Cryptanalysis of a Universally Verifiable Efficient Re-encryption Mixnet

Shahram Khazaei[*]
KTH Royal Institute of Technology
`khazaei@kth.se`

Björn Terelius
KTH Royal Institute of Technology
`terelius@kth.se`

Douglas Wikström
KTH Royal Institute of Technology
`dog@csc.kth.se`

Abstract

We study the heuristically secure mix-net proposed by Puiggalí and Guasch (EVOTE 2010). We present practical attacks on both correctness and privacy for some sets of parameters of the scheme. Although our attacks only allow us to replace a few inputs, or to break the privacy of a few voters, this shows that the scheme can not be proven secure.

1 Introduction

A fundamental problem in implementing electronic elections is how to guarantee the anonymity of the voters. Chaum [1] studied the similar problem of how to allow people to send anonymous e-mail, and introduced *mix-nets* as a solution to this problem.

In Chaum's mix-net, k mix-servers $M_1,\dots,M_k$ are arranged in sequence. Each mix-server M_j generates a public/private key pair and publishes his public key pk_j. To anonymously send a message m_i, the ith sender encrypts the message with all public keys and publishes the resulting ciphertext $\mathsf{Enc}_{pk_1}(\mathsf{Enc}_{pk_2}(\cdots\mathsf{Enc}_{pk_k}(m_i)\cdots))$ on a bulletin board. Let L_0 be the list of all submitted ciphertexts. For $j = 1,\dots,k$, the jth mix-server M_j then takes L_{j-1} as input, removes the outermost layer of encryption using his private key, and permutes the resulting ciphertexts to form its output L_j. Once the last mix-server M_k has decrypted and shuffled the list, he can publish the plaintext messages. One can easily see that Chaum's mix-net prevents linking the plaintext messages published by the last mix-server to the original senders as long as at least one of the servers is honest. On the other hand, any mix-server can replace any ciphertext with a ciphertext of his choice. This is clearly unacceptable in a voting context.

[*]The author is now at the Department of Mathematical Sciences at Sharif University of Technology, Tehran, Iran. The email address is `shahram.khazaei@sharif.edu`.

Another disadvantage of Chaum's mix-net is that the ciphertexts grow with each added mix-server. Park et al. [17] introduced re-encryption mix-nets, where the mix-servers use the homomorphic property of the cryptosystem to re-randomize the ciphertexts instead of decrypting. Sako and Kilian [22] introduced the first universally verifiable mix-net based on the protocol of Park et al. Their mix-net allows each mix-server to prove in zero-knowledge that its output is a re-encryption and permutation of its input. Sako and Kilian's proof was a cut-and-choose protocol, but more efficient proofs of shuffles were given by Neff [15] and Furukawa and Sako [5].

Many other works in the field aim to improve the efficiency of mix-nets, e.g., [10, 9, 7, 11, 12], but vulnerabilities have been found in most mix-nets not based on proofs of shuffles [18, 14, 2, 23, 13].

Puiggalí and Guasch [21] proposed a heuristically secure mix-net at EVOTE 2010 (called the Scytl mix-net in the rest of the paper) which combines ideas of Golle et al. [7] and Jakobsson et al. [12]. To verify that a mix-server correctly re-encrypts and permutes the votes in Scytl's mix-net, a verifier partitions the server's input into blocks and the server reveals the corresponding output blocks. Furthermore, the server proves that the product of the votes in each output block is a re-encryption of the product of the votes in the corresponding input block. This approach is significantly faster than even the most efficient proofs of shuffles [8, 4], but the security is not as well understood.

A version of Scytl's mix-net was implemented and used with four mix-servers in the Norwegian electronic trial elections [16, 6], but all four mix-servers were run by the same semi-trusted party and there was an additional "ballot box". Privacy was ensured under the assumption that either the "ballot box" or the semi-trusted party remained honest. The mix-net was merely used as a way to allow the semi-trusted party to convince auditors that it performed the shuffling correctly, but as far as we know the original plan was to distribute trust on multiple parties by letting different parties run the mix-servers as proposed in [21]. However, our attacks against the mix-net do not apply directly to the Norwegian implementation due to their security model and their choice of parameters. See the discussion in Section 6 for further details.

1.1 Motivation and Contribution

We think it is important to study the Scytl mix-net, since it has already been used in real elections to ensure correctness, and may be used to provide privacy in future elections.

