Computationally efficient mix-nets

Ivo Kubjas

1 Introduction

The current online voting system used in Estonia has not employed mathematically secure verifiable proofs of correctness for ensuring the honesty of the tabulation server [Com13a]. The auditing is based on observation of the procedures and if the procedures are done according to the protocol.

Even though the protocol was not followed, as the cast votes were transferred from ballot storing server to tabulation server on a USB memory card [Com13b], while they were supposed to be transmitted on a DVD, the audit claimed that this incident did not have any evident and permanent consequences to the procedure. On the other hand, there are many examples which suggest that USB memory sticks should not be considered secure [Hua13].

As the voting process is a required part (but not sufficient, as we have seen lately) of a democratic society, then this should not be based on *ad-hoc* methods and unverified hardware. As hardware verification is nearly infeasible considering the costs and skills required, then the trust model could be based on mathematically verifiable proofs of correctness.

Secrecy of a ballot is one of the most fundamental requirements of the elections. For example, using unverified hardware the tallying server may leak some information about the content of a specific ballot. The leak could occur through side-channel attacks [Meh14] [GST13] or by assistance of a malicious adversary. If the attacker has also access to signed ballots, then the voter's choice could be revealed.

If the connection between the tallied vote and signed vote could be obfuscated then the success of this specific attack would be negligible. But this obfuscation raises other problems. For example, how to ensure that the votes are not modified in favour of some third party? One solution is to use mix-nets which permutes and re-encrypts the ballots, while giving a proof that the actions are performed correctly. The mix-net may be a larger system which can also perform threshold decryption, key distribution *etc.* The basis of the mix-net is the shuffle, which is used for permuting and re-encryption. In the case of elections, the number of votes is usually large and this has lead to optimizing the computation to construct the proofs and perform the mixing.

If the mix-net is combined with tabulation on ciphertexts and a proof of ballot inclusion, then the voter can be highly confident that his vote is correctly counted towards the result and that his vote is anonymous. Furthermore, the mixing could be performed by independent parties (for example, by political parties).

In this survey, we will observe some mix-net constructions and give a hint on the optimization techniques. We will not inspect the full details as the optimization methods are rather specific. Furthermore, programming techniques can be used to further speed up the computation.

We will start with the preliminaries which are required to describe the requirements for the cryptosystem used. Then, we will define some notation which is used in the protocol descriptions. Now it is possible to give an intuition of the mix-net construction and then we will show the full (abstracted) protocols. We also discuss the computational complexity of the protocols.

This survey is based on the work by Bayer and Groth [BG12].

2 Preliminaries

The mix-nets rely on several cryptographic primitives which should be familiar to anyone who has been involved with cryptographic literature.

The required definitions are taken from [BG12].

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

Schwartz-Zippel lemma allows to check the equality of two polynomials p_1 and p_2 of degrees d_1 and d_2. If $p_1 = p_2$ then $p_1 - p_2 = 0$ always. Otherwise, if $p_1 \neq p_2$ then for random $x_1, \dots, x_n \leftarrow \mathbb{Z}_q^*$, $p_1 - p_2 = 0$ with probability at most $\frac{\max(d_1, d_2)}{q-1}$.

2.1 Homomorphic public key cryptosystems

It is required to have a homomorphic public key encryption scheme with rerandomization possibility. In most cases, if the encryption scheme is homomorphic then the ciphertexts can also be rerandomized.

Definition 1. A public key cryptosystem $\mathfrak{C}$ is triple of algorithms Gen, $\mathcal{E}$ and $\mathcal{D}$.

Key generation algorithm Gen takes as inputs security parameter μ and randomness ρ and outputs a secret key sk and a corresponding public key pk.

Encryption algorithm $\mathcal{E}$ takes in a public key pk, plaintext M and randomness ρ and outputs corresponding ciphertext C.

Decryption algorithm $\mathcal{D}$ takes in a secret key sk, a ciphertext C and outputs corresponding plaintext m or abort symbol $\perp$ if the ciphertext is invalid.

For a pair $(\mathrm{pk}, \mathrm{sk})$ generated by the key generation algorithm, the cryptosystem is functional:

$$\forall m \in \mathcal{M}, \rho \in \Omega : \mathcal{D}(\mathrm{sk}, \mathcal{E}(\mathrm{pk}, m; \rho)) = m.$$

It is common to denote the output of the key generation algorithm as a keyed encryption and decryption algorithms $(\mathcal{E}_{\mathrm{pk}}, \mathcal{D}_{\mathrm{sk}})$. We follow this notation in this writing.

