New Receipt-Free E-Voting Scheme and Self-Proving Mix Net as New Paradigm

Aram Jivanyan
Russian-Armenian University of Armenia
jivanyan@gmail.com and Gurgen Khachatryan
American University of Armenia
gurgenkh@aua.am

No Institute Given

Abstract. The contribution of this paper is twofold. First we present a new simple electronic voting scheme having standard re-encryption mix net back-end, which allows to cast a ballot and verify its correctness in a new way. Then we extend the proposed scheme to represent a new very efficient mix network construction. We called our mix network to be self-proving mix, because it is shown how mix process correctness can be verified without demanding from mix party a special proof. Our proposed mix network allows to reveal all the cheating occurred during a mix process at verification of decryption parties work.
Key words: E-voting, Mix Net, Receipt-Freeness, Self-Proving Mix.

1 Introduction

A number of end-to-end verifiable voting schemes have been introduced recently. These schemes aim to allow voters to verify that their votes have casted as they intended, and in addition allow anyone to verify that the tally has been computed correctly. These goals must be achieved while maintaining voter privacy and providing receipt-freeness. But early research presumed that each voter would own a trusted device to perform hard cryptographic operations on her behalf. Such assumption does not model real-world voters. In 2004, Chaum [4] and Neff [27] independently introduced mechanisms that enable an unaided human to cast an encrypted ballot without trusting the voting machine. Then, Moran and Naor [28] presented a scheme based on Neffs earlier technique, MarkPledge. However they leaved an open unsolved problem in their paper, regarding to prevention of covert channels in the ballot. Recently, Adida and Neff [29] proposed their solution named as MarkPledge2, which solved the above mentioned problem. MarkPledge2 was the first efficient, covert-channel-resistant, receipt-free ballot casting scheme that can be used by humans without having trusted device for performing hard cryptographic operations. In this paper we introduce another electronic voting scheme based on novel approach, which also is receipt-free and resistent to covert channels. The approach lying in the basis of our scheme is to allow each voter encrypt his vote by his secret key. It means that voter can at first choose several secret keys and demand the voting machine to encrypt all of them with joint public key of election trustees. Having all secrets encrypted, voter can randomly choose one secret key, which will be used for vote encryption. Then all remained keys can be already opened and voter can demand the voting machine to get special proofs, that the keys were encrypted correctly. That proves than can be checked out of voting booth. This approach allows to ensure with high probability that the vote was casted as intended without needing to do complex cryptographic operations at voting place. Vote encryption by secret key must be a simple operation which can be verified by any voter who have minimal computational ability. We use the most simplest encryption technique, namely XOR-ing. The attractiveness of XOR is that the operation can be visualized to make its verification possible by unaided humans, who have no computational ability. This voting scheme has standard mix net back-end. Verifiable mixing has substantial complexity for large scale elections. Although it is not impractical with today's hardware, however it is interesting to have faster verifiable mix nets. After exposing our ballot casting scheme we than extend it to come to the new mix network construction. Our mix network is neither decryption mix net, nor re-encryption. It is a hybrid approach which main contribution is that mix process can be verified without special verification process. The special proof and verification of correct mixing is the most expensive step in mix process. The elimination of that step yields to effective mix. It is shown the correctness of the mix process be verified during the ballot decryption verification.

2 Background

Some basic information about a number of cryptographic primitives necessary to understand the rest of the paper is provided here.

We start with a description of the ElGamal public-key cryptosystem [1], and discuss some of its properties, which makes this cryptosystem very useful for voting. Next we introduce mix networks.

2.1 ElGamal cryptosystem

Let P and Q be two large primes such that $P = 2Q + 1$. We denote by G_Q the subgroup of $\mathbb{Z}_P^*$ of order Q. All the subsequent arithmetic operations are performed in modulo P unless otherwise stated. Let g be a generator of G_Q. The private key is an element $x \in \mathbb{Z}_Q$, and the corresponding public key is $Y = g^x$. To encrypt a plaintext $m \in G_Q$, a random element $r \in \mathbb{Z}_Q$ is chosen and the ciphertext is computed as $\mathcal{E}_y(m, r) = (g^r, mY^r) = (G, M)$, so the ElGamal ciphertext is a pair of elements of G_Q. To The decryption $\mathcal{D}_x(G, M)$ of an ElGamal ciphertext (G, M) can be computed by the owner of secret x in a following way: $D_x(G, M) = M/G^x = m$.

ElGamal has a multiplicative homomorphic property. It means that given two ElGamal ciphertexts(under the same public key Y) $(G_1, M_1) = (g^{r_1}, m_1Y^{r_1})$ and $(G_2, M_2) = (g^{r_2}, m_2Y^{r_2})$ encrypting messages m_1 and m_2 respectively, than their product computed as $(G_1, M_1) \cdot (G_2, M_2) = (G_1 \cdot G_2, M_1 \cdot M_2) = (g^{r_1+r_2}, m_1 \cdot m_2 Y^{r_1+r_2})$ is an encryption of $m_1 \cdot m_2$.

Given an ElGamal encryption of m under public key Y, $(G, M) = (g^r, mY^r)$, this ciphertext can be transformed into a different ciphertext by choosing $r' \in \mathbb{Z}_P$ at random and computing $(G', M') = (G \cdot g^{r'}, M \cdot Y^{r'})$ which decrypts to the same m. Under the Decisional Diffie-Hellman assumption, a third party can not determine whether the plaintexts of (G, M) and (G', M') are the same, i.e. ElGamal cryptosystem is semantically secure. This fact lies in the basis of construction re-encryption mix networks.

In [3] a mechanism is presented that permits a Prover to prove in zero-knowledge way that, given a tuple (g, u, y, v), she knows the secret x satisfying $x = log_g y = log_u v$. The proof is as follows:

– The Prover chooses $s \in \mathbb{Z}_Q$ at random and computes the tuple $(a, b) = (g^s, u^s)$.
– The Verifier gives a challenge c to prover.
– The Prover computes $r = s + cx$ and sends (a,b,r) to the Verifier.

