The simulator also picks $b, s, s_1, \ldots, s_{m-1}, s_{m+1}, \ldots, s_{2m-1} \leftarrow \mathbb{Z}_q$ and defines $s_m = 0$. It computes commitments $\vec{c}_{B_1}, \ldots, \vec{c}_{B_{2m-1}}$ as $c_{B_k} = \mathrm{com}_{ck}(0;s_k)$ and $c_{B_0} = \mathrm{com}_{ck}(b;s) \cdot \prod_{k=1}^{2m-1} c_{B_k}^{-x^k}$. Finally it picks random ciphertexts $E_1, \ldots, E_{m-1}, E_{m+1}, \ldots, E_{2m-1} \leftarrow \mathbb{H}$ and $\tau \leftarrow \mathbb{Z}_q$, sets $E_m = C$ and computes

$$E_0 = \mathcal{E}_{pk}(G^b;\tau) \cdot \prod_{i=1}^{m} \vec{C}_i^{\,(x^{m-i}\vec{a})} \cdot \prod_{k=1}^{2m-1} E_k^{-x^k}.$$

The simulated argument is $c_{A_0}, \{c_{B_k}\}_{k=0}^{2m-1}, \{E_k\}_{k=0}^{2m-1}, x, \vec{a}, r, b, s, \tau$.

The next step is to prove that the simulation on challenge $x$ is indistinguishable from a real argument with challenge $x$. The commitment scheme is perfectly hiding so the distribution of the commitments $c_{B_1}, \ldots, c_{B_{m-1}}, c_{B_{m+1}}, \ldots, c_{B_{2m-1}}$ is identical to the distribution we get in a real argument and $c_{B_m} = \mathrm{com}_{ck}(0;0)$ as in a real argument. The ciphertexts $E_1, \ldots, E_{m-1}, E_{m+1}, \ldots, E_{2m-1}$ are uniformly random as in a real argument and $E_m = C$ as in a real argument. In the real argument the values $\vec{a}_0, r_0, b_0, s_0$ and $\tau_0$ are picked at random giving us that $\vec{a}, r, b, s, \tau$ are uniformly random just as in the simulation. So, up to this point we have the same probability distribution in both real arguments and simulated arguments and the remaining parts $c_{A_0}, c_{B_0}, E_0$ are now uniquely defined by the verification equations. It follows that real arguments and simulated arguments have identical probability distributions.

Finally, we will show that the argument has witness-extended emulation. The witness-extended emulator runs $\langle \mathcal{P}^*, \mathcal{V} \rangle$ to get a transcript. If the verifier rejects the transcript it does not need to do anything else, but if the verifier accepts the witness-extended emulator will attempt to extract a witness. It rewinds the argument to the challenge phase and runs it with fresh challenges until it has $2m$ acceptable arguments.

The witness-extended emulator runs in expected polynomial time. If the witness-extended emulator after the initial message has probability $\epsilon > 0$ of answering a random challenge, then there is probability $\epsilon$ that the witness-extended emulator will have to sample $2m$ accepting transcripts. The witness-extended emulator is expected to rewind $\frac{2m}{\epsilon}$ times to sample the transcripts when the initial argument was accepting. This gives an overall expectation of $2m$ rewinds whenever there is non-zero probability for $\mathcal{P}^*$ to answer the challenge after having sent the initial message.

There is negligible probability that an expected polynomial time emulator would get $2m$ transcripts with a collision among the challenges. We will now analyze the case where it has $2m$ accepting transcripts with

$2m$ different challenges $x_1, \dots, x_{2m}$. The first $m+1$ transcripts give us a transposed Vandermonde matrix

$$X = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_{m+1} \\ \vdots & \vdots & & \vdots \\ x_1^m & x_2^m & \dots & x_{m+1}^m \end{pmatrix}.$$

Since $x_1, \dots, x_{m+1}$ are different $X$ is invertible.

We now have for each $x_\ell$ an opening of $\vec{c}_{A_0}\vec{c}_A^{\,\vec{x}_\ell} = \mathrm{com}_{ck}(\vec{a}^{(\ell)}, r^{(\ell)})$. Let $A_x$ be the matrix with columns $\vec{a}^{(1)} \ \dots \ \vec{a}^{\,(m+1)}$ and let $\vec{r}_x = (r^{(1)}, \dots, r^{(m+1)})$. We then have

$$(c_{A_0}, \dots, c_{A_m}) = ((c_{A_0}, \dots, c_{A_m})^X)^{X^{-1}} = \mathrm{com}_{ck}(A_x; \vec{r}_x)^{X^{-1}} = \mathrm{com}_{ck}(A_x X^{-1}; \vec{r}_x X^{-1}),$$

giving us openings $\vec{a}_0, r_0, \dots, \vec{a}_m, r_m$ of the commitments $c_{A_0}, \dots, c_{A_m}$. In a similar way the emulator can compute openings $b_0, s_0, \dots, b_{2m-1}, s_{2m-1}$ of $c_{B_0}, \dots, c_{B_{2m-1}}$. We now have for $\ell = 1, \dots, 2m$ that $\vec{a}^{(\ell)} = \sum_{j=0}^m x_\ell^j \vec{a}_j$, this means

$$\prod_{i=1}^m \vec{C}_i^{\,x_\ell^{m-i}\vec{a}^{(\ell)}} = \prod_{i=1}^m \vec{C}_i^{\,\sum_{j=0}^m x_\ell^{m+j-i}\vec{a}_j} = \prod_{k=0}^{2m-1} \left( \prod_{\substack{j=0 \\ 1 \le m-k+j \le m}}^m \vec{C}_{(m-k)+j}^{\,\vec{a}_j} \right)^{x_\ell^k}.$$

Let $\vec{E} = (E_0, \dots, E_{2m-1})$ and let $Y$ be the inverse of the $2m \times 2m$ matrix

$$X = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_1^{2m-1} & x_2^{2m-1} & \dots & x_{2m}^{2m-1} \end{pmatrix}$$

and let $\vec{y}_i$ be the $i$th column vector in $Y$ (numbered 0 through $2m-1$). We get for each $i = 0, \dots, 2m-1$

$$\begin{aligned} E_i = \vec{E}^{X\vec{y}_i} &= \prod_{\ell=1}^{2m} \left( \prod_{k=0}^{2m-1} E_k^{x_\ell^k} \right)^{y_{\ell i}} \\ &= \prod_{\ell=1}^{2m} \left( \mathcal{E}_{pk}\Big(G^{b^{(\ell)}}; \tau^{(\ell)}\Big) \prod_{k=0}^{2m-1} \Big( \prod_{\substack{j=0 \\ 1 \le m-k+j \le m}}^m \vec{C}_{(m-k)+j}^{\,\vec{a}_j} \Big)^{x_\ell^k} \right)^{y_{\ell i}} \\ &= \mathcal{E}_{pk}\Big(G^{\sum_{\ell=1}^{2m} b^{(\ell)} y_{\ell i}}; \sum_{\ell=1}^{2m} \tau^{(\ell)} y_{\ell i}\Big) \prod_{k=0}^{2m-1} \left( \prod_{\substack{j=0 \\ 1 \le m-k+j \le m}}^m \vec{C}_{(m-k)+j}^{\,\vec{a}_j} \right)^{\sum_{\ell=1}^{2m} x_\ell^k y_{\ell i}} \\ &= \mathcal{E}_{pk}\big(G^{b_i}; \tau_i\big) \prod_{\substack{j=0 \\ 1 \le m-i+j \le m}}^m \vec{C}_{(m-i)+j}^{\,\vec{a}_j}, \end{aligned}$$

where $\tau_i = \sum_{\ell=1}^{2m} \tau^{(\ell)} y_{\ell i}$.