Definition 2. A cryptosystem $\mathfrak{C}$ is homomorphic cryptosystem if the encryption algorithm is homomorphic:

$$\forall m_1, m_2 \in \mathcal{M}, \rho_1, \rho_2 \in \Omega : \mathcal{E}_{\mathrm{pk}}(m_1 \odot m_2; \rho_1 + \rho_2) = \mathcal{E}_{\mathrm{pk}}(m_1; \rho_1)\mathcal{E}_{\mathrm{pk}}(m_2; \rho_2),$$

where $\odot$ is an operation in $\mathcal{M}$.

If $\odot$ corresponds to multiplication in $\mathcal{M}$, then we say that the cryptosystem is multiplicatively homomorphic. Otherwise, if $\odot$ corresponds to addition in $\mathcal{M}$ then we say that the cryptosystem is additively homomorphic.

The cryptosystem can also be both additively and multiplicatively homomorphic but we do not require this.

Example 1. *ElGamal cryptosystem is multiplicatively homomorphic.*

Let G be a group of order q with a generator $g \in G$.

The key generation function returns (g, x) as a secret key and $(g, g^x = y)$ as a public key. Encrypting m with randomness ρ returns $(g^\rho, y^\rho m)$. Decrypting a ciphertext (c_1, c_2) returns c_2/c_1^x.

Now:

$$\begin{aligned}\mathcal{E}_{\text{pk}}(m_1 m_2; \rho_1 + \rho_2) &= (g^{\rho_1+\rho_2}, y^{\rho_1+\rho_2} m_1 m_2)\\ &= (g^{\rho_1}, y^{\rho_1} m_1)(g^{\rho_2}, y^{\rho_2} m_2)\\ &= \mathcal{E}_{\text{pk}}(m_1; \rho_1)\mathcal{E}_{\text{pk}}(m_2; \rho_2)\end{aligned}$$

The encryption algorithm $\mathcal{E}_{\text{pk}}$ in Definition 1 was an algorithm using a randomness value ρ as an input. If it possible to change the randomness value of the ciphertext without decrypting, then we call the cryptosystem rerandomizable.

Example 2. *ElGamal cryptosystem is rerandomizable.*

This follows directly from the homomorphism:

$$\mathcal{E}_{\text{pk}}(m_1; \rho_1)\mathcal{E}_{\text{pk}}(1; \rho_2) = \mathcal{E}_{\text{pk}}(m_1; \rho_1 + \rho_2).$$

If $\rho_2 \in \Omega$ is uniformly sampled, then the new ciphertext $\mathcal{E}_{\text{pk}}(m_1; \rho_1 + \rho_2)$ is uniformly sampled from ciphertext space.

This means that if ρ_2 is chosen independently from the initial ciphertext then the new and old ciphertexts are independent. Thus, if ρ_2 is kept secret then the initial ciphertext is also unknown.

2.2 Homomorphic commitment schemes

Definition 3. A commitment scheme $\mathfrak{O}$ is triple of algorithms Gen, Com and Open.

Key generation algorithm Gen takes as inputs security parameter μ and randomness ρ and outputs a commitment key ck.

Commitment algorithm Com takes in a commitment key ck, randomness ρ and plaintext m and returns a commitment value c and a decommitment value d.

Opening algorithm takes in a commitment key ck, a commitment value c and a decommitment value d and returns the plaintext or abort symbol $\perp$ if the decommitment value does not correspond to the commitment value.

For a commitment key ck, the commitment scheme is functional:

$$\forall m \in \mathcal{M}, \rho \in \Omega : \text{Open}(\text{ck}, \text{Com}(\text{ck}, m; \rho)) = m.$$

It is common to denote the output of the key generation algorithm as a keyed commitment and opening algorithms $(\mathrm{Com}_{\mathrm{ck}}, \mathrm{Open}_{\mathrm{ck}})$. We also follow this notation in this writing.