We will denote $CP(g, y, u, v)$ the tuple (a, b, r) sent by the Prover to the Verifier. For verifying the proof the Verifier checks whether $g^r = ay^c$ and $u^r = bv^c$. The non-interactive variant of this protocol can be constructed by computing the challenge c via Fiat-Shamir hash trick [2] - $c = \mathcal{H}(g, u, g^s, u^s)$, where $\mathcal{H}$ is a cryptographically secure hash function. Given a publicly known ciphertext (G,M) encrypted under public key Y, the owner of private key x can verifiably decrypt it by publishing the message m and $CP(g, G, Y, \frac{M}{m})$. A verifier that succeeds in checking $CP(g, G, Y, \frac{M}{m})$ is convinced that (G, M) is an encryption of m under public key Y.

2.2 Mix Networks

The concept of mix networks was first introduced by Chaum in [5]. Mix-net is a cryptography construction that enables one or more mix servers to take a sequence of encrypted input messages, re-encrypt or decrypt them, and output them in an unrevealed, randomly permuted order. In theory, if there is one honest mix server, the permutation links are kept secure. Mix networks can be divided into re-encryption mix-net and decryption mix-net. Chaum proposed decryption mix nets, where the inputs of mix network are ballots encrypted under each mix server's public key from the last mix server to the first. When decrypting, each mix server will first decrypt the input ballots by using her own secret key, then throws away the random value and outputs all the resulted messages in permuted order. Re-encryption mix networks were proposed in [13]. In re-encryption mix networks the original ballots are encrypted under the same ElGamal public key Y, which corresponding private key is distributed among trustees in a threshold scheme. In re-encryption mix-nets the re-encryption and decryption process are separated. Each mix server first re-encrypt a list of ElGamal encrypted ciphertexts $\{\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_N\}$ and output the resulted ciphertexts in randomly permuted order as another list of ElGamal encrypted ciphertexts $\{\mathcal{E}'_1, \mathcal{E}'_2, \cdots, \mathcal{E}'_N\}$, after which next mix or decryption parties can perform. When re-encryption phases finishes, the quorum of decryption parties jointly recover the ElGamal secret key x and decrypt all the ballots. In Section **4** we introduce a novel mix net construction, which is neither decryption mix net, nor re-encryption mix net. It is a hybrid approach. It requires a fixed number of decryption servers, but allows to have a flexible number of mix servers. All inputs of our mix net will have K tuples of ElGamal ciphertexts, where K is the number of decryption servers, and each decryption server owns one public/private key pair. Each ciphertext within a tuple is encrypted under one decryption server's public key via ElGamal cryptosystem. We called this self-proving mix, because it is shown how mix parties work can be automatically verified when decryption process is verified. This self-proving property eliminates the need of complex verification steps of mix process and results to efficient mix network.

3 Voting Scheme

We assume that voters vote at certified polling stations and there is one election race with M candidates. As in most election schemes, we also include the notion of "bulletin board" $\mathcal{BB}$ as a shared memory where all authorized parties have sequentially write access after authentication and any observer has a read access. At any moment it must be accessible both for writing and reading via election website. We identify the following players each with a fixed role he must play during election process:

– Polling Stations: For simplicity we assume the existence of only one Polling Station(PS) with one voting machine which can be malicious. We also assume that PS coordinates all the electoral process to not burden our scheme with additional players. When election begins, PS collects all votes and posts them to $\mathcal{BB}$. It is assumed that only legitimate voters will be allowed by PS to vote and they will be able to vote only once. PS owns certified public/private key pair for digitally sign each voter's receipt given by voting machine.

– Decryption parties: There are K decryption parties $\mathcal{D}_1$,$\mathcal{D}_2$, ..., $\mathcal{D}_K$, who owns the decryption keys(shares of decryption key) needed for ballots decryption. Each decryption party can post to the $\mathcal{BB}$.

– Mix parties: Mix party performs the mixing of encrypted votes. Any certified participant can act as mix party and there can be as many mix parties as needed. We will assume that at least one of these parties is honest.

– Voters: There are N voters $\mathcal{V}_1$, $\mathcal{V}_2$, ..., $\mathcal{V}_N$ who cast a vote. Each of them is able to read and compare small strings.

– Verifiers: A verifier can be everyone who is able to do more complex mathematical operations. Usually a verifier owns some trusted computer device and can do cryptographic operations outside of PS. A verifier can read the $\mathcal{BB}$ and any voter's receipt to check the validity of signature on it, ensure the correctness of vote encryption and also verify any proofs provided by decryption parties. Verifier can complain in case he detects any error.

The election process consists of the following phases:

1. Set up: At this phase the PS initializes the $\mathcal{BB}$, publishes the characteristics of group where all public keys used during election are defined and the description of a secure hash function $\mathcal{H}$ which is used at receipt preparation. All decryption servers and Polling Station publish their certified public keys. Also candidates names encoding is defined and announced to all parties.
2. Voting: At this phase PS authenticates each voter and checks if he/she has not voted already. Then each voter votes using voting machine according to the fixed protocol which will be described later and gets special receipt signed by PS, after which his/her ballot will is posted to $\mathcal{BB}$.
3. Vote mixing: Each mix party takes the encrypted ballots from the $\mathcal{BB}$ and re-randomizes them and posts the resulted ballots in shuffled order again to $\mathcal{BB}$ for next mix or decryption party to perform respective actions. Mix party can also be required to provide an additional data for verification purposes.
4. Vote decryption: At this phase the encrypted ballots are opened by decryption parties. Each decryption server except performing his suited actions also provides some additional data required for verification of its actions. The output of this final phase will be an anonymous list of casted ballots which should be already counted.

Next we present all the phases in more details.