The verifier checks $c_{B_m} = \mathrm{com}_{ck}(0;0)$ so the binding property implies that there is negligible probability for the emulator extracting $b_m \neq 0$. Since $E_m = C$ we now have

$$C = \mathcal{E}_{pk}(G^0;\tau_m) \prod_{\substack{j=0\\1\le m-m+j\le m}}^{m} \vec{C}_{m-m+j}^{\,\vec{a}_j} = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^{m}\vec{C}_i^{\,\vec{a}_i}$$

with $\rho = \tau_m$. This shows that we have extracted a valid witness for the statement. □

## 4.1 The prover's computation

The prover has to compute

$$E_0,\ldots,E_{2m-1}.$$

In this section we will for clarity ignore the randomization needed to get honest verifier zero-knowledge, which can be added in a straightforward manner at little extra computational cost. So let us say we need to compute for $k = 1,\ldots,2m-1$ the elements

$$E_k = \prod_{\substack{i=1,j=1\\j=(k-m)+i}}^{m,m} \vec{C}_i^{\,\vec{a}_j}.$$

This can be done by first computing the $m^2$ products $\vec{C}_i^{\,\vec{a}_j}$ and then computing the $E_k$'s as suitable products of some of these values. Since each product $\vec{C}_i^{\,\vec{a}_j}$ is of the form $\prod_{\ell=1}^{n} C_{i\ell}^{a_{j\ell}}$ this gives a total of $m^2 n$ exponentiations in $\mathbb{H}$. For large $m$ this cost is prohibitive.

It turns out that we can do much better by using techniques inspired by multiplication of integers and polynomials, such as Karatsuba [KO63], Toom-Cook [Too00, Coo66] and using the Fast Fourier Transform [CT65]. A common theme in these techniques is to compute the coefficients of the product $p(x)q(x)$ of two degree $m-1$ polynomials $p(x)$ and $q(x)$ by evaluating $p(x)q(x)$ in $2m-1$ points $\omega_0,\ldots,\omega_{2m-2}$ and using polynomial interpolation to recover the coefficients of $p(x)q(x)$ from $p(\omega_0)q(\omega_0),\ldots,p(\omega_{2m-2})q(\omega_{2m-2})$.

If we pick $\omega \in \mathbb{Z}_q$ we can evaluate the vectors

$$\prod_{i=1}^{m}\vec{C}_i^{\,\omega^{m-i}} \qquad\qquad \sum_{j=1}^{m}\omega^{j-1}\vec{a}_j.$$

This gives us

$$\left(\prod_{i=1}^{m}\vec{C}_i^{\,\omega^{m-i}}\right)^{\sum_{j=1}^{m}\omega^{j-1}\vec{a}_j} = \prod_{k=1}^{2m-1}\left(\prod_{\substack{i=1,j=1\\j=(k-m)+i}}^{m,m}\vec{C}_i^{\,\vec{a}_j}\right)^{\omega^{k-1}} = \prod_{k=1}^{2m-1}E_k^{\omega^{k-1}}.$$

Picking $2m-1$ different $\omega_0,\ldots,\omega_{2m-2}\in\mathbb{Z}_p$ we get the $2m-1$ ciphertexts

$$\prod_{k=1}^{2m-1}E_k^{\omega_0^{k-1}},\ \ldots,\ \prod_{k=1}^{2m-1}E_k^{\omega_{2m-2}^{k-1}}.$$

The $\omega_0,\ldots,\omega_{2m-2}$ are different and therefore the transposed Vandermonde matrix

$$\begin{pmatrix} 1 & \ldots & 1\\ \vdots & & \vdots\\ \omega_0^{2m-2} & \ldots & \omega_{2m-2}^{2m-2}\end{pmatrix}$$

is invertible. Let $\vec{y}_i = (y_0, \ldots, y_{2m-2})^T$ be the $i$th column of the inverse matrix. We can now compute $E_i$ as

$$E_i = \prod_{\ell=0}^{2m-2} \left( \prod_{k=1}^{2m-1} E_k^{\omega_\ell^{k-1}} \right)^{y_\ell} = \prod_{\ell=0}^{2m-2} \left( \left( \prod_{i=1}^{m} \vec{C}_i^{\,\omega_\ell^{m-i}} \right)^{\sum_{j=1}^{m} \omega_\ell^{j-1} \vec{a}_j} \right)^{y_\ell}$$

This means the prover can compute $E_1, \ldots, E_{2m-1}$ as linear combinations of

$$\left( \prod_{i=1}^{m} \vec{C}_i^{\,\omega_0^{m-i}} \right)^{\sum_{j=1}^{m} \omega_0^{j-1} \vec{a}_j} \qquad \cdots \qquad \left( \prod_{i=1}^{m} \vec{C}_i^{\,\omega_{2m-2}^{m-i}} \right)^{\sum_{j=1}^{m} \omega_{2m-2}^{j-1} \vec{a}_j}.$$

The expensive step in this computation is to compute $\prod_{i=1}^{m} \vec{C}_i^{\,\omega_0^{m-i}}, \ldots, \prod_{i=1}^{m} \vec{C}_i^{\,\omega_{2m-2}^{m-i}}$.

If $2m-2$ is a power of 2 and $2m-2 | q-1$ we can pick $\omega_1, \ldots, \omega_{2m-2}$ as roots of unity, i.e., $\omega_k^{2m-2} = 1$. This allows us to use the Fast Fourier Transformation "in the exponent" to simultaneously calculate $\prod_{i=1}^{m} \vec{C}_i^{\,\omega_k^{m-i}}$ in all of the roots of unity using only $O(mn \log m)$ exponentiations. This is asymptotically the fastest technique we know for computing $E_0, \ldots, E_{2m-2}$.

Unfortunately, the FFT is not well suited for being used in combination with multi-exponentiation techniques and in practice it therefore takes a while before the asymptotic behavior kicks in. For small $m$ it is therefore useful to consider other strategies. Inspired by the Toom-Cook method for integer multiplication, we may for instance choose $\omega_0, \omega_1, \ldots, \omega_{2m-2}$ to be small integers. When $m$ is small even the largest exponent $\omega_k^{2m-2}$ will remain small. For instance, if $m = 4$ we may choose $\omega_k \in \{0, -1, 1, -2, 2, -3, 3\}$, which makes the largest exponent $\omega_k^{m-1} = 3^3 = 27$. This makes it cheap to compute each $\prod_{i=1}^{m} \vec{C}_i^{\,\omega_k^{m-i}}$ because the exponents are very small.

The basic step of Toom-Cook sketched above can be optimized by choosing the evaluation points carefully. However, the performance degrades quickly as $m$ grows. Using recursion it is possible to get subquadratic complexity also for large $m$, however, the cost still grows relatively fast. In the next section we will therefore describe an interactive technique for reducing the prover's computation. In our implementation, see Section 6, we have used a combination of the interactive technique and Toom-Cook as the two techniques work well together.

## 4.2 Trading computation for interaction

We will present an interactive technique that can be used to reduce the prover's computation. The prover wants to show that $C$ has the same plaintext as the product of the main diagonal of following matrix (here illustrated for $m = 16$).

$$\begin{pmatrix}
\vec{C}_1^{\vec{a}_1} & \vec{C}_1^{\vec{a}_2} & \vec{C}_1^{\vec{a}_3} & \vec{C}_1^{\vec{a}_4} & & & & \\
\vec{C}_2^{\vec{a}_1} & \vec{C}_2^{\vec{a}_2} & \vec{C}_2^{\vec{a}_3} & \vec{C}_2^{\vec{a}_4} & & \ddots & & \\
\vec{C}_3^{\vec{a}_1} & \vec{C}_3^{\vec{a}_2} & \vec{C}_3^{\vec{a}_3} & \vec{C}_3^{\vec{a}_4} & & & & \\
\vec{C}_4^{\vec{a}_1} & \vec{C}_4^{\vec{a}_2} & \vec{C}_4^{\vec{a}_3} & \vec{C}_4^{\vec{a}_4} & & & & \\
& & & \ddots & & & & \\
& & & & \vec{C}_{13}^{\vec{a}_{13}} & \vec{C}_{13}^{\vec{a}_{14}} & \vec{C}_{13}^{\vec{a}_{15}} & \vec{C}_{13}^{\vec{a}_{16}} \\
& \ddots & & & \vec{C}_{14}^{\vec{a}_{13}} & \vec{C}_{14}^{\vec{a}_{14}} & \vec{C}_{14}^{\vec{a}_{15}} & \vec{C}_{14}^{\vec{a}_{16}} \\
& & & & \vec{C}_{15}^{\vec{a}_{13}} & \vec{C}_{15}^{\vec{a}_{14}} & \vec{C}_{15}^{\vec{a}_{15}} & \vec{C}_{15}^{\vec{a}_{16}} \\
& & & & \vec{C}_{16}^{\vec{a}_{13}} & \vec{C}_{16}^{\vec{a}_{14}} & \vec{C}_{16}^{\vec{a}_{15}} & \vec{C}_{16}^{\vec{a}_{16}}
\end{pmatrix}$$

In the previous section the prover calculated all $m^2$ entries of the matrix. But we are only interested in the product along the diagonal so we can save computation by just focusing on the blocks close to the main diagonal.