Definition 4. A commitment scheme $\mathfrak{O}$ is homomorphic commitment scheme if the commitment algorithm is homomorphic[1]:

$$\forall m_1, m_2 \in \mathcal{M}, \rho_1, \rho_2 \in \Omega : \mathrm{Com}_{\mathrm{ck}}(m_1 \odot m_2; \rho_1 + \rho_2) = \mathrm{Com}_{\mathrm{ck}}(m_1; \rho_1)\mathrm{Com}_{\mathrm{ck}}(m_2; \rho_2).$$

Example 3. *Pedersen commitment scheme is additively homomorphic.*

The key generation function returns (g, y) as a commitment key. Commiting to m with randomness ρ returns $(g^m y^\rho)$ as a commitment value and (m, ρ) as a decommitment value. The opening algorithm checks whether $c = g^m y^r$ and returns m on success and $\perp$ on failure.

Now:

$$\begin{aligned}\mathrm{Com}_{\mathrm{ck}}(m_1 + m_2; \rho_1 + \rho_2) &= (g^{m_1+m_2} y^{\rho_1+\rho_2}) \\ &= (g^{m_1} y^{\rho_1})(g^{m_2} y^{\rho_2}) \\ &= \mathrm{Com}_{\mathrm{ck}}(m_1; \rho_1)\mathrm{Com}_{\mathrm{ck}}(m_2; \rho_2).\end{aligned}$$

Using homomorphism it is possible to commit to several values and to show that these values have some property without revealing the values themselves.

2.3 Zero-knowledge proofs

Informally, the framework of zero-knowledge proofs gives the prover $\mathcal{P}$ the ability to prove to verifier $\mathcal{V}$ that it knows some secret without revealing it (*proof of knowledge*) or access to some oracle (*proof of possession*), e.g. smart card.

Zero-knowledge proofs are necessary for constructing mix-nets as they allow to prove that the shuffling function is a permutation.

Let σ be some arbitrary side-information all parties possess during interaction. For example, if a cryptosystem and a commitment scheme is used in the protocol, then $\sigma = (\mathrm{pk}, \mathrm{ck})$. The statement x is the claim the prover is trying to prove. The witness w is some fact that backs the claim. The set R is the collection of triples (σ, x, w) such that corresponding witness w would prove the statement x.

The language L_σ is the set of all statements x that have a witness w for the relation R. Formally:

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

We denote the transcript between $\mathcal{P}$ and $\mathcal{V}$ on inputs s and t, respectively, as $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle$. We denote the acceptance or rejection as $b = \langle \mathcal{P}(s), \mathcal{V}(t) \rangle$, where $b = 0$ denotes rejection and $b = 1$ denotes acceptance.

We start defining zero-knowledge proofs by describing the requirements.

[1] Note that in this definition and in the following example we consider only the commitment value as the output of the commitment algorithm. This is just for notational convenience and the homomorphism of opening algorithm follows from this homomorphism

Firstly, the protocol needs to be *complete*, meaning that honest verifier should accept honest prover's claim. Formally, for any polynomial-time adversary $\mathcal{A}$:

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

Secondly, the protocol needs to be *sound*, meaning that if honest verifier accepts the claim, then it actually holds. Formally, for any polynomial-time adversary $\mathcal{A}$:

$$\Pr[\sigma \leftarrow \mathrm{Gen}(1^\lambda); x \leftarrow \mathcal{A}(\sigma) : x \notin L_\sigma \text{ and } \langle \mathcal{A}, \mathcal{V}(\sigma, x)\rangle = 1] < \mathrm{negl}(\lambda).$$

Thirdly, the protocol needs to be *simulatable*, meaning that there exists a simulator $\mathcal{S}$ which can simulate the transcript without access to the witness w. Having such a simulator provides zero-knowledge as the verifier could also interact with the simulator, who by the definition does not have access to witness w, so the verifier also learns nothing about w. Formally, there exists a polynomial-time simulator $\mathcal{S}$ for all polynomial-time adversaries $\mathcal{A}$ such that:

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

Note that in this formal definition the adversary can append additional information ρ to the verifier $\mathcal{V}$. But this is not a contradiction, as the verifier can itself modify any auxiliary information it has.

Definition 5. A triple $(\mathrm{Gen}, \mathcal{P}, \mathcal{V})$ is called *perfect special honest verifier zero knowledge*(SHVZK) argument for R with common reference string generator Gen if the protocol is perfectly complete, computationally sound and simulatable.