3.1 Setup

At this phase Polling Station initializes the $\mathcal{BB}$ and publishes the parameters of group, where all public keys used for elections are defined. These are two large primes P and Q where $P = 2Q + 1$. All public keys are defined over the Q order subgroup G_Q of $\mathbb{Z}_P^*$. PS publishes its public key. The Polling Station's key pair is (x_{PS}, Y_{PS}). The decryption servers generate the election public key Y using threshold ElGamal [21], such that each decryption party has a secret share x_i. A cryptographically secure hash function $\mathcal{H}$ definition used in election is specified. For encoding candidates names we take $l = \lceil \log_2 M \rceil$ where M is the number of candidates. For simplicity we assume that $M = 2^l$. Each candidate is uniquely assigned a binary vector code $C \in \{0, 1\}^l$ and the candidate name-code pairs are published. Each candidate code must be also conformed with one G_Q element. For that reason we take the first M prime numbers(included 1) and conform each code to one prime in the way shown in Table 1:

00..000	1
00..001	2
00..010	3
00..011	5
00..100	7
....	..
11..111	p_M

Table 1: Keys encoding by prime numbers

3.2 Voting

We do not discuss here the authentication phase of voters and vote unicity checks done by PS. Instead we detail how an authenticated and first time voting voter $\mathcal{V}_i$ casts here vote V_i. Each $\mathcal{V}_i$ is given an electronic ballot with unique ID. Next we detail voting phase steps:

$\underbrace{BallotCastingProtocol}$

1. The voting machine prepares

   (a) 3 random values $\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_{CP} \in \mathbb{Z}_Q$ needed for vote encryption.

   (b) Computes and displays(also prints on receipt) $\mathrm{H}_1 = \mathcal{H}(g^{\boldsymbol{r}_1} \| g^{\boldsymbol{r}_2} \| g^{\boldsymbol{r}_{CP}})$, where $\|$ means concatenation.

   Note that the random values are selected and the hash of their commitments is printed on the receipt before any information is provided by voter.

2. The voter inputs 2 random l bit vectors after seeing H_1 printed already on the receipt. We denote the keys by key_1 and key_2. It is assumed that all voters can provide a short random inputs of size l.

3. The Voting machine

   (a) encrypts voter inputed keys key_1 and key_2 via ElGamal cryptosystem by using the predefined random values $\boldsymbol{r}_1$, $\boldsymbol{r}_2$: $\mathcal{E}_1(key_1, r_1) = (G_1, M_1) = (g^{\boldsymbol{r}_1}, p_1 \cdot Y^{\boldsymbol{r}_1})$, $\mathcal{E}_2(key_2, r_2) = (G_2, M_2) = (g^{\boldsymbol{r}_2}, p_2 \cdot Y^{\boldsymbol{r}_2})$
   Here p_1 and p_2 stands as corresponding primes of key_1 and key_2 according to Table 1.

   (b) adds $\mathcal{E}_1$ and $\mathcal{E}_2$ to the ballot.

   (c) computes $\mathrm{H}_2 = \mathcal{H}(M_1 \| M_2)$ and prints it on the receipt.

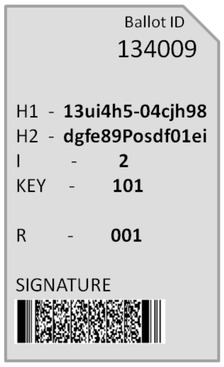
4. After seeing H_2 printed on the receipt the voter challenges one of the encryptions by choosing a random index $i \in [1, 2]$, and the voting machine prints on the receipt the index i and the key_i and adds to the ballot the index i and a non-interactive Chaum-Pedersen proof $CP = (g^{r_{CP}}, Y^{r_{CP}}, z)$ that $\mathcal{E}_i$ is really a correct encryption of key_i.

5. Voter selects his candidate, whose l bit binary code C is publicly known and visible on voting machine interface. $C \in [0..00, 0..01, 0..10, ..., 1..11]$

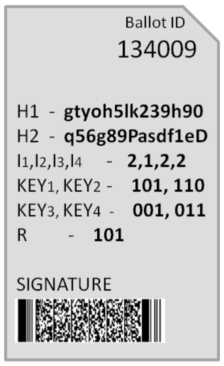
6. Voting machine
   (a) computs $R = key_{\bar{i}} \oplus C$ and displays it, where $\bar{i}$ is the index not selected by voter at previous step.
   (b) adds R field on the ballot and also prints it on the voter receipt.

7. If the voter confirms his/her vote than voting machine signs both the ballot and receipt by Polling Station's private key and gives receipt to voter.

After voting the ballot of $\mathcal{V}_i$ looks like V_i=$\{ID, R, i, CP, \mathcal{E}_1, \mathcal{E}_2, S_{PS}(ID, R, \mathcal{CP}, \mathcal{E}_1, \mathcal{E}_2)\}$, where $S_{PS}(ID, R, \mathcal{CP}, \mathcal{E}_1, \mathcal{E}_2)$ is the signature of Polling Station. The receipt given to voter remains quite simple: $Receipt = \{ID, H_1, H_2, R, i, key_i, S_{PS}(ID, H_1, H_2, R\ , i, key_i)\}$. Assuming that we have 8 candidates, which means all keys are 3 bit length, then a typical receipt is shown in Figure 1a.


Ballot ID
134009
H1 - **13ui4h5-04cjh98**
H2 - **dgfe89Posdf01ei**
I - **2**
KEY - **101**
R - **001**
SIGNATURE

(a) Joint key


Ballot ID
134009
H1 - **gtyoh5lk239h90**
H2 - **q56g89Pasdf1eD**
I1,I2,I3,I4 - **2,1,2,2**
KEY1, KEY2 - **101, 110**
KEY3, KEY4 - **001, 011**
R - **101**
SIGNATURE

(b) Distinct keys(4 keys)

Fig. 1

Having the receipt the voter can check the validity and correctness of his vote with help of verifier outside of Polling Station, but two checks must be done by voter before he leaves PS:

1. Voter must verify the correctness of R field printed on receipt. As we have already mentioned, the XOR operation used for computing R from voter chosen candidate code and voter entered secret key, can be visualized by methods presented in [24, 25], as it was firstly used by Chaum in [4] to build one of the first encrypted voter receipt techniques. Note that not all voters are required to verify the correctness of R. A malicious voting machine will be discovered even when a small percent of voters do it. Note that if we have a boolean vector $\mathbb{X} = (X_1, X_2, ...X_m)$, then the XOR of X_i can be computed just as

   $$\bigoplus_{i=1}^{m} X_i = \begin{cases} 1 & \text{if the number of 1 in vector is odd} \\ 0 & \text{otherwise} \end{cases}$$

   So even without visualization techniques, a small percent of all voters will have enough computational ability to do XOR on behalf. Moreover, there can be some auditors able to compute XOR of small keys, who can from time to time audit the voting machine.
2. Voter must check that the code printed on the receipt among with its index match the one he marked to be opened.