Let us explain the idea in the case of $m = 16$. We can divide the matrix into $4 \times 4$ blocks and only use the four blocks that are on the main diagonal. Suppose the prover wants to demonstrate $C = \prod_{i=1}^{16} \vec{C}_i^{\vec{a}_i}$. Let us for now just focus on soundness and return to the question of honest verifier zero-knowledge later. The prover starts by sending $E_0, E_1, E_2, E_3, E_4, E_5, E_6$ that are the products along the diagonals of the elements in the blocks that we are interested in. I.e., $E_0 = \prod_{i=1}^{4} \vec{C}_{4i}^{\vec{a}_{4i-3}}, \ldots, E_6 = \prod_{i=1}^{4} \vec{C}_{4i-3}^{\vec{a}_{4i}}$ and $E_3 = C$. The verifier sends a random challenge $x$ and using the homomorphic properties of the encryption scheme and of the commitment scheme both the prover and the verifier can compute $\vec{C}'_1, \ldots, \vec{C}'_4$ and $c_{A'_1}, \ldots, c_{A'_4}$ as

$$\vec{C}'_i = \vec{C}_{4i-3}^{x^3} \vec{C}_{4i-2}^{x^2} \vec{C}_{4i-1}^{x} \vec{C}_{4i} \qquad\qquad c_{A'_j} = c_{A_{4j-3}} c_{A_{4j-2}}^{x} c_{A_{4j-1}}^{x^2} c_{A_{4j}}^{x^3}.$$

They can also both compute $C' = \prod_{k=0}^{6} E_k^{x^k}$. The prover and the verifier now engage in an SHVZK argument for the smaller statement $C' = \prod_{i=1}^{4} \vec{C}'_i{}^{\vec{a}'_i}$. The prover can compute a witness for this statement with $\vec{a}'_i = \vec{a}_{4i-3} + x\vec{a}_{4i-2} + x^2\vec{a}_{4i-1} + x^3\vec{a}_{4i}$. This shows

$$C^{x^3} \prod_{\substack{k=0 \\ k\neq 3}}^{6} E_k^{x^k} = \prod_{i=1}^{4} (\vec{C}_{4i-3}^{x^3} \vec{C}_{4i-2}^{x^2} \vec{C}_{4i-1}^{x} \vec{C}_{4i})^{(\vec{a}_{4i-3} + x\vec{a}_{4i-2} + x^2\vec{a}_{4i-1} + x^3\vec{a}_{4i})}.$$

Looking at the $x^3$-related terms, we see this has negligible chance of holding for a random $x$ unless $C = \prod_{i=1}^{16} \vec{C}_i^{\vec{a}_i}$, which is what the prover wanted to demonstrate.

We will generalize the technique to reducing a statement $\vec{C}_1, \ldots, \vec{C}_m, C, c_{A_1}, \ldots, c_{A_m}$ with a factor $\mu$ to a statement $\vec{C}'_1, \ldots, \vec{C}'_{m'}, C', c_{A'_1}, \ldots, c_{A'_{m'}}$ where $m = \mu m'$. To add honest verifier zero-knowledge to the protocol, we have to prevent the $E_k$'s from leaking information about $\vec{a}_1, \ldots, \vec{a}_m$. We do this by randomizing each $E_k$ with a random ciphertext $\mathcal{E}_{pk}(G^{b_k}; t_k)$. To prevent the prover to use the randomization to cheat she will have to commit the $b_k$'s before seeing the challenge $x$.

**Common Reference string:** $pk, ck$.

**Statement:** $\vec{C}_1, \ldots, \vec{C}_m \in \mathbb{H}^n$ and $C \in \mathbb{H}$ and $c_{A_1}, \ldots, c_{A_m} \in \mathbb{G}$ where $m = \mu m'$.

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

$$C = \mathcal{E}_{pk}(1; \rho) \prod_{i=1}^{m} \vec{C}_i^{\vec{a}_i} \qquad \text{and} \qquad \vec{c}_A = \mathrm{com}_{ck}(A; \vec{r}).$$

**Initial message:** Pick $\vec{b} = (b_0, \ldots, b_{2\mu-2}), \vec{s}, \vec{\tau} \leftarrow \mathbb{Z}_q^{2\mu-1}$ and set $b_{\mu-1} = 0, s_{\mu-1} = 0, \tau_{\mu-1} = \rho$. Compute for $k = 0, \ldots, 2\mu - 2$

$$c_{b_k} = \mathrm{com}_{ck}(b_k; s_k) \qquad\qquad E_k = \mathcal{E}_{pk}(G^{b_k}; \tau_k) \prod_{\ell=0}^{m'-1} \prod_{\substack{i=1, j=1 \\ j=(k+1-\mu)+i}}^{\mu,\mu} \vec{C}_{\mu\ell+i}^{\vec{a}_{\mu\ell+j}}.$$

Send: $\vec{c}_b = (c_{b_0}, \ldots, c_{b_{2\mu-2}})$ and $\vec{E} = (E_0, \ldots, E_{2\mu-2})$.

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

**Answer:** Set $\vec{x} = (1, x, \dots, x^{2\mu-2})^T$ and send $b = \vec{b} \cdot \vec{x}$ and $s = \vec{s} \cdot \vec{x}$ to the verifier.

Compute for $\ell = 1, \dots, m'$

$$\vec{a}'_\ell = \sum_{j=1}^{\mu} x^{j-1} \vec{a}_{\mu(\ell-1)+j} \qquad r'_\ell = \sum_{j=1}^{\mu} x^{j-1} r_{\mu(\ell-1)+j} \qquad \rho' = \vec{\tau} \cdot \vec{x}.$$

Define $\vec{C}'_1, \dots, \vec{C}'_{m'}$ and $c_{A'_1}, \dots, c_{A'_{m'}}$ and $C'$ by

$$\vec{C}'_\ell = \prod_{i=1}^{\mu} \vec{C}_{\mu(\ell-1)+i}^{\,x^{\mu-i}} \qquad c_{A'_\ell} = \prod_{j=1}^{\mu} c_{A_{\mu(\ell-1)+j}}^{x^{j-1}} \qquad C' = \mathcal{E}_{pk}(G^{-b}; 0)\vec{E}^{\vec{x}}.$$

Engage in an SHVZK argument of openings $\vec{a}'_1, \dots, \vec{a}'_{m'}, \vec{r}'$ and $\rho'$ such that $C' = \mathcal{E}_{pk}(1; \rho') \prod_{\ell=1}^{m'} \vec{C}'^{\,\vec{a}'_\ell}_\ell$.

**Verification:** Check $\vec{c}_b \in \mathbb{G}^{2\mu-1}$ and $E_0, \dots, E_{2\mu-2} \in \mathbb{H}$ and $b, s \in \mathbb{Z}_q$. Accept if

$$c_{b_{\mu-1}} = \mathrm{com}_{ck}(0; 0) \qquad E_{\mu-1} = C \qquad \vec{c}_b^{\,\vec{x}} = \mathrm{com}_{ck}(b; s)$$

and if the SHVZK argument for $\vec{C}'_1, \dots, \vec{C}'_{m'}, C', c_{A'_1}, \dots, c_{A'_{m'}}$ is valid.