In the definition, perfect completeness assures that an honest claim is always accepted. Statistically, there are no false rejects for honest prover and verifier. Computational soundness restricts the malicious provers to have an upper limit on its computational performance.

This definition is still not complete. Firstly, it does not cover the case of dishonest verifiers. This could be enforced by using commitments schemes - the verifier commits to its random challenge before seeing the messages from the prover.

Secondly, the definition does not imply the possession of the secret. If the prover would possess the secret, then it could be extracted. A weaker definition just requires the existence of such extractor.

Definition 6. An argument $(\mathrm{Gen}, \mathcal{P}, \mathcal{V})$ has witness-extended emulation, if for all polynomial time deterministic $\mathcal{P}^*$ there exists an expected polynomial-time emulator χ such that for all polynomial-time adversaries $\mathcal{A}$ we have

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

This definition requires that there exists an extractor χ, which produces similar argument to the initial argument and is able to extract the witness used by the prover.

The definitions of simulatability and witness-extended emulation are not contradicting as in the latter case the emulator has access to additional information ρ which helps it to reach the goal.

Currently, the prover $\mathcal{P}$ must not obtain the challenges sent by the verifier $\mathcal{V}$ before it has sent its messages. This setting is inconvinient as it forces an interactive communication between each verifier and the prover.

In the non-interactive proofs, the challenges are generated by the prover and it publishes the transcript. The verifiers check the transcript and see if the responses to the challenges are correct. However, if the prover can choose the challenges at its will, then the previous definitions do not hold. The solution is to generate the challenges as an output of some hash function applied to some value (for example, applying the hash function to the statement x). Such construction of non-interactive zero-knowledge proofs from interactive zero-knowledge proofs is called a Fiat-Shamir heuristic.

In the random oracle model, the Fiat-Shamir heuristic allows to obtain same security assumptions as in interactive proofs. Proving the security assumptions using a practical hash function (with short description) may be difficult or even impossible.

3 Notation

We denote the number of ciphertexts as $N = mn$. The choice of n and m are covered later when we look at the storage and computation requirements of the protocol.

For the vectors $\vec{x}$ and $\vec{y}$ we write the point-wise product as $\vec{x}\vec{y} = (x_1y_1, \ldots, x_ny_n)$. Similarly, we write $\vec{x}^k = (x_1^k, \ldots, x_n^k)$. Applying the permutation π to vector $\vec{x}$ results in a permuted vector $\vec{x}_\pi = (x_{\pi(1)}, \ldots, x_{\pi(n)})$.

For the vectors of plaintexts $\vec{M} = (M_1, \ldots, M_n)$ and randomnesses $\vec{\rho} = (\rho_1, \ldots, \rho_n)$ we write $\mathcal{E}_{\mathrm{pk}}(\vec{M}; \vec{\rho}) = (\mathcal{E}_{\mathrm{pk}}(M_1; \rho_1), \ldots, \mathcal{E}_{\mathrm{pk}}(M_n; \rho_n))$. For the vectors of ciphertexts $\vec{C} = (C_1, \ldots, C_n)$ and integers $\vec{a} = (a_1, \ldots, a_n)$ we write $\vec{C}^{\vec{a}} = \prod_{i=1}^n C_i^{a_i}$. For a matrix $A \in \mathbb{Z}^{n\times m}$ which is represented by column vectors $\vec{a}_1, \ldots, \vec{a}_m$ we write $\vec{C}^A = (\vec{C}^{\vec{a}_1}, \ldots, \vec{C}^{\vec{a}_m})$.