In this paper we demonstrate attacks against both the correctness and the privacy of the proposed mix-net. The attacks are based on a recent attack [13] on mix-nets with randomized partial checking [12] and the observation that votes can be modified without detection if the modified elements end up in the same block during the verification phase.

Our first attack lets the first mix-server break the privacy of any chosen voter or small group of voters, assuming that the server can corrupt $O(\sqrt{b})$ voters, where b denotes the number of blocks in the verification step. The second attack is similar to the first, but reduces the number of voters that have to be corrupted if the two first mix-servers collude. Our third attack uses the particular way the lists are partitioned to allow any mix-server except the first to replace $O(\sqrt{b})$ votes without being detected. The last attack can also be used to violate the privacy of $O(\sqrt{b})$ voters.

1.2 Summary of the Attacks

In the following table, which summarizes our results, b denotes the number of blocks in the verification and $\ell = N/b$ denotes the block size, where N is the number of voters.

	condition	# corrupted servers	# corrupted voters	attack on	# targeted ciphertexts
claimed	-	1	$O(b+1)$	priv.	$O(1)$
Sect. 5.1	-	1	$O(\sqrt{b})$	priv.	$O(1)$
Sect. 5.2	-	2	$O(\sqrt{b/l})$	priv.	$O(1)$
Sect. 5.3	$l \geq b$	1	-	corr.	$O(\sqrt{b})$
Sect. 5.3	$l \geq b$	1	-	priv.	$O(\sqrt{b})$

1.2.1 Concrete examples

In an election with $N = 1000000$ votes and $b = \sqrt{N} = 1000$ blocks, each of size $\ell = 1000$, our first attack can recover the vote of one targeted sender with probability 0.98 by corrupting the first mix-server and about 100 voters. If we instead use the attack of Section 5.2 and corrupt the 2 first mix-servers, then it suffices to corrupt 2 voters to attack the privacy of one chosen voter with probability 0.999. Finally, the third attack in Section 5.3 lets us replace 9 votes by corrupting one mix-server and remain undetected with probability about 0.95.

The blocks in the previous examples were fairly large. If one uses smaller blocks then the cost of running the mix-net increases, but our attacks become less efficient. Puiggalí and Guasch [21] recommend using $\ell = \sqrt[k]{N}$ when mixing N votes using k mix-servers. With 1000000 votes and 3 mix-servers, this gives $b = 10000$ blocks of size $\ell = 100$. The attack in Section 5.1 then requires about 300 corrupted voters in addition to the corrupted mix-server in order to succeed with probability 0.98. By using the second attack and corrupting 2 mix-servers, we can bring down the number of corrupted voters to 30 and be successful with probability 0.93. The third attack does not apply since $\ell < b$.

2 Notation

We consider a mix-net with k mix-servers $M_1, \ldots, M_k$ that provides anonymity for a group of N voters. We denote a cryptosystem by $\mathsf{CS} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where Gen, Enc, and Dec are the key generation algorithm, the encryption algorithm, and the decryption algorithm respectively. The key generation algorithm Gen outputs a pair (pk, sk) consisting of a public key and a private key. We let M_{pk}, C_{pk}, and R_{pk} be the sets of possible plaintexts, ciphertexts, and randomizers, respectively, associated with the public key pk. We write $c = \mathsf{Enc}_{pk}(m, r)$ for the encryption of a plaintext m using randomness r, and $\mathsf{Dec}_{sk}(c) = m$ for the decryption of a ciphertext c. We sometimes view Enc as a probabilistic algorithm and drop r from our notation.

Recall that a cryptosystem is called *homomorphic* if for every public key pk: M_{pk}, C_{pk}, and R_{pk} are groups and Enc_{pk} is a homomorphism from $\mathsf{M}_{pk} \times \mathsf{R}_{pk}$ to C_{pk}, i.e., if for every $m_0, m_1 \in \mathsf{M}_{pk}$ and $r_0, r_1 \in \mathsf{R}_{pk}$ we have

$$\mathsf{Enc}_{pk}(m_0, r_0)\mathsf{Enc}_{pk}(m_1, r_1) = \mathsf{Enc}_{pk}(m_0 m_1, r_0 + r_1) \ .$$

We write M_{pk} and C_{pk} multiplicatively and R_{pk} additively since this is the case in, for example, the ElGamal cryptosystem. Homomorphic cryptosystems allow ciphertexts to be *re-encrypted*. This means that anybody with access to the public key can take a ciphertext c and form $c \cdot \mathsf{Enc}_{pk}(1, r)$, for a randomly chosen $r \in \mathsf{R}_{pk}$, and the resulting ciphertext is identically, but independently, distributed to the original ciphertext.