**Theorem 7.** *The protocol above is a public coin perfect SHVZK argument of knowledge of $\vec{a}_1, \dots, \vec{a}_m$, $\vec{r}$ such that $C = \mathcal{E}_{pk}(1; \rho) \prod_{i=1}^{m} \vec{C}_i^{\,\vec{a}_i}$*

*Proof.* Let us first argue that we have perfect completeness. It follows by straightforward verification that $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0; 0)$ and $C = E_{\mu-1}$ and $\vec{c}_b^{\,\vec{x}} = \mathrm{com}_{ck}(b; s)$. Both the prover and the verifier can compute the reduced statement $\vec{C}'_1, \dots, \vec{C}'_{m'}, C', \vec{c}_{A'}$ as described above and the prover can compute openings of $\vec{c}_{A'}$ and $\rho' = \vec{\tau} \cdot \vec{x}$. Perfect completeness now follows from the perfect completeness of the underlying SHVZK argument because $C' = \mathcal{E}_{pk}(1; \rho') \prod_{i=1}^{m'} \vec{C}'^{\,\vec{a}'_i}_i$. To see this is the case, we compute

$$\begin{aligned}
C' &= \mathcal{E}_{pk}(G^{-b}; 0) \prod_{k=0}^{2\mu-2} E_k^{x^k} \\
&= \mathcal{E}_{pk}(G^{-\vec{b}\cdot\vec{x}}; 0) \prod_{k=0}^{2\mu-2} \left( \mathcal{E}_{pk}(G^{b_k}; \tau_k) \prod_{\ell=0}^{m'-1} \prod_{\substack{i=1,j=1 \\ j=(k+1-\mu)+i}}^{\mu,\mu} \vec{C}_{\mu\ell+i}^{\,\vec{a}_{\mu\ell+j}} \right)^{x^k} \\
&= \mathcal{E}_{pk}\big(G^0; \vec{\tau} \cdot \vec{x}\big) \prod_{k=0}^{2\mu-2} \prod_{\ell=0}^{m'-1} \prod_{\substack{i=1,j=1 \\ j=(k+1-\mu)+i}}^{\mu,\mu} \big(\vec{C}_{\mu\ell+i}^{\,x^{\mu-i}}\big)^{x^{j-1}\vec{a}_{\mu\ell+j}} \\
&= \mathcal{E}_{pk}(1; \rho') \prod_{\ell=1}^{m'} \prod_{k=0}^{2\mu-2} \prod_{\substack{i=1,j=1 \\ j=(k+1-\mu)+i}}^{\mu,\mu} \big(\vec{C}_{\mu(\ell-1)+i}^{\,x^{\mu-i}}\big)^{x^{j-1}\vec{a}_{\mu(\ell-1)+j}} \\
&= \mathcal{E}_{pk}(1; \rho') \prod_{\ell=1}^{m'} \vec{C}'^{\,\vec{a}'_\ell}_\ell
\end{aligned}$$

Let us now describe the SHVZK simulator. On challenge $x$ it picks $b, s \leftarrow \mathbb{Z}_q$ and $c_{b_1}, \ldots, c_{b_{2\mu-2}} \leftarrow \mathbb{G}$ and $E_0, \ldots, E_{2\mu-2} \leftarrow \mathbb{H}$. It sets $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0;0)$ and $E_{\mu-1} = C$ and computes $c_{b_0} = \mathrm{com}_{ck}(b;s)\prod_{k=1}^{2\mu-2} c_{b_k}^{-x^k}$. Now it runs the simulator for the SHVZK argument on $\vec{C}'_1, \ldots, \vec{C}'_{m'}, C', c_{A'_1}, \ldots, c_{A'_{m'}}$.

Both in real arguments and simulated arguments we have uniformly random ciphertexts $E_0, \ldots, E_{\mu-2}, E_\mu, \ldots, E_{2\mu-2}$ and $E_{\mu-1} = C$. The commitment scheme is perfectly hiding, so we have that $c_{b_1}, \ldots, c_{b_{\mu-2}}, c_{b_\mu}, \ldots, c_{b_{2\mu-2}}$ are uniformly random in real arguments and $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0;0)$ just as in the simulation. In a real argument we have $b_0, s_0 \leftarrow \mathbb{Z}_q$ chosen at random, which implies that $b, s$ are uniformly random just as in the simulation. Given all these values the verification equation uniquely defines $c_{b_0} = \mathrm{com}_{ck}(b;s)\prod_{k=1}^{2\mu-2} c_{b_k}^{-x^k}$. It now follows from the perfect SHVZK of the underlying argument that the simulation is perfect.

Now, we have to show that the argument has witness-extended emulation. The emulator will run the argument with a random challenge $x$ and output the resulting transcript. If the verifier rejects in the transcript the emulator is done, but if the verifier accepts the emulator will try to extract a witness. The emulator rewinds the argument to the challenge phase and repeatedly runs it with fresh challenges until it gets $2\mu - 1$ accepting transcripts. In this process it uses the witness-extended emulator for the underlying SHVZK argument to get witnesses $\vec{a}'_1, r'_1, \ldots, \vec{a}'_{m'}, r'_{m'}, \rho'$. The emulator on average makes $2\mu - 1$ runs when the prover has non-zero probability of succeeding giving an expected polynomial time.

Suppose now the emulator has satisfactory answers to $2\mu - 1$ challenges $x_1, \ldots, x_{2\mu-1}$. The matrix $X$ with columns of the form $\vec{x}_\ell = (1,\ x_\ell,\ \ldots, x_\ell^{2\mu-2})^T$ is a transposed Vandermonde matrix and therefore if $x_1, \ldots, x_{2\mu-1}$ are different $X$ is invertible. Define $\vec{b}_x = (b^{(1)},\ \ldots,\ b^{(2\mu-1)})$ and $\vec{s}_x = (s^{(1)},\ \ldots,\ s^{(2\mu-1)})$. We now have

$$\vec{c}_b = (\vec{c}_b^{\,X})^{X^{-1}} = \mathrm{com}_{ck}(\vec{b}_x; \vec{s}_x)^{X^{-1}} = \mathrm{com}_{ck}(\vec{b}_x X^{-1}; \vec{s}_x X^{-1})$$

giving us openings $b_0, \ldots, b_{2\mu-1}, s_0, \ldots, s_{2\mu-1}$ of all the commitments $c_{b_0}, \ldots, c_{b_{2\mu-2}}$ the prover sent.

The witness-extended emulator for the underlying SHVZK argument provides us with openings $(\vec{a}'_j)^{(\ell)}, (r'_j)^{(\ell)}$ of $c_{A'_j} = \prod_{k=1}^{\mu} c_{A_{j\mu+k}}^{x^{j-1}}$. Using a similar technique as we did for the $b_i$'s we can by taking appropriate linear combinations find openings $\vec{a}_1, r_1, \ldots, \vec{a}_m, r_m$ of $c_{A_1}, \ldots, c_{A_m}$.

The witness-extended emulations of the underlying argument give us $(\rho')^{(1)}, \ldots, (\rho')^{(2\mu-1)}$ in response to the challenges $x_1, \ldots, x_\ell$ satisfying $(C')^{(\ell)} = \mathcal{E}_{pk}(1; (\rho')^{(\ell)})\prod_{i=1}^{m'} \left(\vec{C}_i'^{(\ell)}\right)^{(\vec{a}'_i)^{(\ell)}}$. We will now argue that a linear combination of those will give us the desired $\rho$, needed to complete our argument. Let $\vec{y} = (y_1, \ldots, y_{2\mu-1})$ be the $\mu - 1$th column of $X^{-1}$ such that $\sum_{\ell=1}^{2\mu-1} x_\ell^k y_\ell = 1$ for $k = \mu - 1$, else 0, and define $\rho = \sum_{\ell=1}^{2\mu-1}(\rho')^{(\ell)} y_\ell$. The verifier checks $c_{b_{\mu-1}} = \mathrm{com}_{ck}(0;0)$, so the binding property implies that