We use a generalization of Pedersen commitment scheme which allows to commit to several values. The key generator returns $(g_1, \ldots, g_n, y) \subset \mathbb{G}^{n+1}$ and the commitment function returns $\mathrm{Com}_{\mathrm{ck}}(a_1, \ldots, a_n; r) = y^r \prod_{i=1}^n g_i^{a_i}$ if commiting to values $a_1, \ldots, a_n$ with randomness r. For the vectors of commitments $\vec{c}$ and integers $\vec{b}$ we write $\vec{c}^{\vec{b}} = \prod_{j=1}^m c_j^{b_j}$. For a matrix B with column vectors $\vec{b}_1, \ldots, \vec{b}_n$ we write $\vec{c}^B = (\vec{c}_1^{\vec{b}_1}, \ldots, \vec{c}_n^{\vec{b}_n})$. For a matrix $A \in \mathbb{Z}^{n\times m}$ we define $\mathrm{Com}_{\mathrm{ck}}(A; \vec{r}) = (\mathrm{Com}_{\mathrm{ck}}(\vec{a}_1; r_1), \ldots, \mathrm{Com}_{\mathrm{ck}}(\vec{a}_n; r_n))$. Also, if the column vectors of A are concatenated to form a new vector, $\vec{a} = \vec{a}_1 | \ldots | \vec{a}_n$, then $\mathrm{Com}_{\mathrm{ck}}(\vec{a}; \vec{r}) = \mathrm{Com}_{\mathrm{ck}}(A, \vec{r})$.

The set of off all permutations of input and output size N is denoted by Σ_N.

4 The Bayer-Groth shuffle protocol

In [Gro09], an argument was given in connection to linear algebra over finite field. The argument could be used for shuffling elements of a finite fields. Bayer and Groth modified

[BG12] this argument to be used with group elements. As some of the cryptosystems work over groups then this also applies to ciphertexts. They modified the argument such that the computation is done in the exponent of the generator. This approach, however, increased the number of exponentiations. To overcome this, several techniques could be used to decrease the overall computation. For example, they used Toom-Cook technique for polynomial multiplication if the number of ciphertexts is small and Fast Fourier Transformation otherwise to reduce the amount of computation.

We will describe the protocol at a higher level without too much focus on the computational improvements, as they are very technical. Those who are interested, can refer to their article [BG12, Section 4].

Formally, the proof of knowledge for ciphertexts $\vec{C}, \vec{C'}$ is a statement that the prover knows $\vec{\rho} \in \mathbb{Z}_q^N$ and $\pi \in \Sigma_N$ such that $\vec{C'} = \mathcal{E}_{\mathrm{pk}}(\vec{1}; \vec{\rho})\vec{C}_\pi$. The argument uses multi-exponentiation and product argument as sub-arguments. Informally, this means that the prover knows a permutation and randomness values which map initial ciphertexts to new ciphertexts.

The product argument is a proof of knowledge for inputs $\vec{c}_A \in \mathbb{G}^m, b \in \mathbb{Z}_q$ that the prover knows $A \in \mathbb{Z}^{n \times m}$ and $\vec{r} \in \mathbb{Z}_q^m$ such that $\vec{c}_A = \mathrm{Com}_{\mathrm{ck}}(A; \vec{r})$ and $\prod\limits_{i=1}^{m} \prod\limits_{j=1}^{m} a_{ij} = b$.

Firstly, the prover commits to the values $\pi(1), \dots, \pi(N)$. Then, he receives the challenge x from the verifier and then commits to $x^{\pi(1)}, \dots, x^{\pi(N)}$. The prover then receives challenges y, z and calculates commitments to (using homomorphic properties of the commitment scheme) $d_i - z = y\pi(i) + x^{\pi(i)} - z$ for all $i = 1, \dots, N$. As the verifier can compute $\prod_{i=1}^{N}(yi + x^i - z)$, then the prover uses the product argument to show that $\prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(yi + x^i - z)$. Over randomly chosen z, Schwartz-Zippel lemma states that the probability that the argument holds is $N/(q-1)$. If q chosen to be larger than N by a polynomial factor, then with negligible probability $d_i = yi + x^i$.

The multi-exponentiation argument is a proof of knowledge for inputs $\vec{C}_1, \dots, \vec{C}_m \in \mathbb{H}^n, C \in H, \vec{c}_A \in \mathbb{G}^m$ that the prover knows $A = \{a_j\}_{j=1}^m \in \mathbb{Z}_q^{n \times m}$ and $\vec{r} \in \mathbb{Z}_q^m, \exists \rho \in \mathbb{Z}_q$ such that $C = \mathcal{E}_{\mathrm{pk}}(1; \rho) \prod\limits_{i=1}^{m} \vec{C}_i^{\vec{a}_i}$ and $\vec{c}_A = \mathrm{Com}_{\mathrm{ck}}(A; \rho r)$. The multi-exponentiation argument is used to prove that the rerandomization of ciphertexts is correct.