Throughout the paper we employ the estimate of collision probabilities used to prove the birthday bound. More precisely, we use the fact that if we pick s elements from a large set of size b with repetition, then some element in the set is picked at least twice with probability roughly $1-e^{-\lambda^2/2}$, where $\lambda = s/\sqrt{b}$.

3 Description of the Mix-Net

Puiggalí and Guasch [21] propose a homomorphic mix-net that combines ideas of Golle et al. [7] and Jakobsson et al. [12] for the verification. On a high level, the mix-net works as follows. The voters submit their inputs encrypted with a homomorphic cryptosystem, e.g., ElGamal, to the mix-net. Starting from the first mix-server, in turn, each mix-server re-encrypts and permutes the list of the ciphertexts before passing the list on to the next mix-server in the chain. Once the last mix-server has published his output list on the bulletin board, the verification phase starts. A verifier partitions the input to each mix-server into a number of blocks. The mix-server then reveals the output block corresponding to each input block without revealing how the individual ciphertexts are shuffled. Then the server proves that the product of all the ciphertexts in each output block is a re-encryption of the product of the ciphertexts in the corresponding input block. If the verification is passed, then the mix-servers jointly decrypt the final list of ciphertexts and otherwise the mixing restarts. Below we give more details on the scheme. The reader is referred to [21] for the original description.

3.1 Setup

The mix-net uses a homomorphic cryptosystem, e.g. ElGamal. The public key pk and the corresponding secret key sk are generated during a setup phase and the secret key is verifiably secret shared among the servers [3]. To ensure that the result can be decrypted even if some servers refuse to cooperate there is typically some threshold τ such that any set of τ servers can decrypt the results, but smaller subsets gain no information about the secret key. The details of how this is done is not important for our attacks.

There is also a parameter b for the mix-net. For each mix server, the input list containing N encrypted votes will be divided into b blocks of (almost) equal size ℓ. To simplify the exposition we assume that $N = \ell b$. Our results are easily generalized to the case where b does not divide N, e.g., by allowing $N \bmod b$ blocks to have size $\ell + 1$ and the remaining blocks to have size ℓ.

3.2 Ballot Preparation and Encryption

The ith voter computes an encryption $c_{0,i} = \mathsf{Enc}_{pk}(m_i)$ of its vote m_i, and posts the ciphertext on the bulletin board. To prevent voters from performing Pfitzmann's attack [19, 18] directly, each ciphertext is also augmented with a non-interactive zero-knowledge proof of knowledge of the plaintext.

3.3 Initial Ballot Checking

When all voters have submitted their ciphertexts, the mix-servers check that the proofs are valid and that no vote has been duplicated. After removing any such duplicate or invalid ciphertexts, the mix-servers agree on an initial list $L_0 = (c_{0,1}, \ldots, c_{0,N})$ of submitted ciphertexts. This ballot checking is not important in our attacks since even the corrupted voters will submit valid votes. Therefore we assume, without loss of generality, that the list L_0 contains N distinct well-formed ciphertexts.

3.4 Mixing Phase

For $j = 1, \ldots, k$, the jth mix-server M_j reads the list of ciphertexts $L_{j-1} = (c_{j-1,1}, \ldots, c_{j-1,N})$ from the bulletin board, chooses a permutation π_j and re-encryption factors $r_{j,1}, \ldots, r_{j,N}$ randomly, computes

$$c_{j,i} = \mathsf{Enc}_{pk}(1, r_{j,\pi_j(i)}) \cdot c_{j-1,\pi_j(i)} \ ,$$

and writes $L_j = (c_{j,1}, \ldots, c_{j,N})$ on the bulletin board.

3.5 Verification Phase

The verification is performed in a challenge-response manner, with the challenge being a partitioning of the mix-server's input list. The parameters b and ℓ, chosen before the mixing starts, denote the number of blocks in the partitioning and the size of each block respectively.