$b_{\mu-1} = 0$. We then have

$$\begin{aligned}
C &= E_{\mu-1} = \prod_{\ell=1}^{2\mu-1}\left(\prod_{k=0}^{2\mu-2} E_k^{x_\ell^k}\right)^{y_\ell} = \prod_{\ell=1}^{2\mu-1}\Big(\mathcal{E}_{pk}(G^{b^{(\ell)}};0)(C')^{(\ell)}\Big)^{y_\ell} \\
&= \prod_{\ell=1}^{2\mu-1}\left(\mathcal{E}_{pk}(G^{b^{(\ell)}};(\rho')^{\ell})\prod_{u=1}^{m'}(\vec{C}'^{(\ell)}_u)^{(\vec{a}'_u)^{(\ell)}}\right)^{y_\ell} \\
&= \mathcal{E}_{pk}\Big(G^{\sum_{\ell=1}^{2\mu-1} b^{(\ell)}y_\ell};\sum_{\ell=1}^{2\mu-1}(\rho')^{(\ell)}y_\ell\Big)\prod_{\ell=1}^{2\mu-1}\left(\prod_{u=0}^{m'-1}\Big(\prod_{i=1}^{\mu}\vec{C}_{u\mu+i}^{x_\ell^{\mu-i}}\Big)^{\sum_{j=1}^{\mu}x_\ell^{j-1}\vec{a}_{u\mu+j}}\right)^{y_\ell} \\
&= \mathcal{E}_{pk}(G^{b_{\mu-1}};\rho)\prod_{\ell=1}^{2\mu-1}\left(\prod_{u=0}^{m'-1}\prod_{k=0}^{2\mu-2}\Big(\prod_{\substack{i=1,j=1\\ j=(k-\mu+1)+i}}^{2\mu-2}\vec{C}_{u\mu+i}^{\vec{a}_{u\mu+j}}\Big)^{x_\ell^k}\right)^{y_\ell} \\
&= \mathcal{E}_{pk}(G^{0};\rho)\prod_{u=0}^{m'-1}\prod_{k=0}^{2\mu-2}\Big(\prod_{\substack{i=1,j=1\\ j=(k-\mu+1)+i}}^{2\mu-2}\vec{C}_{u\mu+i}^{\vec{a}_{u\mu+j}}\Big)^{\sum_{\ell=1}^{2\mu-1}y_\ell x_\ell^k} \\
&= \mathcal{E}_{pk}(1;\rho)\prod_{u=0}^{m'-1}\prod_{\substack{i=1,j=1\\ j=(\mu-1-\mu+1)+i}}^{2\mu-2}\vec{C}_{u\mu+i}^{\vec{a}_{u\mu+j}} = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^{m}\vec{C}_i^{\vec{a}_i}.
\end{aligned}$$

This means the emulator has extracted a valid witness. □

## 5 Product Argument

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

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

We give an argument of knowledge that $c_b$ is correct, i.e., it contains $\prod_{j=1}^{m} a_{1j},\ldots,\prod_{j=1}^{m} a_{nj}$. Next, we give an argument of knowledge that $b$ is the product of the values inside $c_b$. We will present these two arguments in Sections 5.1 and 5.3. Here, we just give an overview of the protocol.

**Common reference string:** $pk, ck$.

**Statement:** $\vec{c}_A \in \mathbb{G}^m$ and $b \in \mathbb{Z}_q$.

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

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

**Initial message:** Pick $s \leftarrow \mathbb{Z}_q$ and compute $c_b = \mathrm{com}_{ck}(\prod_{j=1}^m a_{1j}, \dots, \prod_{j=1}^m a_{nj}; s)$. Send $c_b$ to the verifier.

Engage in an SHVZK argument of knowledge as described in Section 5.1 of $c_b = \mathrm{com}_{ck}(\prod_{j=1}^m a_{1j}, \dots, \prod_{j=1}^m a_{nj}; s)$, where $a_{11}, \dots, a_{nm}$ are the committed values in $\vec{c}_A$.

Engage (in parallel) in an SHVZK argument of knowledge as described in Section 5.3 of $b$ being the product of the committed values in $c_b$.

**Verification:** The verifier accepts if $c_b \in \mathbb{G}$ and both SHVZK arguments are convincing.

**Theorem 8.** *The protocol is a public coin perfect SHVZK argument of knowledge of openings $a_{11}, \dots, a_{nm}, r_1, \dots, r_m \in \mathbb{Z}_q$ such that $b = \prod_{i=1}^n \prod_{i=1}^m a_{ij}$.*

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

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

It remains to argue that we have witness-extended emulation. The witness-extended emulator runs the argument using the witness-extended simulators for the underlying arguments such that if it is successful it has openings $a_{11}, \dots, a_{nm}, r_1, \dots, r_m$ of $\vec{c}_A$ and an opening $b_1, \dots, b_n, s$ of $c_b$ such that

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

This implies that the extracted openings of $\vec{c}_A$ satisfy $\prod_{i=1}^n \prod_{j=1}^m a_{ij} = b$. □

## 5.1 Hadamard product argument

We will give an argument for committed values $a_{11}, \dots, a_{nm}$ and $b_1, \dots, b_n$ satisfying $b_i = \prod_{j=1}^m a_{ij}$. It will be convenient to write this with vector notation. We have commitments $\vec{c}_A$ and $c_b$ and the prover wants to argue knowledge of openings to tuples $\vec{a}_1, \dots, \vec{a}_m, \vec{b} \in \mathbb{Z}_q^n$ such that $\vec{b} = \prod_{i=1}^m \vec{a}_i$, where we use entry-wise multiplication, which is also known as the Hadamard product.

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

$$\vec{b}_1 = \vec{a}_1 \qquad \vec{b}_2 = \prod_{i=1}^2 \vec{a}_i \qquad \dots \qquad \vec{b}_{m-1} = \prod_{i=1}^{m-1} \vec{a}_i \qquad \vec{b}_m = \prod_{i=1}^m \vec{a}_i.$$

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

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

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

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

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

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

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

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

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

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

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

The prover will now use the zero argument from Section 5.2 to demonstrate that

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

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

**Common reference string:** $pk, ck$.

**Statement:** $\vec{c}_A, c_b$.

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

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

**Initial message:** Define $\vec{b}_1 = \vec{a}_1, \vec{b}_2 = \vec{a}_1\vec{a}_2, \ldots, \vec{b}_{m-1} = \vec{a}_1 \cdots \vec{a}_{m-1}$, and $\vec{b}_m = \vec{b}$. Pick $s_2, \ldots, s_{m-1} \leftarrow \mathbb{Z}_q$ and compute $c_{B_2} = \mathrm{com}_{ck}(\vec{b}_2; s_2), \ldots, c_{B_{m-1}} = \mathrm{com}_{ck}(\vec{b}_{m-1}; s_{m-1})$. Define $s_1 = r_1$ and $s_m = s$, and set $c_{B_1} = c_{A_1}$ and $c_{B_m} = c_b$.

Send $\vec{c}_B$.

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

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

Define $c_{D_i} = c_{B_i}^{x^i}$ and $c_D = \prod_{i=1}^{m-1} c_{B_{i+1}}^{x^i}$ and $c_{-1} = \mathrm{com}_{ck}(-\vec{1}; 0)$ and engage in the SHVZK zero argument described in Section 5.2 for the committed values satisfying

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

The prover's witness in this argument consists of the openings of $c_{A_2}, \ldots, c_{A_m}, c_{-1}$ and the openings of $c_{D_1}, \ldots, c_{D_{m-1}}, c_D$. The latter openings can be computed as $\vec{d}_1 = x\vec{b}_1, t_1 = xs_1, \ldots, \vec{d}_{m-1} = x^{m-1}\vec{b}_{m-1}, t_{m-1} = x^{m-1}s_{m-1}$, and $\vec{d} = \sum_{i=1}^{m-1} x^i \vec{b}_{i+1}, t = \sum_{i=1}^{m-1} x^i s_{i+1}$.

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

**Theorem 9.** *The protocol is a public coin perfect SHVZK argument of knowledge of committed vectors $\vec{a}_1, \ldots, \vec{a}_m, \vec{b}$ such that $\vec{b} = \prod_{i=1}^m \vec{a}_i$.*

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

Given challenges $x, y$ the simulator picks $s_2, \ldots, s_{m-1} \leftarrow \mathbb{Z}_q$, computes $c_{B_2} = \mathrm{com}_{ck}(\vec{0}; s_2), \ldots, c_{B_{m-1}} = \mathrm{com}_{ck}(\vec{0}; s_{m-1})$ and runs the SHVZK simulator for the underlying zero knowledge argument. The perfect hiding property of the commitments and the perfect SHVZK of the underlying zero argument shows that this is a perfect simulation.