Here, $\{C_i\}_{i=1}^N$ are the initial ciphertexts and $\{C'_i\}_{i=1}^N$ are the permuted ciphertexts. For the permutation $x^{\pi(1)}, \dots, x^{\pi(N)}$, the prover can show that

$$\prod_{i=1}^{N} C_i^{x^i} = \prod_{i=1}^{N} (C'_i)^{x^{\pi(i)}}$$

and $\vec{c}_B = \mathrm{Com}_{\mathrm{ck}}(\vec{b}; \vec{s})$. As the encryption scheme is homomorphic, then also

$$\prod_{i=1}^{N} M_i = \prod_{i=1}^{N} (M'_i)^{x^{\pi(i)}}.$$

After taking the discrete logarithm, we have

$$\sum_{i=1}^{N} \log(M_i) x^i = \sum_{i=1}^{N} \log(M'_i) x^{\pi(i)}.$$

We can rewrite the right side as

$$\sum_{i=1}^{N} \log(M'_i) x^{\pi(i)} = \sum_{i=1}^{N} \log(M'_{\pi^{-1}(i)}) x^i.$$

Thus

$$\sum_{i=1}^{N} \log(M_i) x^i = \sum_{i=1}^{N} \log(M_{\pi^{-1}(i)}) x^i.$$

We can apply Schwartz-Zippel lemma and thus $\log(M_i) = \log(M'_{\pi^{-1}(i)})$ with probability greater than $1 - \frac{N}{q-1}$. Thus also $M'_1 = M_{\pi^{-1}(1)}, \ldots, M'_N = M_{\pi^{-1}(N)}$. This implies that the rerandomization is correct.

The corresponding protocol is illustrated in Figure 4

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

Proof idea. It can be shown that the product argument and the multi-exponentiation argument are perfect SHVZK.

For completeness, let $d_i = y\pi(i) + x^{\pi(i)}$. Then

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

Also, if $C'_i = \mathcal{E}_{\text{pk}}(1; \rho_i) C_{\pi(i)}$, then

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

Because the sub-arguments are perfectly complete, then the full argument is also perfectly complete.

The soundness of the argument was described in the protocol description. If we consider that the sub-arguments are sound, then the whole argument is also sound.

The simulator picks random $\vec{r}, \vec{s} \leftarrow \mathbb{Z}_q^m$ and sets commitments $\vec{c}_A = \text{Com}_{\text{ck}}(\vec{0}; \vec{r})$ and $\vec{c}_B = \text{Com}_{\text{ck}}(\vec{0}; \vec{s})$. It then runs the simulators of the product argument and multi-exponentiation argument.

Using the witness-extended emulation of the sub-arguments, the emulator can extract the openings for $\vec{c}_B$ and $\vec{c}_D$. From $\vec{c}_D = \vec{c}_A^{\,y} \vec{c}_B$ the emulator can compute the openings $\vec{a}, \vec{r}$ for $\vec{c}_A$. The openings $\vec{a}$ correspond to the permutation $\vec{a} = \{\pi(i)\}_{i=1}^{N}$ and the emulator has extracted the permutation.

If the execution is rewinded for N times for different x, then for each x_j the multi-exponentiation emulator returns a witness containing $\rho^{(j)}$ such that