Challenge-Response. Each mix-server M_j, receives a partitioning of its input list L_{j-1} into b blocks as a challenge. More precisely, M_j receives a partitioning $I_{j-1,1}, \ldots, I_{j-1,b}$ of the set $[1,N]$ where the tth block of ciphertexts are those in L_{j-1} whose indices are in $I_{j-1,t}$. For each $I_{j-1,t}$, the server reveals the corresponding block of re-encrypted votes. In other words, M_j defines

$$O_{j,t} = \left\{ \pi_j^{-1}(i) \;\middle|\; i \in I_{j-1,t} \right\}$$

and publishes $O_{j,1}, \ldots, O_{j,b}$ along with a proof that the product of the ciphertexts in each output block is a re-encryption of the ciphertexts in the corresponding input block, i.e., a proof of knowledge of an $R_{j,t}$ such that

$$\prod_{i \in O_{j,t}} c_{j,i} = \mathsf{Enc}_{pk}(1, R_{j,t}) \cdot \prod_{i \in I_{j-1,t}} c_{j-1,i} \ .$$

Clearly M_j knows $R_{j,t} = \sum_{i \in I_{j-1,t}} r_{j,i}$ since he picked the re-encryption factors himself. Note that the mix-servers do not prove that each output block is a permutation and re-encryption of the input block.

Input Partitioning. The partitioning of the input of the first mix-server can be generated randomly by a trusted party, jointly by the mix-servers themselves, or using a random oracle. For our attacks, it is not important exactly how the input partitioning of the first mix-server is chosen as long as it is random.

For every other mix-server the partitioning of the input is determined by the partitioning of the output of the preceding mix-server. More precisely, to form the input partitioning of M_j, the indices of each output block $O_{j-1,1}, \dots, O_{j-1,b}$ are first sorted by numerical value [20]. Then the first input block $I_{j-1,1}$ of mix-server M_j is defined by choosing the first element of $O_{j-1,1}$, the first element of $O_{j-1,2}$, the first element of $O_{j-1,3}$ and so on until the block is full; once the first elements of every block $O_{j-1,1}, \dots, O_{j-1,b}$ has been used, the process is continued with the second element from each of those blocks in the same order. This process is continued until all output blocks are full.

Puiggalí and Guasch have considered other ways of generating the challenge partitionings [20], e.g., a random partitioning could be chosen independently for each mix-server. When we present our attacks we also discuss the impact of changing the partitioning scheme.

3.6 Ballot Decryption

If the mixing operation completes without detecting any cheating mix-server, then the holders of the secret key sk jointly decrypt all output ciphertexts, yielding the full list of plaintext ballots. Otherwise, the mixing starts from the beginning after eliminating the mix-server that failed to respond to its challenge partitioning.

4 Pfitzmann's Attack

A modified variant of the attack of Pfitzmann [19, 18] and its generalization [13] can be adopted to break the privacy of any given group of voters (of constant size) with probability roughly $1/b$, where b is the number of blocks. Since this forms the basis of our attacks on privacy, we detail it below.

The attacker knows the correspondence between voters and initial ciphertexts and targets a group of s voters with submitted ciphertexts $c_1, \dots, c_s \in L_0$. It corrupts the first mix-server and selects two additional ciphertexts $c_{0,1}, c_{0,2} \in L_0$. Then he chooses exponents $\delta_1, \dots, \delta_s$ randomly and forms a modified list L'_0 by replacing $c_{0,1}$ and $c_{0,2}$ by

$$u_1 = \prod_{i=1}^{s} c_i^{\delta_i} \text{ and } u_2 = \frac{c_{0,1} c_{0,2}}{u_1} \ .$$

Finally, he re-encrypts the ciphertexts in L'_0 and permutes them to form L_1 and publishes L_1 on the bulletin board like an honest mix-server.

If the mix-net produces an output, then the attacker searches for $s+1$ plaintexts $m_1, \dots, m_s$ and m in the final list of plaintext ballots that satisfy $m = \prod_{i=1}^{s} m_i^{\delta_i}$. This lets the attacker conclude that with overwhelming probability the ith targeted ciphertext is an encryption of m_i. We must show that M_1 passes the verification step with probability $1/b$.