If these two checks are successfully accepted by voter it guaranties that any other possible cheating done against ballot validness will be discovered with very high probability by verifier outside of Polling Station via more complicated consistency checks. These checks are as follows:

– Check of signatures validity on receipt and ballot.
– Check of H_1 and H_2 validness: There are 2 ElGamal ciphertexts and 1 Chaum-Pedersen proof. Verifier must concatenate together first components of both ciphertexts and Chaum-Pedersen proof in fixed order as it was done by voting machine, and compute hash of the resulted message by $\mathcal{H}$. The result must be equal to H_1 printed on the voter receipt. At the same way the hash of the concatenation of second parts of ElGamal ciphertexts must be computed, which must be equal to H_2. The first check ensures that all predefined random values are used for keys encryption and Chaum-Pedersen proofs, the second check ensures that ciphertexts are not modified after voter has chosen the key subject to disclosure.
– Check of Chaum-Pedersen Proof: There is one pair like $[key_i,i]$ printed on the receipt. The verifier must take the i-th ciphertext of ballot $E_i = (g^{r_i}, p_i \cdot Y^{r_i})$ where $i \in [1, 2]$, and verify with the Chaum-Pedersen proof that $(g, Y, g^{r_i}, p_i \cdot Y^{r_i}/p_{key_i})$ is DDH tuple. Here p_{key_i} is the corresponding prime of key_1 from Table 1.

As one of two keys randomly chosen by voter is opened after both 2 keys have been encrypted and a Chaum-Pedersen proof is provided ensuring its correct encryption, than any cheating done during voter's keys encryption will be discovered by the third check with 1/2 probability for each ballot. We assume that 1/2 soundness is sufficient for large scale elections, where a lot of ballots will be verified. Note that the probability can be increased by requiring to voter enter for example 4 keys and then reveal 3 of them.

The first step in our protocol has significant importance. The random values used for keys encryption and Chaum-Pedersen proofs can be used by malicious computer to covertly encode some information about ballot. When voting machine knows the secrets he needs to encrypt, it can chose random values in a special way so that the last few bits of ciphertexts could reveal the secret. To prevent the possibility of covertly leaking any information it is required that voting machine has to pre-generate all random values and provides a proof that he indeed used the chosen values. That is why computer first computes and shows to voter the H_1 and prints it on the receipt after which all other information is provided by voter. Having the H_1 and also the Chaum-Pedersen proofs for half of encrypted keys any verifier can check and be sure that computer indeed have used the predefined random values and so ciphertexts do not reveal any information related to voter preference. This steps make our voting scheme resistent to covert channels. Receipt-freeness comes naturally once we have covert-channel resistance and show that our scheme is coercion-resistant. Let us detail the coercion-resistance strategy of voter. The Voter's only interactions with the voting machine are:

1. Selection of 2 secret keys.
2. Selection of random index from [1,2].
3. Selection of a candidate code.

Coercer can instructs the voter to input special keys key'_1 and key'_2 and choose candidate C'. The R code, which the coercer is expected to see is R = $C' \oplus key'_1 \oplus key'_2$. Coercer can also say to voter which key he must open. Let us assume that he want the voter to open first key. Thus, the Voter's coercion-resistance strategy is very simple: He input keys key'_1 and key_2 and choose his preferred candidate, which code is C. The voter's entered key_2 is computed in such way, that the resulted R will be exactly what the coercer expects to see. It means $key_2 = key'_2 \oplus C \oplus C'$.

3.3 Vote mixing and decryption

When the voting phase finishes and all votes are collected, the $\mathcal{BB}$ contains ballots set composed of N votes where for each $i \in [1, ..., N]$ $V_i = (ID, R, i, CP, \mathcal{E}_1, \mathcal{E}_2, S_{PS}(ID, R, \mathcal{CP}, \mathcal{E}_1, \mathcal{E}_2))$. Before mix process begins the set of casted ballots, which are published on $\mathcal{BB}$, is taken and everything is erased from ballots except the R field and $\mathcal{E}_{\bar{i}}$ ciphertext. If $i = 1$ then $\bar{i} = 2$ and vice versa. Then a series of t mix operations is performed, which results to an anonymous ballot set. Each mix operation re-encrypts and shuffles all ballots. Note that the R field of ballots is not remasked during re-encryption and somebody can argue that it will uncover certain ballot after shuffling. But as R takes value from small $M = 2^l$ length range, where $M << N$, each ballot after shuffling will be hidden among

N/M ballots. For large scale elections this ensures enough privacy. For example if there are 64000 voters and the race consists of 8 candidates, then each vote will be hidden among 8000 ballots.

Each mix party is also required to provide an additional data to prove that he acted correctly. It means that no ballot was modified or deleted and replaced by another one during ballots re-encryption process. Most of existing verification protocols can be used here as well as the fastest and most secure protocols due to Neff [8] and Furukawa-Sako [9] to provide such proofs. In the next section we will extend our protocol in such way that it will yield to novel type mix construction with lower computational complexity in verifiable mixing.