It remains to argue that we have witness-extended emulation. The witness-extended emulator runs the zero argument with witness-extended emulation to get openings of $c_{A_2}, \ldots, c_{A_m}, c_{-1}$ and $c_{D_1}, \ldots, c_{D_{m-1}}, c_D$ of the form $\vec{a}_2, r_2, \ldots, \vec{a}_m, r_m, -\vec{1}, 0$ and $\vec{d}_1, t_1, \ldots, \vec{d}_{m-1}, t_{m-1}, \vec{d}, t$. This gives us openings of $\vec{c}_B$ computed as

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

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

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

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

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

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

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

which has negligible probability of holding for random $x$ unless $\vec{b}_2 = \vec{a}_2\vec{b}_1, \ldots, \vec{b}_m = \vec{a}_m\vec{b}_{m-1}$. This shows the extracted openings satisfy $\vec{b} = \vec{b}_m = \prod_{i=1}^m \vec{a}_i$ as required in the statement. □

## 5.2 Zero argument

Given a bilinear map $* : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q$ (in our case defined by $y \in \mathbb{Z}_q$ as described in Section 5.1) and commitments to $\vec{a}_1, \vec{b}_0, \ldots, \vec{a}_m, \vec{b}_{m-1}$ the prover wants to show that $0 = \sum_{i=1}^m \vec{a}_i * \vec{b}_{i-1}$.

The prover picks $\vec{a}_0, \vec{b}_m \leftarrow \mathbb{Z}_q^n$ and calculates commitments to these values. The prover computes for $k = 0, \ldots, 2m$ the values $d_k = \sum_{\substack{0 \le i,j \le m \\ j=(m-k)+i}} \vec{a}_i * \vec{b}_j$ and commits to them. Observe that $d_{m+1} = \sum_{i=1}^m \vec{a}_i * \vec{b}_{i-1} = 0$. The prover sets $c_{D_{m+1}} = \mathrm{com}_{ck}(0; 0)$ such that the verifier can see $d_{m+1} = 0$.

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

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

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

**Common reference string:** $pk, ck$.

**Statement:** $\vec{c}_A, \vec{c}_B$ and a specification of a bilinear map $* : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q$.

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

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

**Initial message:** Pick $\vec{a}_0, \vec{b}_m \leftarrow \mathbb{Z}_q^n$ and $r_0, s_m \leftarrow \mathbb{Z}_q$ and compute

$$c_{A_0} = \mathrm{com}_{ck}(\vec{a}_0; r_0) \qquad\qquad c_{B_m} = \mathrm{com}_{ck}(\vec{b}_m; s_m).$$

Compute $d_0, \ldots, d_{2m}$ as

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

Pick $\vec{t} = (t_0 \ldots t_{2m}) \leftarrow \mathbb{Z}_q^{2m+1}$ and set $t_{m+1} = 0$ and compute commitments $\vec{c}_D = \mathrm{com}_{ck}(\vec{d}; \vec{t})$.

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

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

**Answer:**

$$\vec{a} = \sum_{i=0}^{m} x^i \vec{a}_i \qquad r = \sum_{i=0}^{m} x^i r_i \qquad \vec{b} = \sum_{j=0}^{m} x^{m-j} \vec{b}_j \qquad s = \sum_{j=0}^{m} x^{m-j} s_j \qquad t = \sum_{k=0}^{2m} x^k t_k.$$

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

**Verification:** Accept if $c_{A_0}, c_{B_m} \in \mathbb{G}, \vec{c}_D \in \mathbb{G}^{2m+1}$, $c_{D_{m+1}} = \mathrm{com}_{ck}(0;0)$, $\vec{a}, \vec{b} \in \mathbb{Z}_q^n$, $r, s, t \in \mathbb{Z}_q$ and

$$\prod_{i=0}^{m} c_{A_i}^{x^i} = \mathrm{com}_{ck}(\vec{a}, r) \qquad \prod_{j=0}^{m} c_{B_j}^{x^{m-j}} = \mathrm{com}_{ck}(\vec{b}; s) \qquad \prod_{k=0}^{2m} c_{D_k}^{x^k} = \mathrm{com}_{ck}(\vec{a} * \vec{b}; t).$$

**Theorem 10.** *The protocol is a public coin perfect SHVZK argument of knowledge of committed values* $\vec{a}_1, \vec{b}_0, \ldots, \vec{a}_m, \vec{b}_{m-1}$ *such that* $0 = \sum_{i=1}^{m} \vec{a}_i * \vec{b}_{i-1}$.

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

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

Perfect completeness now follows by direct verification.

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

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

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

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

It remains to prove that we have witness-extended emulation. The emulator runs the argument $\langle \mathcal{P}^*, \mathcal{V} \rangle$ and if the transcript is accepting it has to extract a witness. Therefore, it rewinds to the challenge phase and runs it again with random challenges until it has $2m+1$ accepting transcripts. On average the witness-extended emulator will be making $2m+1$ arguments so it runs in expected polynomial time.

There is negligible probability for the emulator ending up with $2m+1$ accepting arguments with a collision in the challenges. Provided the challenges are different, the emulator now has accepting transcripts for $x_0, \ldots, x_{2m}$ satisfying for each $x_\ell$ that $c_{D_{m+1}} = \mathrm{com}_{ck}(0; 0)$ and

$$\prod_{i=0}^{m} c_{A_i}^{x^i} = \mathrm{com}_{ck}(\vec{a}^{(\ell)}, r^{(\ell)}) \qquad \prod_{j=0}^{m} c_{B_j}^{x^{m-j}} = \mathrm{com}_{ck}(\vec{b}^{(\ell)}; s^{\ell}) \qquad \prod_{k=0}^{2m} c_{D_k}^{x^k} = \mathrm{com}_{ck}(\vec{a}^{(\ell)} * \vec{b}^{(\ell)}; t^{(\ell)}).$$

Since the vectors $(1, x_i, \ldots, x_i^{2m})$ form the columns of a transposed Vandermonde matrix and all the the $x_i$'s are different we can find the inverse matrix $X^{-1}$. Define $\vec{d}_x = (a^{(0)} * b^{(0)}, \ \ldots, \ a^{(2m)} * b^{(2m)})$ and $\vec{t}_x = (t^{(0)}, \ \ldots, \ t^{(2m)})$, this gives for each $i = 0, \ldots, 2m$ an opening of $c_{D_i}$ applying $\vec{c}_D = (\vec{c}_D^{X})^{X^{-1}} = \mathrm{com}_{ck}(\vec{d}_x; \vec{t}_x)^{X^{-1}} = \mathrm{com}_{ck}(\vec{d}_x X^{-1}; \vec{t}_x X^{-1})$. The emulator gets openings of $c_{A_0}, \ldots, c_{A_m}$ and $c_{B_0}, \ldots, c_{B_m}$ in a similar manner.

Having openings of $\vec{c}_A, \vec{c}_B, \vec{c}_D$ to values $\vec{a}_0, \ldots, \vec{a}_m, \vec{b}_0, \ldots, \vec{b}_m, d_0, \ldots, d_{2m}$ the binding property of the commitment scheme implies that the answer to a random challenge $x$ is of the form

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

This implies

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

The Schwartz-Zippel lemma implies that the prover has negligible chance of making an acceptable argument unless $d_{m+1} = \sum_{i=1}^{m} \vec{a}_i * \vec{b}_{i-1}$. Since $d_{m+1} = 0$ this gives us that the extracted openings satisfy $0 = \sum_{i=1}^{m} \vec{a}_i * \vec{b}_{i-1}$. $\square$

## 5.3 Single value product argument

For completeness we restate the following 3-move argument of knowledge [Gro10] of committed single values having a particular product.

**Common reference string:** $pk, ck$.

**Statement:** $c_a \in \mathbb{G}$ and $b \in \mathbb{Z}_q$.

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

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

**Initial message:** Compute

$$b_1 = a_1 \qquad b_2 = a_1 a_2 \qquad \ldots \qquad b_n = \prod_{i=1}^{n} a_i.$$

Pick $d_1, \ldots, d_n, r_d \leftarrow \mathbb{Z}_q$. Define $\delta_1 = d_1$ and $\delta_n = 0$ and pick $\delta_2, \ldots, \delta_{n-1} \leftarrow \mathbb{Z}_q$. Pick $s_1, s_x \leftarrow \mathbb{Z}_q$ and compute

$$c_d = \mathrm{com}_{ck}(\vec{d}; r_d) \qquad c_\delta = \mathrm{com}_{ck}(-\delta_1 d_2, \ldots, -\delta_{n-1} d_n; s_1)$$
$$c_\Delta = \mathrm{com}_{ck}(\delta_2 - a_2\delta_1 - b_1 d_2, \ldots, \delta_n - a_n \delta_{n-1} - b_{n-1} d_n; s_x).$$

Send: $c_d, c_\delta, c_\Delta$.