By construction, $u_1 u_2 = c_{0,1} c_{0,2}$ so if $1, 2 \in I_{0,t}$ for some t, then

$$\prod_{i \in O_{1,t}} c_{1,i} = \mathsf{Enc}_{pk}(1, R_{1,t}) \cdot \prod_{i \in I_{0,t}} c_{0,i} \ ,$$

where $R_{1,t} = \sum_{i \in I_{0,t}} r_{1,i}$. That is, the proof that the first mix-server provides for the modified list L'_0 is also a valid proof for the original list L_0. Thus, the attack succeeds with probability $1/b$ and breaks the privacy of a constant number of voters. More voters can not be attacked, since the complexity of identifying the desired $s+1$ plaintexts in the output list grows exponentially with the number of attacked voters s.

We remark that the attack may be detected at the end of the mixing if the modified ciphertexts u_1 and u_2 do not decrypt to valid votes. Thus, it is in practice likely that the mix-net would have been replaced after a first successful attack.

5 Our Attacks

The basic attack of Section 4 only requires corrupting the first mix-server without corrupting any voter, but it is only successful with low probability. In this section, we first show how an attacker can perform Pfitzmann's attack without detection by corrupting a small number of voters in addition to the first mix-server. Then we describe an attack on privacy which reduces the required number of corrupted voters further, but needs two mix-servers to be corrupted. Finally, we present an attack on the correctness of the mix-net which can also be turned into an attack on privacy. We detail our attacks below.

5.1 Attack on Privacy Without Detection

The key idea stems from a recent attack [13] on mix-nets with randomized partial checking [12] and can be explained as follows. If several corrupted voters submit re-encryptions of the same ciphertext, then a corrupted mix-server has the freedom to match them arbitrarily to their re-encryptions in his output list during the verification. Thus, if any two ciphertexts submitted by corrupted voters belong to the same block of the input partitioning, we may map them to ciphertexts u_1 and u_2 defined as in Section 4. This happens with high probability by the birthday bound if the attacker corrupts enough voters. We now detail the attack.

Without loss of generality we assume that the corrupted voters submit ciphertexts $c_{0,1},\ldots,c_{0,B}$ that are constructed to be re-encryptions of one another. It is not essential that the adversary corrupts the first B voters; the attack works the same with any B corrupted voters as long as the attacker knows how to re-encrypt one vote to any other. To simplify the exposition we assume that $c_{0,i} = \mathsf{Enc}_{pk}(1,r_i)$ for these ciphertexts. The first mix-server is corrupted and forms a modified list L'_0 by replacing $c_{0,1}$ and $c_{0,2}$ by u_1 and u_2, which are computed as in Section 4. That is, the attacker chooses random exponents $\delta_1,\ldots,\delta_s$ and sets $u_1 = \prod_{i=1}^{s} c_i^{\delta_i}$ and $u_2 = c_{0,1}c_{0,2}/u_1$, where the c_i's are the ciphertexts submitted by the targeted voters. All the remaining $N-2$ ciphertexts are left unchanged. Then the first mix-server re-encrypts each ciphertext in L'_0 and permutes them to form L_1 as an honest mix-server.

If any two ciphertexts submitted by corrupted voters end up in the same input block to the first mix-server, we say that a collision has occurred. More precisely, we have a collision if there are two ciphertexts c_{0,i_1} and c_{0,i_2} submitted by corrupted voters and an input block $I_{0,t}$ such that $i_1,i_2 \in I_{0,t}$. Let c_{1,i'_1} and c_{1,i'_2} be the re-encryptions of u_1 and u_2 respectively and, similarly, let c_{1,l'_1} and c_{1,l'_2} be the re-encryptions of c_{0,i_1} and c_{0,i_2}. Thus, we have

$$\pi_1^{-1}(1) = i'_1 \qquad \pi_1^{-1}(2) = i'_2$$
$$\pi_1^{-1}(i_1) = l'_1 \qquad \pi_1^{-1}(i_2) = l'_2 \ .$$

To answer the challenge partitioning, the first mix-server instead views it as if c_{0,i_1} and c_{0,i_2} had been replaced by u_1 and u_2 to form L'_0. In other words, the mix-server re-defines π_1 such that

$$\pi_1^{-1}(i_1) = i'_1 \qquad \pi_1^{-1}(i_2) = i'_2$$
$$\pi_1^{-1}(1) = l'_1 \qquad \pi_1^{-1}(2) = l'_2 \ .$$

as shown in Figure 1.