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

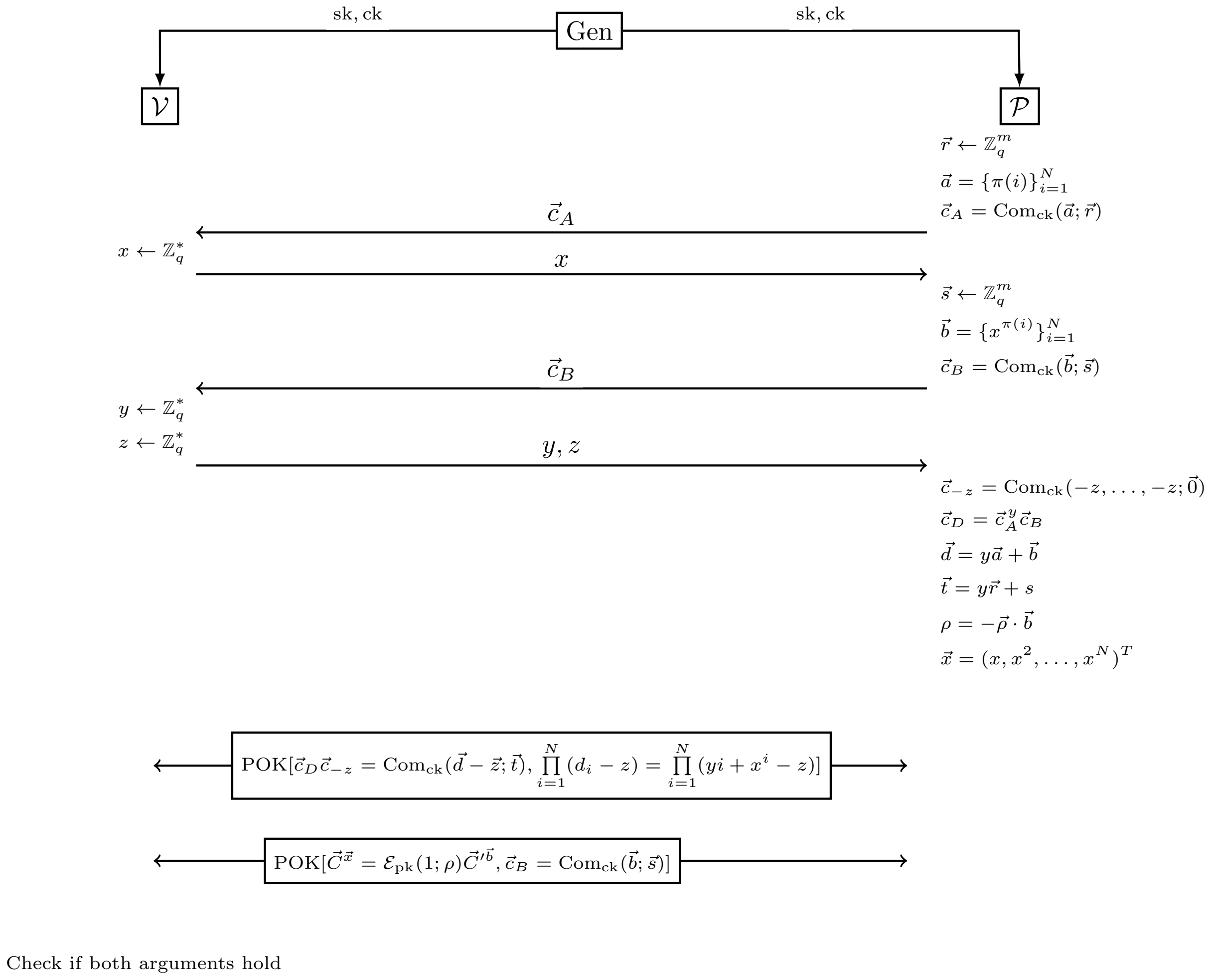
sk, ck
Gen
sk, ck
$\mathcal{V}$
$\mathcal{P}$
$\vec{r} \leftarrow \mathbb{Z}_q^m$
$\vec{a} = \{\pi(i)\}_{i=1}^N$
$\vec{c}_A = \mathrm{Com}_{\mathrm{ck}}(\vec{a}; \vec{r})$
$\vec{c}_A$
$x \leftarrow \mathbb{Z}_q^*$
x
$\vec{s} \leftarrow \mathbb{Z}_q^m$
$\vec{b} = \{x^{\pi(i)}\}_{i=1}^N$
$\vec{c}_B = \mathrm{Com}_{\mathrm{ck}}(\vec{b}; \vec{s})$
$\vec{c}_B$
$y \leftarrow \mathbb{Z}_q^*$
$z \leftarrow \mathbb{Z}_q^*$
y, z
$\vec{c}_{-z} = \mathrm{Com}_{\mathrm{ck}}(-z, \ldots, -z; \vec{0})$
$\vec{c}_D = \vec{c}_A^{\,y} \vec{c}_B$
$\vec{d} = y\vec{a} + \vec{b}$
$\vec{t} = y\vec{r} + s$
$\rho = -\vec{\rho} \cdot \vec{b}$
$\vec{x} = (x, x^2, \ldots, x^N)^T$
POK[$\vec{c}_D \vec{c}_{-z} = \mathrm{Com}_{\mathrm{ck}}(\vec{d} - \vec{z}; \vec{t}), \prod_{i=1}^{N} (d_i - z) = \prod_{i=1}^{N} (yi + x^i - z)$]
POK[$\vec{C}^{\vec{x}} = \mathcal{E}_{\mathrm{pk}}(1; \rho) \vec{C}'^{\vec{b}}, \vec{c}_B = \mathrm{Com}_{\mathrm{ck}}(\vec{b}; \vec{s})$]
Check if both arguments hold