4 Self-Proving Mix Net Design

The intuition behind verification of mix network process directly comes after the re-encryption mix net construction and is as follows: each server must prove, that his received and outputted sets are the same. If each server succeeds to do this, then it follows that the final set would correspond to the preliminary set. Note that while proving this, mix server must keep his permutation secret. This fact complicates the situation and makes the verification proofs not trivial. The verification techniques which provides the full spectrum of privacy, requires number of modular exponentiations linear proportional to the number of inputs. Our e-voting scheme allows to show a different approach to mix process verification. The underlying idea of our suggested mechanism is as follows: instead of each re-encryption server proving that he has operated correctly, which has seemed to be the only way to reveal any cheating done by mix servers, the decryption servers reveal and expose any cheating done by mix servers if they are some. In our scheme malicious mix party can falsify the votes in two different ways. First he can change votes in targeted way by deleting votes and inserting others which belongs to his preferred candidate. Second he can change votes in "blind" way without knowing whom the changed votes will go. This is a realistic attack, which aims to disfigure election results. We design our scheme in such way, that targeted attacks are revealed immediately after malicious mix party performs. For discovering "blind" attacks done by mix parties we need to add one extra element to each ballot we called "check" element. The "check" element allows to discover at the end of mix process all ballots which were "blindly" falsified. If any such problematic ballot is encountered, the mistake is traced by asking each mix party to reveal how it shuffled and re-encrypted that faulty ballot. The cheating parties are removed and the process must be restarted. For simplifying an exposition, we will assume that there are the same number mix and decryption parties. We require each decryption party to follow one mix party as is shown in Figure 2.

Before exposing the new mix net construction, we need to extend presented voting protocol. One of the key things lying in the basis of our mix net is the requirement of each decryption server to have its own public/private key pair instead of common key's share. If there are K decryption servers, than we denote

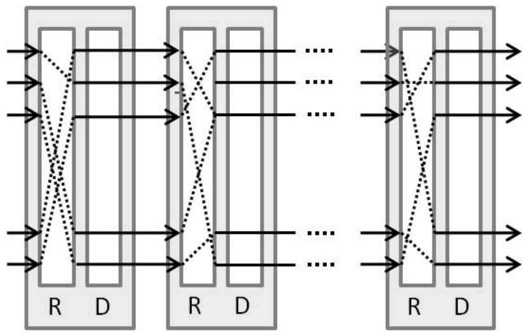


Fig. 2: Combined ReEncryption-Decryption Mix.

by Y_j the j-th server's public key. The voting protocol is modified as is shown bellow:

$\underbrace{ExtentedBallotCastingProtocol}$

1. The voting machine prepares

   (a) 3K+2 random values $\boldsymbol{r}_1^1, \boldsymbol{r}_1^2, ..., \boldsymbol{r}_K^1, \boldsymbol{r}_K^2,\ \boldsymbol{r}_{CP_1}, ..., \boldsymbol{r}_{CP_K},\ \boldsymbol{r}_0,\ \boldsymbol{r}_P \in \mathbb{Z}_Q$ needed for the vote encryption.

   (b) Computes and displays(also prints on receipt) $\mathrm{H}_1 = \mathcal{H}(g^{\boldsymbol{r}_1} \| g^{\boldsymbol{r}_2} ... \| g^{\boldsymbol{r}_{3K+2}})$, where we indexed random values sequentially to simplify exposition.

2. The voter inputs 2K random l bit vectors after seeing H_1 printed already on the receipt. We denote the keys by $[key_{\mathbf{1}}^1,\ key_{\mathbf{1}}^2], \cdots, [key_{\mathbf{K}}^1, key_{\mathbf{K}}^2]$.

3. The Voting machine

   (a) encrypts voter inputed first two keys $key_{\mathbf{1}}^1$ and $key_{\mathbf{1}}^2$ via first decryption server's public key Y_1 by using random values $\boldsymbol{r}_1^1$ and $\boldsymbol{r}_1^2$: $\mathcal{E}_1^1(key_{\mathbf{1}}^1, \boldsymbol{r}_1^1) = (G_1^1, M_1^1) = (g^{\boldsymbol{r}_1^1}, p_1^1 \cdot Y_1^{\boldsymbol{r}_1^1})$, $\mathcal{E}_1^2(key_{\mathbf{1}}^2, \boldsymbol{r}_1^2) = (G_1^2, M_1^2) = (g^{\boldsymbol{r}_1^2}, p_1^2 \cdot Y_1^{\boldsymbol{r}_1^2})$. Here and afterward p_i^j stands for key_i^j's corresponding prime number taken from Table 1. In the same way the second pair of keys are encrypted via second decryption server's public key Y_2 and so on.

   (b) adds all 2K ElGamal ciphertexts $(\mathcal{E}_1^1, \mathcal{E}_1^2, \cdots, \mathcal{E}_K^1, \mathcal{E}_K^2)$ to the ballot.

   (c) computes $\mathrm{H}_2 = \mathcal{H}(M_1^1 \| M_1^2 \| \cdots \| M_K^1 \| M_K^2)$ and prints it on the receipt.

4. After seeing H_2 printed on the receipt the voter randomly selects one key from each key pair $[key^1_{\mathbf{j}}, key^2_{\mathbf{j}}]$ by selecting K indexes $i_j \in [1, 2]$ for $j \in [1, \cdots, K]$.

5. The voting machine prints on the receipt all K indexes i_j with the $key^{i_j}_j$ and also adds to the ballot the indexes and K Chaum-Pedersen proofs $CP_j = (g^{r_{CP_j}}, Y_j^{r_{CP_j}}, z_j)$ proof that $\mathcal{E}^{i_j}_j$ is really a correct encryption of the $key^{i_j}_j$.

6. Voter selects his candidate, whose l bit binary code C is publicly known and visible on the voting machine interface. $C \in [0..00, 0..01, 0..10, \cdots, 1..11]$

7. Voting machine

   (a) computs $R = \bigoplus_{j=1}^{K} key^{\overline{i_j}}_j \oplus C$ and displays it, where $\overline{i_j}$ is the index from pair j not selected by voter at step 4.