To see that the first mix-server can pass the verification test, note that

$$\begin{aligned} c_{1,i'_1}c_{1,i'_2} &= u_1u_2\cdot\mathsf{Enc}_{pk}(1,r_{1,1}+r_{1,2}) \\ &= c_{1,i_1}c_{1,i_2}\cdot\mathsf{Enc}_{pk}(1,(r_1-r_{i_1}+r_{1,1})+(r_2-r_{i_2}+r_{1,2})) \end{aligned}$$

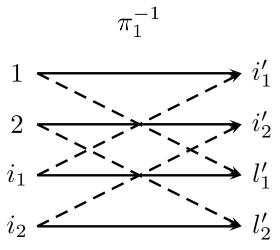

Figure 1: Modification for re-definition of π_1.The solid lines represent the original permutation and the dashed lines show the modifications.

and

$$\begin{aligned} c_{1,l'_1} &= c_{1,1}\cdot\mathsf{Enc}_{pk}(1,r_{i_1}-r_1+r_{1,i_1}) \\ c_{1,l'_2} &= c_{1,2}\cdot\mathsf{Enc}_{pk}(1,r_{i_2}-r_2+r_{1,i_2}) \ , \end{aligned}$$

i.e., it can: replace r_{1,i_1} by $r_1 - r_{i_1} + r_{1,1}$, replace r_{1,i_2} by $r_2 - r_{i_2} + r_{1,2}$, replace $r_{1,1}$ by $r_{i_1} - r_1 + r_{1,i_1}$, and replace $r_{1,2}$ by $r_{i_2} - r_2 + r_{1,i_2}$ and then compute the response with $R_{1,1},\ldots,R_{1,b}$ to the challenge partitionings as an honest mix-server.

Due to the pigeonhole principle, if the attacker corrupts $B = b+1$ voters, he will get a collision with probability one. However, thanks to the birthday paradox, the success probability is already significant if about $\sqrt{b}$ voters are corrupted. In particular, if b is large and we set $B = 3\sqrt{b}$, then we get a collision with probability $1 - e^{-3^2/2} \approx 0.98$.

It is natural to assume that the adversary can choose the indices of his ciphertexts in L_0 or at least influence the order enough to get a random partition to the first mix-server. When this is the case, the attack above applies regardless of how the challenge partitioning is defined. Thus, this is a fundamental flaw of the verification scheme.

5.2 Additional Attack on Privacy

To describe the attack it is convenient to first consider a modified version of the protocol with a different challenge partitioning scheme for the first mix-server. Then, given the attack against the modified protocol, it is easy to see that another attack on the real protocol is possible.

Consider the following modified challenge partitioning scheme. Instead of choosing the input partitioning of the first mix-server at random, choose a random partitioning $O_{0,1},\ldots,O_{0,b}$ of $[1,N]$ and then derive the challenge partitioning $I_{0,1},\ldots,I_{0,b}$ exactly as $I_{j-1,1},\ldots,I_{j-1,b}$ is derived from $O_{j-1,1},\ldots,O_{j-1,b}$ for $j>1$ (see Section 3.5). We attack the mix-net with this modified challenge scheme by corrupting the first mix-server and the first B voters and proceeding exactly as in the previous attack. The only difference is that the probability of a collision is much larger here.

Let $O_{0,i_1},\ldots,O_{0,i_{B'}}$, $B' \leq B$, be the blocks that contain at least one of the integers $1,\ldots,B$. Then within each such block the smallest integer is one of $1,\ldots,B$. Let S be the set of such smallest integers. Then by construction, the integers in S are contained in $I_{0,1} \cup \cdots \cup I_{0,\lceil b/\ell \rceil}$. We say that we get a collision if at least two integers of S appear in some $I_{0,t}$.

For any $\lceil b/\ell \rceil$, it suffices that $|S| > \lceil b/\ell \rceil$ to ensure that there is a collision due to the pigeonhole principle. When $\lceil b/\ell \rceil$ is large, then it suffices that $|S| > 3\sqrt{b/\ell}$ to get a collision with probability at least 0.98 by the birthday bound. The success probability of the attack drops slightly due to the possibility of the event $|S| < B$. Suppose that $\ell \geq 36$ and b is large. If we set $B = 3\sqrt{b/\ell}$, then we get $\lambda = 3/\sqrt{\ell}$ in the approximation of the collision probability, so we can conclude that $|S| = B$ with probability roughly $e^{-\lambda^2/2} \geq e^{-1/8} \approx 0.88$ (and this probability increases rapidly with increasing ℓ). Thus, the probability that the attacker is not detected is roughly $0.98e^{-\lambda^2/2} \geq 0.86$.