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

**Answer:** Compute

$$\tilde{a}_1 = x a_1 + d_1 \qquad \ldots \qquad \tilde{a}_n = x a_n + d_n \qquad \tilde{r} = x r + r_d$$
$$\tilde{b}_1 = x b_1 + \delta_1 \qquad \ldots \qquad \tilde{b}_n = x b_n + \delta_n \qquad \tilde{s} = x s_x + s_1.$$

Send: $\tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_n, \tilde{b}_n, \tilde{r}, \tilde{s}$.

**Verification:** The verifier accepts if $c_d, c_\delta, c_\Delta \in \mathbb{G}$ and $\tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_n, \tilde{b}_n\ \tilde{r}, \tilde{s} \in \mathbb{Z}_q$ and

$$c_a^x c_d = \mathrm{com}_{ck}(\tilde{a}_1, \ldots, \tilde{a}_n; \tilde{r}) \qquad c_\Delta^x c_\delta = \mathrm{com}_{ck}(x\tilde{b}_2 - \tilde{b}_1\tilde{a}_2, \ldots, x\tilde{b}_n - \tilde{b}_{n-1}\tilde{a}_n; \tilde{s})$$
$$\tilde{b}_1 = \tilde{a}_1 \qquad \tilde{b}_n = xb.$$

**Theorem 11.** *The protocol is a public coin perfect SHVZK argument of knowledge of an opening $a_1, \ldots, a_n, r$ such that $c_a = \mathrm{com}_{ck}(\vec{a}; r)$ and $b = \prod_{i=1}^{n} a_i$.*

*Proof.* Perfect completeness follows from

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

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

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

are distributed just like in a real proof. These choices uniquely determine $c_d, c_\delta$ according to the verification equations giving us that simulated and real proofs are identically distributed.

Finally, we will show that the protocol has witness-extended emulation. The witness-extended emulator runs the argument on a random challenge $x$. If the prover is successful, the witness-extended emulator rewinds to the second move until it gets another convincing arguments on a challenge $x'$. Now, if $x \neq x'$ we have

$$c_a^x c_d = \mathrm{com}_{ck}(\tilde{a}_1, \ldots, \tilde{a}_n; \tilde{r}) \qquad c_a^{x'} c_d = \mathrm{com}_{ck}(\tilde{a}'_1, \ldots, \tilde{a}'_n; \tilde{r}'),$$

which implies $c_a^{x-x'} = \mathrm{com}_{ck}(\tilde{a}_1 - \tilde{a}'_1, \ldots, \tilde{a}_n - \tilde{a}'_n; \tilde{r} - \tilde{r}')$ giving us an opening of $c_a$. The equation $c_d = c_a^{-x}\mathrm{com}_{ck}(\tilde{a}_1, \ldots, \tilde{a}_n; \tilde{r})$ then gives us an opening of $c_d$. Similarly, we can get openings of $c_\delta, c_\Delta$ provided $x \neq x'$.

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

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

where $p_{n-1}$ is a fixed degree $n-1$ polynomial determined by the committed values. Since $\tilde{a}_i = xa_i + d_i$ the Schwartz-Zippel lemma implies that there is negligible probability of satisfying this equation for random $x$ unless indeed $b = \prod_{i=1}^n a_j$. □

## 6 Implementation and Comparison

We will now compare our protocol with the most efficient shuffle arguments for ElGamal encryption. First, we compare the theoretical performance of the schemes without any optimization. Second, we compare an implementation of our protocol with the implementation by Furukawa et al. [FMM$^+$02] and with the implementation in the Verificatum mix-net library [Wik10].

**Theoretical comparison.** Previous work in the literature mainly investigated the case where we use ElGamal encryption and commitments over the same group $\mathbb{G}$, i.e., $\mathbb{H} = \mathbb{G} \times \mathbb{G}$. Table 1 gives the asymptotic behavior of these protocols compared to our protocol for $N = mn$ as $m$ and $n$ grows.

| SHVZK argument | Rounds | Time $\mathcal{P}$ Expos | Time $\mathcal{V}$ Expos | Size Elements |
|---|---|---|---|---|
| [FS01] | 3 | $8N$ | $10N$ | $5N$ $\mathbb{G}$+ $N$ $\mathbb{Z}_q$ |
| [FMM$^+$02] | 5 | $9N$ | $10N$ | $5N$ $\mathbb{G}$ + $N$ $\mathbb{Z}_q$ |
| [Gro10] | 7 | $6N$ | $6N$ | $3N$ $\mathbb{Z}_q$ |
| [Fur05] | 3 | $7N$ | $8N$ | $N$ $\mathbb{G}$+ $2N$ $\mathbb{Z}_q$ |
| [TW10] | 5 | $9N$ | $11N$ | $3N$ $\mathbb{G}$ + $4N$ $\mathbb{Z}_q$ |
| [GI08] | 7 | $3mN$ | $4N$ | $3m^2$ $\mathbb{G}$+ $3n$ $\mathbb{Z}_q$ |
| This paper | 9 | $2\log(m)N$ | $4N$ | $11m$ $\mathbb{G}$+ $5n$ $\mathbb{Z}_q$ |

Table 1: Comparison of the protocols with ElGamal encryption.

In our protocol, we may as detailed in Section 4.1 use FFT techniques to reduce the prover's computation to $O(N \log m)$ exponentiations as listed in Table 1. Furthermore, by increasing the round complexity as in

Section 4.2 we could even get a linear complexity of $O(N)$ exponentiations. These techniques do not apply to the other shuffle arguments; in particular it is not possible to use FFT techniques to reduce the factor $m$ in the shuffle by Groth and Ishai [GI08].

As the multi-exponentiation argument, which is the most expensive step, already starts in round 3 we can insert two rounds of interactive reduction as described in Section 4.2 without increasing the round complexity above 9 rounds. For practical parameters this would give us enough of a reduction to make the prover's computation comparable to the schemes with linear $O(N)$ computation.

The figures in Table 1 are for non-optimized versions of the schemes. All of the schemes may for instance benefit from the use of multi-exponentiation techniques, see e.g. Lim [Lim00] for how to compute a product of $n$ exponentiations using only $O(\frac{n}{\log n})$ multiplications. The schemes may also benefit from randomization techniques, where the verifier does a batch verification of all the equations it has to check.

**Experimental results.** We implemented our shuffle argument in C++ using the NTL library by Shoup [Sho09] for the underlying modular arithmetic. We experimented with five different implementations to compare their relative merit:

1. Without any optimizations at all.
2. Using multi-exponentiation techniques.
3. Using multi-exponentiation and the Fast Fourier transform.
4. Using multi-exponentiation and a round of the interactive technique with $\mu = 4$ and Toom-Cook for $m' = 4$ giving $m = \mu m' = 16$.
5. Using multi-exponentiation and two rounds of the interactive technique first with $\mu = 4$ and Toom-Cook for $m' = 4$ giving $m = \mu^2 m' = 64$.

In our experiments we used ElGamal encryption and commitments over the same group $\mathbb{G}$, which was chosen as an order $q$ subgroup of $\mathbb{Z}_p^*$, where $|q| = 160$ and $|p| = 1024$. These security parameters are on the low end for present day use but facilitate comparison with earlier work. The results can be found in Table 2.

| | Optimization | Total time | Time $\mathcal{P}$ | Time $\mathcal{V}$ | Size |
|---|---|---|---|---|---|
| $m = 8$ | Unoptimized | 570 | 462 | 108 | 4.3 MB |
| | Multi-expo | 162 | 125 | 37 | |
| | FFT | 228 | 190 | 38 | |
| $m = 16$ | Unoptimized | 900 | 803 | 97 | 2.2 MB |
| | Multi-expo | 193 | 169 | 24 | |
| | FFT | 245 | 221 | 24 | |
| | Toom-Cook | 139 | 101 | 38 | 2.2 MB |
| $m = 64$ | Multi-expo | 615 | 594 | 21 | 0.7MB |
| | FFT | 328 | 307 | 20 | |
| | Toom-Cook | 128 | 91 | 18 | MB |

Table 2: Run time of the shuffle arguments in seconds on a Core2Duo 2.53 GHz, 3 MByte L2-Cache, 4 GByte Ram machine for $N = 100,000$ and different choices of $m$.