Figure 1: The transmitted messages in the Groth-Bayer shuffle protocol

If $x_1, \ldots, x_N$ are different, then the matrix

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

is invertible and

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

The emulator has recovered the randomness $\vec{\rho}' = (-\vec{\rho} X^{-1})$.

4.1 Computational complexity and proof size of the Bayer-Groth shuffle protocol

Let $N = nm$. Theoretically, the prover needs to do $2\log(m)N$ exponentiations and the verifier needs to do $4N$ exponentiations. Compared to previous results, the prover's computation is larger. For example, in Verificatum the prover needs to do $9N$ exponentiations.

The size of the proof is $11m$ elements of group $\mathbb{G}$ and $5n$ elements of field $\mathbb{Z}_q$. Compared to previous results, the proof size is smaller. For example, in Verificatum it is needed to have $3N$ elements of group $\mathbb{G}$ and $4N$ elements of field $\mathbb{Z}_q$.

As m can be chosen freely, then the prover can control the size of the proof and the computation complexity. If $m \to N$, then the computational complexity and the size of the proof increases. However, if $m \to 1$, then the computational complexity decreases while the size of the proof increases. For obtaining the smallest proof, m should be taken $m \approx \sqrt{N}$.

In [BG12], the authors gave several optimization methods. The efficiency comparison using different methods is illustrated in the following table.

	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	
$m = 64$	Multi-expo	615	594	21	0.7 MB
	FFT	328	307	20	
	Toom-Cook	128	91	18	

Table 1: Run time of Bayer-Groth shuffle argument for $N = 100000$ on a Core2Duo 2.53 GHz, 3MB L2 cache, 4 GB ram computer

5 Conclusion

We surveyed the shuffle protocol constructed by Bayer and Groth. This shuffle has shorter proofs of correctness than previous shuffle constructions.

Using a mix-net in practice can lead to new and non-mathematical problems. If we consider the i-voting protocol, then there should be several parties mixing the votes. Depending on political choices and legislation this could be prohibited, as this means handing over the ballots to some external party.

The additional security guarantee that a mix-net could provide, is privacy of the voter. It obfuscates the connection between input ciphertexts and returned ciphertexts so the adversary could not distinguish encrypted ballots.

Because the shuffle does not have any external information, then none of the initial security guarantees are sacrificed if a mix-net is used. If the constraints allow, a mix-net can be used within an organization to increase potential adversary's work.

However, the mix-nets require homomorphism from the used cryptosystem as the proofs rely on this property. It could mean that modifications are required for the whole system.

In conclusion, mix-nets could provide additional security in the i-voting protocol but its use and possible consequences should be thoroughly studied. There may be a need to change legislation to allow the use of mix-nets and possible the underlying cryptosystem should be changed. Furthermore, it is important to educate the voters of the construction and inner workings of the proof.

References

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer Berlin Heidelberg, 2012.

[Com13a] The National Electoral Committee. Elektroonilise hääletamise süsteemi üldkirjeldus (Description of electronic voting system). 2004-2013. http://www.vvk.ee/public/dok/elektroonilise-haaletamise-systeemi-yldkirjeldus-EH-03-03-1_2013.pdf.

[Com13b] The National Electoral Committee. Kohaliku omavalitsuse volikogu valimiste e-hääletamise protseduuride hindamise vahearuanne II (The second interim report of local municipalities e-voting procedures audit). 2013. https://www.valimised.ee/vahearuanne_2_2013.ddoc.

[Gro09] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 192–208. Springer Berlin Heidelberg, 2009.

[GST13] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. Cryptology ePrint Archive, Report 2013/857, 2013. http://eprint.iacr.org/.

[Hua13] Andrew Huang. On hacking MicroSD cards, December 2013. http://www.bunniestudios.com/blog/?p=3554.

[Meh14] Neel Mehta. The Heartbleed bug, 2014. http://heartbleed.com/.