Finally, we note that we can transform the attack on the mix-net with the modified way to generate the challenge partitioning of M_1 into an attack on the real mix-net by corrupting both M_1 and M_2. The first mix-server follows the protocol except that it does not re-encrypt its input ciphertexts and it chooses the permutation such that the inputs of corrupted voters appear at the top of L_1. Then M_2 plays the role of M_1 in the attack on the modified mix-net.

It is easy to adapt the attack to the case where the permutation π_1 used by M_1 is determined by sorting L_1. To see this, note that M_1 can re-encrypt its input ciphertexts in such a way that the re-encryptions of the input ciphertexts of corrupted voters still appear at the top of L_1 and the attack can be employed by taking the re-encryption exponents of M_1 into consideration.

When one of the mix-servers can influence the challenge partitioning of the input to the following mix-server, then we expect to find similar vulnerabilities, but this attack fails if the challenge partitioning is chosen randomly and independently for each mix-server.

5.3 Attack on Correctness

This attack requires only one corrupted mix-server and shows that if $\ell \geq b$, then the correctness can be attacked by replacing $R = \frac{1}{3}\sqrt{b} - 1$ votes with small probability of detection.

The attacker corrupts a mix-server M_j other than the first one. The mix-server replaces $c_{j-1,1},\ldots,c_{j-1,R}$ by its own ciphertexts $u_1,\ldots,u_R$ and it replaces $c_{j-1,R+1}$ by

$$u_{R+1} = \prod_{i=1}^{R+1} c_{j-1,i} \Big/ \prod_{i=1}^{R} u_i$$

to form a modified list L'_{j-1}. Note that the products of the respective ciphertexts are equal, i.e., $\prod_{i=1}^{R+1} u_i = \prod_{i=1}^{R+1} c_{j-1,i}$. Then it re-encrypts and permutes L'_{j-1} to form L_j following the protocol.

The challenge partitioning $O_{j-1,1},\ldots,O_{j-1,b}$ is randomly chosen. Modifying $R+1 = \frac{1}{3}\sqrt{b}$ votes gives $\lambda = 1/3$ in the birthday bound, so we may conclude that the probability that two integers in $[1, R+1]$ belong to the same block is roughly $1 - e^{-\lambda^2/2} \approx 0.05$. When this is not the case, the integers $1,\ldots,R+1$ all belong to $I_{j-1,1}$. To see that this implies that the attack goes undetected it suffices to note that

$$\begin{aligned}\prod_{i \in O_{j,1}} c_{j,i} &= \mathsf{Enc}_{pk}(1, R_{j,1}) \prod_{i=1}^{R+1} u_i \prod_{i \in I_{j-1,1} \setminus [1,R+1]} c_{j-1,i} \\ &= \mathsf{Enc}_{pk}(1, R_{j,1}) \prod_{i \in I_{j-1,1}} c_{j-1,i} \quad ,\end{aligned}$$

i.e., the revealed randomness is valid for both L'_{j-1} and L_{j-1}.

The mix-server can double the number of replacements by doing the same trick for the ciphertexts that appear at the end of L_{j-1} at the cost of squaring the probability of executing the attack without detection. This is far better than simply increasing R by a factor of two.

It is straightforward to turn this attack on correctness into an attack on privacy by using the ciphertexts $u_1,\ldots,u_R$ to employ Pfitzmann's attack.

6 Discussion

Our first attack shows that by corrupting $O(\sqrt{b})$ voters and the first mix-server, the privacy of *any* targeted voter can be broken without detection. This attack is applicable regardless how the challenge partitioning of the first mix-server is chosen. Thus, this attack illustrates a fundamental shortcoming of the construction unless b is very large.

Our second attack shows that if a mix-server can influence the challenge partitioning of the the next mix-server, then by corrupting both servers and $O(\sqrt{b/\ell})$ voters, the privacy of *any* targeted voter can be violated. Thus, b must be much larger than ℓ. On the other hand, if ℓ is very small, then the overall privacy of the mix-net starts to deteriorate, since much more information about the permutations are revealed. The complexity of the construction also increases drastically. One way to reduce the second attack to the first attack is to choose the challenge partitioning randomly and independently for each mix-server, but this also reduces the overall privacy of the mix-net compared to the proposed scheme.