   (b) adds R field on the ballot and also prints it on the voter receipt.

   (c) computes one extra pair $\mathcal{E}_0 = (G_0, M_0) = (g^{\mathbf{r}_0}, p_R \cdot (p_1^{\overline{i_1}} \cdot p_2^{\overline{i_2}} \cdot \cdots \cdot p_K^{\overline{i_K}})^2 \cdot \cdot Y_K^{\mathbf{r}_0} \cdots Y_2^{\mathbf{r}_0} Y_1^{\mathbf{r}_0})$ and adds it with aspecial zero-knowledge proof P to the ballot. The computation of P is described in Appendix A.

8. If the voter confirms his/her vote than voting machine signs both the ballot and receipt by Polling Station's private key and gives receipt to the voter.

The checks any voter must do after voting are very similar to the 3 checks described before and are intuitively clear. In case K=4 the receipt given to the voter will look as is shown in Figure 1b. When election phase finishes, then all the ballots are taken from $\mathfrak{BB}$ and all information is erased from them except the R fields, "check element" $\mathcal{E}_0$ and the K ciphertexts, which are not selected by voters to be opened. So the entire set of ballots to be shuffled can be viewed as $N \times (K + 2)$ matrix as it is shown in Table 2.

$\mathcal{E}^1_1$	$\mathcal{E}^1_2$	...	$\mathcal{E}^1_K$	$\mathcal{E}^1_0$	R_1
$\mathcal{E}^2_1$	$\mathcal{E}^2_2$	...	$\mathcal{E}^2_K$	$\mathcal{E}^2_0$	R_2
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$\mathcal{E}^N_1$	$\mathcal{E}^N_2$	...	$\mathcal{E}^N_K$	$\mathcal{E}^N_0$	R_N

Table 2: Encrypted Ballots

The first decryption server can decrypt only the first ciphertext within each ballot, or the first column in the Table 2. The second decryption server can decrypt only the second column and so on. All ciphertexts within ballot must be decrypted to reveal the vote. Note that if all decryption servers collude, they can reveal votes before mix process started and thus break the privacy of ballots. For that reason we require that at least one decryption server to be honest and not cooperate with others before mix process begins. Note that this is as natural requirement as it is for the standard mix nets to require at least one mix party to be honest and not reveal its secret permutation. We require each decryption party to compute in a verifiable way some "fingerprint" which uniquely identifies the keys set encrypted under its public key. The taken "fingerprint" can be used after decryption of already re-encrypted messages to check whether the re-encryption processing of the ciphertexts was performed correctly or not. The "fingerprint" reveals some kinds of possible attack resulted to the modification of entire keys set. This is the case when malicious mix party replace some ballots with his preferred ones. As we said already, the mix can cheat also in "blind" way just by swapping some fields between ballots. Note that this cheating does not modify the entire set of ciphertexts within a column. We show next how our special "check element" will detect such kinds of cheating by detecting at the end of mix process all ballots which have been subjected to "blind" attacks.
Remember that all keys were transformed to corresponding prime numbers for encryption. Unique factorization in $\mathbb{Z}$ implies that if we compute the product of all encrypted primes, than the resulted quantity can be viewed as a "fingerprint" for that keys set. Homomorphic property of ElGamal allows each decryption server to compute in a verifiable way and publish the product without revealing keys. For each column the "fingerprint" is computed in the following way: All ballots are divided into $\Delta = \lceil N/\alpha \rceil$ groups where each group size is $\alpha = \lfloor log_{p_M} P \rfloor$. It can be done by sorting all ballots by R fields and taking the first Δ ballots as the first group, next Δ ballots as the second group and so on. Such dividing ensures that within each group the product of all keys(primes) computed using homomorphic property of ElGamal will not overflow modulo P. The Δ groups are denoted as $S_1, S_2, ..., S_\Delta$. Each decryption party D_i $i \in [1, \cdots, K]$ computes $F_1^i, F_2^i, \cdots, F_\Delta^i$ in a homomorphic way

$$F_1^i = D_{x_i}(\prod_{l\in S_1} G_1^l, \prod_{l\in S_1} M_1^l) = (g^{\sum_{l\in S_1} r_i^l}, (\prod_{l\in S_1} p_1^l)\cdot Y_i^{\sum_{l\in S_1} r_i^l}) = \prod_{l\in S_1} p_1^l$$

$$F_2^i = D_{x_i}(\prod_{l\in S_2} G_1^l, \prod_{l\in S_2} M_1^l) = (g^{\sum_{l\in S_1} r_i^l}, (\prod_{l\in S_2} p_1^l)\cdot Y_i^{\sum_{l\in S_2} r_i^l}) = \prod_{l\in S_2} p_1^l$$

$$\vdots$$

$$F_\Delta^i = D_{x_i}(\prod_{l\in S_\Delta} G_1^l, \prod_{l\in S_\Delta} M_1^l) = (g^{\sum_{l\in S_\Delta} r_i^l}, (\prod_{l\in S_\Delta} p_1^l)\cdot Y_i^{\sum_{l\in S_\Delta} r_i^l}) = \prod_{l\in S_\Delta} p_1^l$$

The correctness of this quantities can be checked by computing the product of ElGamal ciphertexts within each group S and asking decryption server to decrypt the product and also provide a zero-knowledge proof of correct decryption. Thus ith decryption server's published "fingerprint" will be $F_i = \prod_{s=1}^{\Delta} F_s^i$.

After computation of all "fingerprints" the mix process begins as it is shown in Figure 2. First mix party takes all ballots, re-encrypts all ciphertexts within each ballot including the "check" element and outputs the obtained result in permuted order to the first decryption party. Then decryption party should :

1. Decrypt N ElGamal ciphertexts.
2. Prove the correct decryption of those N ciphertexts.
3. Transform each decrypted prime to corresponding binary key and XOR the key with R field for each ballot.
4. Reconstruct N "check" elements .