Table 2 states the results for $N = 100,000$, $m = 8, 16, 64$ on our machine. We see that the plain multi-exponentiation techniques yield better results than the FFT method for small $m$; the better asymptotic behavior of the FFT only kicks in for $m > 16$. As expected the Toom-Cook inspired version with added interaction has the best running time and communication cost.

**Comparison with other implementations.** Furukawa, Miyauchi, Mori, Obana and Sako [FMM$^+$02] gave performance results for a mix-net using a version of the Furukawa-Sako [FS01] shuffle arguments. They optimized the mix-net by combining the shuffling and decryption operations into one. They used three shuffle centers communicating with each other and their results include both the process of shuffling and the cost of the arguments. So, to compare the values we multiply our shuffle argument times with 3 and add the cost of our shuffling operation on top of that. The comparison can be found in Table 3.

| $N = 100,000$ | [FMM$^+$02] | This paper |
|---|---|---|
| Single argument | 51 min | 15 min |
| Argument size | 66 MB | 0.7 MB |
| Total mix-net time | 3 hrs 44 min | 53 min |

Table 3: Runtime comparison of [FMM$^+$02] (CPU: 1 GHz, 256 MB) to our shuffle argument (Toom-Cook with $m = 64$, CPU: 1.4 GHz, 4 GB) .

We expected to get better performance than they did and indeed we see that our argument is much faster and the communication is a factor 100 smaller. When adding the cost of shuffling and decryption to our argument we still have a speedup of a factor 3 in Table 3 when comparing the two mix-net implementations and taking the difference in the machines into account.

Recently, Furukawa et al. [FMS10] announced a new implementation based on elliptic curve groups. Due to the speed of using elliptic curves this gave them a speedup of a factor 3. A similar speedup can be expected for our shuffle argument if we switch to using elliptic curves in our implementation.

Stamer [Sta05] reported on an implementation of Groth's shuffle [Gro10]. However, this was an unoptimized version and he only reported on results up to $N = 1,000$.

Recently Wikström made a complete implementation of a mix-net in Java in [Wik10] called Verificatum, which is based on the shuffle argument in [TW10]. To produce comparable data, we ran the demo file with only one mix party in the non-interactive mode using the same modular group as in our protocol. We only counted the time of the relevant parts. As described in Table 1 the theoretical performance of the shuffle argument takes $20N$ exponentiations, our prover needs with Toom-Cook and 2 extra rounds of interaction $12N$ exponentiations and our verifier $4N$, so in total $16N$ exponentiations. So we expect a similar running time for the Verificatum mix-net. As shown in Table 4 we perform slightly better, but due to the different programming languages used and different levels of optimization in the code we will not draw any conclusion except that both protocols are efficient and usable in current applications. In terms of size it is clear that our arguments leave a much smaller footprint than Verificatum; we save a factor 50 in the communication.

| $N = 100,000$ | [TW10] | This paper Toom-Cook |
|---|---|---|
| Single argument | 5 min | 2 min |
| Argument size | 37.7 MB | 0.7 MB |

Table 4: Runtime comparison of [TW10] to our shuffle argument on our machine (CPU: 2.54 GHz, MB) .

## Acknowledgment

We would like to thank Douglas Wikström for discussions and help regarding our comparison with the shuffle argument used in Verificatum [Wik10].

## References

[Abe98] M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. *EUROCRYPT*, LNCS vol. 1403:437–447, 1998.

[Abe99] M. Abe. Mix-networks on permutation networks. *ASIACRYPT*, LNCS vol. 1716:258–273, 1999.

[AH01] M. Abe and F. Hoshino. Remarks on mix-network based on permutation networks. *PKC*, LNCS vol. 1992:317–324, 2001.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[Coo66] S. Cook. *On the minimum computation time of functions*. PhD thesis, Department of Mathematics, Harvard University, 1966. `http://cr.yp.to/bib/1966/cook.html`.

[CT65] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comp.*, 19:297–301, 1965.

[DK00] Y. Desmedt and K. Kurosawa. How to break a practical mix and design a new one. *EUROCRYPT*, LNCS vol. 1807:557–572, 2000.

[ElG84] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *CRYPTO*, LNCS vol. 196:10–18, 1984.

[FMM$^+$02] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. *Financial Cryptography*, LNCS vol. 2357:16–30, 2002.

[FMS10] J. Furukawa, K. Mori, and K. Sako. An implementation of a mix-net based network voting scheme and its use in a private organization. *Towards Trustworthy Elections*, LNCS vol. 6000:141–154, 2010.

[FS01] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. *CRYPTO*, LNCS vol. 2139:368–387, 2001.

[Fur05] J. Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE Transactions*, 88-A(1):172–188, 2005.

[GI08] J. Groth and Y. Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. *EUROCRYPT*, LNCS vol. 4965:379–396, 2008.

[GL07] J. Groth and S. Lu. Verifiable shuffle of large size ciphertexts. *PKC*, LNCS vol. 4450:377–392, 2007.

[GMY06] J. Garay, P. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. *J. Cryptology*, 19(2):169–209, 2006.

[Gro04] J. Groth. Honest verifier zero-knowledge arguments applied. *Dissertation Series DS-04-3, BRICS, 2004. PhD thesis. xii+119*, 2004.

[Gro09] J. Groth. Linear algebra with sub-linear zero-knowledge arguments. *CRYPTO*, LNCS vol. 5677:192–208, 2009.

[Gro10] J. Groth. A verifiable secret shuffle of homomorphic encryptions. *J. Cryptology*, 23(4):546–579, 2010.

[JJR02] M. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. *USENIX Security Symposium*, pages 339–353, 2002.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Dokl.*, 7:595–596, 1963.

[Lim00] C. Lim. Efficient multi-exponentiation and application to batch verification of digital signatures. `http://dasan.sejong.ac.kr/~chlim/pub/multiexp.ps`, 2000.

[Lin03] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *J. Cryptology*, 16(3):143–184, 2003.

[Nef01] C. A. Neff. A verifiable secret shuffle and its application to e-voting. *ACM CCS*, pages 116–125, 2001.

[Nef03] C. A. Neff. Verifiable mixing (shuffling) of elgamal pairs. `http://people.csail.mit.edu/rivest/voting/papers/Neff-2004-04-21-ElGamalShuffles.pdf`, 2003.

[PBDV04] K. Peng, C. Boyd, E. Dawson, and K. Viswanathan. A correct, private, and efficient mix network. *PKC*, LNCS vol. 2947:439–454, 2004.

[Ped91] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. *CRYPTO*, LNCS vol. 576:129–140, 1991.

[Sho09] V. Shoup. Ntl library. `http://www.shoup.net/ntl/`, 2009.

[SK95] K. Sako and J. Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. *EUROCRYPT*, LNCS vol. 921:393–403, 1995.

[Sta05] H. Stamer. Efficient electronic gambling: An extended implementation of the toolbox for mental card games. *WEWoRC*, P-74:1–12, 2005.

[Too00] A. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. `http://www.de.ufpe.br/~toom/my_articles/engmat/MULT-E.PDF`, 2000.

[TW10] B. Terelius and D. Wikström. Proofs of restricted shuffles. *AFRICACRYPT*, LNCS vol. 6055:100–113, 2010.

[Wik02] D. Wikström. The security of a mix-center based on a semantically secure cryptosystem. *INDOCRYPT*, LNCS vol. 2551:368–381, 2002.

[Wik09] D. Wikström. A commitment-consistent proof of a shuffle. *ACISP*, LNCS vol. 5594:407–421, 2009.

[Wik10] D. Wikström. Verificatum. `http://www.verificatum.com/`, 2010.