The third attack shows that if $\ell \geq b$, then no matter how big b is, an adversary that corrupts a single mix-server can replace $O(\sqrt{b})$ ciphertexts, or violate the privacy of up to $O(\sqrt{b})$ arbitrarily targeted voters, without detection. Thus, the mix-net must not be used with $\ell \geq b$.

Our attacks against the mix-net are efficient in terms of the number of corrupted voters as long as b is relatively small. However, we believe that the attacks are relevant even if a substantial number of corrupted voters are required, since all we really need from the corrupted voters is their randomness. (The other requirement, that they all vote for the same candidate, is easy to satisfy in any election with a small number of popular candidates.) Note that the corrupted voters still get their votes counted correctly (except for the two replaced votes). Thus, being corrupted does not pose a drawback for the voter. Furthermore, adding an additional check that the vote is cast as intended will not help in the setting where the voter's computer has been compromised but the voter is unaware of this fact. This is not an unlikely scenario if the voters use their own computers to cast the votes. Also note that while we can only attack the privacy of a small number of people, we can actually target celebrities or other public figures.

Our attacks do not apply directly to the implementation used in the recent electronic elections in Norway due to additional components in the protocol that help ensuring the privacy of the voters. In particular, the encrypted ballot are collected by a "ballot-box" and it is assumed that an attacker can not corrupt both the ballot-box and the mix-servers. If we could corrupt the ballot-box, then our attacks on privacy would work, but would still be inefficient due to the small value of ℓ. The attack on correctness does not work for the same reason. However, there is no guarantee that more serious vulnerabilities cannot be found and our attacks precludes a proof of security for the mix-net.

Acknowledgments

We are grateful to Jordi Puiggalí and Sandra Guasch Castelló for clarifying some details of their mix-net. We thank Kristian Gjøsteen for answering questions about his analysis of the voting system used in Norway. We also thank Ben Adida for providing comments that helped improve the presentation in this paper.

References

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

[2] Y. Desmedt and K. Kurosawa. How to break a practical mix and design a new one. In B. Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 557–572. Springer, 2000.

[3] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437. IEEE Computer Society, 1987.

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

[5] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2001.

[6] K. Gjøsteen. Analysis of an internet voting protocol. Cryptology ePrint Archive, Report 2010/380, 2010. `http://eprint.iacr.org/`.

[7] P. Golle, S. Zhong, D. Boneh, M. Jakobsson, and A. Juels. Optimistic mixing for exit-polls. In Y. Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2002.

[8] J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *PKC '03: Proc. of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, pages 145–160, London, UK, 2003. Springer-Verlag.

[9] M. Jakobsson. A practical mix. In *EUROCRYPT*, pages 448–461, 1998.

[10] M. Jakobsson. Flash mixing. In *PODC*, pages 83–89, 1999.

[11] M. Jakobsson and A. Juels. An optimally robust hybrid mix network. In *PODC*, pages 284–292, New York, NY, USA, 2001. ACM Press.

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

[13] S. Khazaei and D. Wikström. Randomized partial checking revisited. Cryptology ePrint Archive, Report 2012/063, 2012. `http://eprint.iacr.org/`.

[14] M. Mitomo and K. Kurosawa. Attack for flash mix. In T. Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2000.

[15] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *CCS '01: Proc. of the 8th ACM conference on Computer and Communications Security*, pages 116–125, New York, NY, USA, 2001. ACM.

[16] Norwegian E-VOTE 2011 Project. `http://www.regjeringen.no/en/dep/krd/prosjekter/e-vote-2011-project.html`. 17 February, 2012.

[17] C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*, pages 248–259, 1993.

[18] B. Pfitzmann. Breaking efficient anonymous channel. In *EUROCRYPT*, pages 332–340, 1994.

[19] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of mixes. In *EUROCRYPT*, pages 373–381, 1989.

[20] J. Puiggalí. Private communication. January, 2012.

[21] J. Puiggalí Allepuz and S. Guasch Castelló. Universally verifiable efficient re-encryption mixnet. In R. Krimmer and R. Grimm, editors, *Electronic Voting*, volume 167 of *LNI*, pages 241–254. GI, 2010.

[22] K. Sako and J. Kilian. Receipt-free mix-type voting scheme — a practical solution to the implementation of a voting booth. In *EUROCRYPT*, pages 393–403, 1995.

[23] D. Wikström. Five practical attacks for "optimistic mixing for exit-polls". In M. Matsui and R. J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2004.