When decryption party decrypts all ciphertexts and provides a proof of correct decryption, then anyone can check whether the product of all the revealed messages(primes) is equal to the preprinted "fingerprint". The non-equality will mean that mix party replaced some of the ballots. The malicious mix party must be removed and the mix must be restarted by another mix party. For showing how reconstruction of "check" element is done, let us assume that for the first ballot the revealed prime is p_1 and the "check" element is $(g^{\mathbf{r}_0}, p_R \cdot (p_1 p_2 \cdots p_K)^2 Y_K^{\mathbf{r}_0} \cdots Y_2^{\mathbf{r}_0} Y_1^{\mathbf{r}_0})$. For reconstruction of "check" element decryption party first computes $Y_1^{r_0} = (g^{r_0})^{x_1}$ and provides a proof of correct computation. Then the reconstructed element will be
$(g^{\mathbf{r}_0}, p_R \cdot (p_1 p_2 \cdots p_K)^2 Y_K^{\mathbf{r}_0} \cdots Y_2^{\mathbf{r}_0} Y_1^{\mathbf{r}_0} / p_1 \cdot Y_1^{r_0}) = (g^{\mathbf{r}_0}, p_R p_1 \cdot (p_2 \cdots p_K)^2 Y_K^{\mathbf{r}_0} \cdots Y_2^{\mathbf{r}_0})$.

For performing all 4 steps the decryption party is required to do 1 exponentiation for each decryption plus 2 exponentiations for generating each proof plus 1 exponentiation for reconstructing the "check" element. The correct computation of $Y_1^{r_0}$ can be proved by using the commitments already obtained for proving correct decryption. Thereby, the overall amount of modular exponentiations will be 4N for each decryption party.

There are following 3 type of attacks against the integrity of elections, which can be done by malicious mix parties:

1. Target attack: Attacker can replace the ballot row with his specified one.
2. "Blind" attack 1: Attacker can replace some of ciphertexts within any vector including "check" element.
3. "Blind" attack 2: Attacker can just swap any fields (ciphertexts, R, "check" elements) among two ballots.

As we have already said, the precomputed "fingerprints" allows to detect target attacks done by mix parties. The "blind" attacks can be detected at the end of mix process, when the candidate code will be recomputed and also the "check" value will be revealed. Note that after mix finishes all ballots will contain two values (C_1, C_2), where C_1 will be already the chosen candidate code reconstructed as $\bigoplus_{i=1}^{K} key_i \oplus R$ and C_2 will be the "check" value - $C_2 = p_R \prod_{i=1}^{K} p_i$. Factorization of C_2 will reveal primes $p_R, p_1, p_2, ..., p_K$ from which the original R field and voter's keys $(key_1, key_2, ..., key_K)$ can be constructed. If $C_1 = \bigoplus_{i=1}^{K} key_i \oplus R$ equation holds, it will mean that no "blind" cheating was done against that ballot. If the equation will not hold, it means that any of mix parties changed the ballot by replacing some ciphertexts or R field or "check" element. The malicious mix parties can be discovered by tracing all the problematic ballots by asking each mix party to reveal how it shuffled and re-encrypted the faulty ballots.

In table 3 we compare our mix net with the fastest secure mix network [8].

Scheme	Re-encrypt	Proof and verification	Preliminary work	Decrypt
Polynomial scheme [8]	2N	8N(2K-1)	0	(2+4K)N
Self-proving mixing(this paper)	(K+2)N	0	$\approx (N/\alpha)K$	4KN

Table 3: Cost per server(for a total of K server) of mixing N items with different mix schemes,measured in number of exponentiations.

The cost of re-encryption is higher in our scheme than in others, but we dramatically save in the mix proof and verification steps, actually we do not perform such steps at all. Furthermore, the re-encryption exponentiations can be precomputed. Our mixing scheme can take advantage of the speed-up techniques proposed in [19] for multiple exponentiations with respect to a fixed base. These techniques, based on addition chains, significantly reduce the cost of one exponentiation. This amounts to a very significant speed-up. Note that our scheme is not robust against decryption server failure, because all of them must be at place to decrypt voter keys. To achieve fault tolerance, each decryption server is required to distribute the shares of its secret key in a verifiable way [30] to the other decryption servers before mixing begins. In case any server refuses to properly decrypt, a quorum of remaining honest decryption servers could recover this servers secret key and perform decryption in its place.

5 Conclusion

In this paper a new ballot casting scheme is proposed where each voter encrypts his vote with his secret keys and shares the keys between trustees so that the vote can be decrypted where all keys are available. This scheme provides ballot casting assurance to the voter, who has minimal computational ability. The scheme presented in this paper is one of the few existing e-voting schemes providing ballot-casting assurance to unaided voter. Our scheme yielded us to novel mix construction, which seems very attractive because of its efficiency. It is briefly shown that proposed mix network guaranties the integrity of elections, but the full analysis of presented mix network is beyond the scope of this paper. We suppose that presented approaches of ballot casting and mix verification methods have great potential and must be developed further. For example it is interesting what another symmetric encryption method can be used instead of XOR, which can improve the usability of the scheme.

References

1. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete Logarithms. In George R. Blakley and David Chaum, editors, Advances in Cryptology - CRYPTO '84, volume 196 of LNCS, pages 10-18, 1984
2. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, Advances in Cryptology-CRYPTO '86, volume 263 of LNCS, pages 186-194, 1986.
3. D. Chaum and T. Pedersen. Wallet databases with observers. In Proc. of Crypto'92, pp. 89-105.Springer-Verlag, 1993. LNCS 740.
4. D. Chaum. Secret Ballot Receipts and Transparent Integrity Better and less-costly electronic voting at polling places. http://vreceipt.com/article.pdf, 2004.
5. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In Communications of the ACM, 24(2):84-88, 1981.
6. Adi Shamir. How to share a secret. Communications of the ACM, 22(11):612-613, 1979.
7. D. Boneh, P. Golle. Almost Entirely Correct Mixing With Application to Voting.
8. A. Neff. A verifiable secret shuffle and its application to E-Voting. In Proc. of ACM CCS'01, pp. 116-125. ACM Press, 2001.
9. J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In Proc. of Crypto '01, pp. 368-387. Springer-Verlag, 2001. LNCS 2139.
10. P. Golle, S. Zhong, D. Boneh, M. Jakobsson, A. Juels; Optimsitic mixing for exit-polls, in Y. Zheng, ed., 'Advances in Cryptology (Asiacrypt 2002)', Vol. 2501 of LNCS, Springer-Verlag
11. M. Jakobsson, A. Juels and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In Proc. of USENIX'02.
12. D. Wikstrom. Four practical attacks for "optimistic mixing for exit-polls", Technical Report T2003-04, Swedish Institute of Computer Science.
13. C. Park, K. Itoh and K. Kurosawa. Efficient anonymous channel and all/nothing election Scheme. In Proc. of Eurocrypt 93, pp. 248-259. Springer-Verlag, 1993. LNCS 765.
14. W. Ogata, K. Kurosawa, K. Sako and K. Takatani. Fault tolerant anonymous channel. In Proc. of ICICS 97, pp. 440-444, 1997. LNCS 1334.
15. K. Sako and J. Kilian. Receipt-free mix-type voting scheme. In Proc. of Eurocrypt 95. Springer-Verlag, 1995. LNCS 921.
16. M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In Proc. of Eurocrypt 98, pp. 437-447. Springer-Verlag, 1998. LNCS 1403.
17. Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Advances in Cryptology CRYPTO 2006, volume 4117 of Lecture Notes in Computer Science, pages 373392, August 2006
18. M. Jakobsson and A. Juels. Millimix: mixing in small batches. DIMACS Technical Report 99-33.
19. M. Jakobsson. Flash mixing. In Proc. of PODC 99, pp. 83-89. ACM, 1999.

20. B. Adida and C. A. Neff. Efficient receipt-free ballot casting resistant to covert channels. In EVT/WOTE, 2009.
21. T. Pedersen. A Threshold cryptosystem without a trusted party. In Proc. of Eurocrypt91,pp. 522-526, 1991.
22. Y. Tsiounis and M. Yung. On the security of ElGamal based encryption. In Proc. of PKC 98
23. M. Stadler, Publicly verifiable secret sharing. Proc. Eurocrypt '96, pp. 190-199.
24. Moni Naor and Adi Shamir. Visual cryptography. In EUROCRYPT, pages 112, 1994.
25. Stefan Droste. New results on visual cryptography. In Neal Koblitz, editor, CRYPTO, volume 1109 of Lecture Notes in Computer Science, pages 401415. Springer, 1996.
26. Quang Viet Duong and Kaoru Kurosawa. Almost ideal contrast visual cryptography with reversing. In Tatsuaki Okamoto, editor, CT-RSA, volume 2964 of Lecture Notes in Computer Science, pages 353365. Springer, 2004.
27. C. Andrew Neff. Practical High Certainty Intent Verification for Encrypted Votes.
28. Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork(Editor), CRYPTO 2006, volume 4117 of Lecture Notes in Computer Science, pages 373392.
29. Ben Adida and Andrew Ne. Ecient receipt-free ballot casting resistant to covert channel. In EVT'09, 2009.
30. M. Stadler, Publicly verifiable secret sharing. Proc. Eurocrypt '96, pp. 190-199.

A Proof of "Check" Element's Correct Computation

When casting the ballot will contain 2K ElGamal ciphertexts. The messages under K of them are opened, and that ciphertexts are also augmented with Chaum-Pedersen proofs of correct encryption. Lets denote the remaining K ciphertexts by $(G_1, M_1), (G_2, M_1), \cdots, (G_K, M_K)$ where $(G_i, M_i) = (g^{r_i}, p_i Y_i^{r_i})$. There is also one special element (G_0, M_0) we called "check" element, which in case of being correctly formed, must be equal to $(g^{r_0}, p_R(p_1 p_2 \cdots p_K)^2 Y_1^{r_0} Y_2^{r_0} \cdots Y_K^{r_0})$. Lets see how voting machine can provide zero-knowledge proof that (G_0, M_0) is correctly formed without revealing $p_1, p_2, \cdots, p_K$, but just using the ciphertexts $(G_1, M_1), (G_2, M_1), \cdots, (G_K, M_K)$. At first P generates random value s and provides to verifier 2 values $(g^s, Y_1^S Y_2^S \cdots Y_K^S) = (G_s, M_s)$. Then a challenge c is given to P. Acquiring the challenge P computes and gives to V the following K values

$$\begin{gathered} z_1 = c(2r_1 - r_0) + s \\ z_2 = c(2r_2 - r_0) + s \\ \vdots \\ z_K = c(2r_K - r_0) + s \end{gathered}$$

V computes the following quantity - $A = \dfrac{p_R \cdot \prod_{i=1}^{K} (M_i)^2)}{M_0}$.

If M_0 is formed as it should be, then we will have the following equality $A =$

$Y_1^{(2r_1 - r_0)} Y_2^{(2r_2 - r_0)} \cdots Y_K^{(2r_K - r_0)}$. The verifier needs to check the following equations to be sure that the Voting machine didn't cheat while forming (G_0, M_0) or providing the values $z_1, z_2, \cdots, z_K$.

$$\begin{aligned} A \cdot M_s &= \textstyle\prod_{i=1}^{K} Y_i^{z_i} \\ g^{z_1} &= G_1^{2c} \cdot G_s \\ g^{z_2} &= G_2^{2c} \cdot G_s \\ &\vdots \\ g^{z_K} &= G_K^{2c} \cdot G_s \end{aligned}$$

Fiat-Shamir hash trick is used to generate challenge c in non-interactive way - $c = \mathcal{H}(g \| Y_1 Y_2 \cdots Y_K \| g^s \| Y_1^S Y_2^S \cdots Y_K^S)$. Note that the verification of the proof requires 3K exponentations, but the generation of the proof by Voting Machine requires only 2 exponentiations.