Randomized Partial Checking Revisited

Shahram Khazaei* and Douglas Wikström

KTH Royal Institute of Technology
khazaei@kth.se,
dog@csc.kth.se

Abstract. We study mix-nets with randomized partial checking (RPC) as proposed by Jakobsson, Juels, and Rivest (2002). RPC is a technique to verify the correctness of an execution both for Chaumian and homomorphic mix-nets. The idea is to relax the correctness and privacy requirements to achieve a more efficient mix-net.

We identify serious issues in the original description of mix-nets with RPC and show how to exploit these to break both correctness and privacy, both for Chaumian and homomorphic mix-nets. Our attacks are practical and applicable to real world mix-net implementations, e.g., the Civitas and the Scantegrity voting systems.

1 Introduction

A *mix-net* is a protocol that provides anonymity for a group of senders. This notion was first introduced by Chaum in 1981 [2] to implement anonymous channels in general and electronic voting schemes in particular. A voter submits an encrypted ballot and the mix-net later outputs the plaintexts in random order. Other applications of mix-nets include anonymous web browsing [8], private payment systems [16], and multiparty computation [13].

The original mix-net proposed by Chaum is a decryption mix-net that works as follows with mix-servers $\mathcal{M}_1, \ldots, \mathcal{M}_k$. The jth mix-server generates a key pair (pk_j, sk_j) and publishes the public key. To encrypt a message m_i, the ith sender forms a ciphertext

$$c_{i,0} = \mathsf{Enc}_{pk_1}(\mathsf{Enc}_{pk_2}(\cdots \mathsf{Enc}_{pk_k}(m_i)\cdots)) \ .$$

The mix-servers then take turns and "peel off" a layer of encryption and permute the resulting ciphertexts before publishing them. More precisely, the jth mix-server computes

$$c_{i,j} = \mathsf{Dec}_{sk_j}(c_{\pi_j(i),j-1})$$

for a random permutation π_j. Note that the output of the last mix-server is the list of randomly permuted plaintexts. Chaum's mix-net preserves the privacy of the senders as long as at least one server keeps its secret key and its random permutation secret, but a single mix-server can replace all ciphertexts with ciphertexts of his own choosing.

* The author is now at the Department of Mathematical Sciences at Sharif University of Technology, Tehran, Iran: shahram.khazaei@sharif.edu

E. Dawson (Ed.): RSA 2013, LNCS 7779, pp. 115–128, 2013.

Jakobsson, Juels, and Rivest [15] proposed *Randomized Partial Checking* (RPC) as a technique to address this problem and gave heuristic arguments showing that it should be difficult to change more than a small number of ciphertexts. The idea is strikingly simple: each mix-server is challenged to reveal how he processed a random subset of his input ciphertexts. If more than a handful of the ciphertexts are processed incorrectly, then he should get caught. In this scheme a single mix-server clearly does not provide privacy, but it seems that several mix-servers taken together still provide fairly strong privacy guarantees.

For homomorphic cryptosystems, Park, Itoh and Kurosawa introduced re-encryption mix-nets [20]. Here the mix-servers generate a single joint public key with a verifiably secret shared secret key and decryption is replaced by re-encryption, followed by a joint verifiable decryption step. This means that the size of the ciphertext is independent of the number of mix-servers. Sako and Kilian constructed the first universally verifiable mix-net [23], where senders can verify that the entire shuffle was performed correctly (and not just that their own input was included in the output). Sako and Kilian's construction was based on cut-and-choose zero-knowledge proofs; Neff [18] and Furukawa and Sako [7] gave much more efficient zero-knowledge proofs of shuffles.

Much of the work in the field aim to improve the efficiency of the mix-net, e.g., [12,11,9,14], but most mix-nets not based on proofs of shuffles have been broken or vulnerabilities have been found.

1.1 Motivation and Contribution

Random partial checking (RPC) is fast and in contrast to efficient proofs of shuffles [18,7] it is compatible with *any* cryptosystem for which a mix-server can efficiently prove that it processed a single ciphertext correctly. Thus, it is one of few mix-nets that can be used with cryptosystems conjectured to be secure against quantum computers. Currently, it is also the only viable option to construct a *universally verifiable* mix-net from any cryptosystem. Furthermore, it is perhaps the most common heuristically secure mix-net found in real implementations. Jakobsson et al. only claim a restricted form of security for their mix-net, but as such it has resisted all attacks for 10 years. This makes it an important cryptographic construction to study.

We show that the description of RPC by Jakobsson et al. [15] does not capture their ideas correctly. More precisely, we have discovered fully practical attacks on both the privacy and the correctness of their protocol and notable real world implementations of it, e.g., the Civitas voting system [4]. Using similar ideas we can also attack the correctness of related schemes, e.g., the Scantegrity voting system [3] that was used in real elections including Takoma Park City Municipal Elections 2009 and 2011. The most serious attack allows an adversary to replace the complete output of the mix-net without detection by corrupting a single mix-server, but we can also break the anonymity of targeted senders at low cost in terms of the number of corrupted senders. Furthermore, we show that even if the issues we have identified are handled correctly, an adversary can replace t ciphertexts in the homomorphic mixing without detection with probability roughly $(3/4)^t$ and not 2^{-t} as claimed. Finally, we argue informally that there is no blackbox proof of security for homomorphic mix-nets with RPC.

The proper interpretation of our results is not that the basic ideas behind RPC are flawed, but we think RPC should not be used at all with homomorphic mix-nets, and it should not be used for Chaumian mix-nets until there is a rigorous proof of security. We believe that modeling and proving the security is possible if only the issues identified in this paper are rectified and hope to provide such a proof in future work.

2 Notation

We consider a mix-net with k pairs of mix-servers $\mathcal{M}_1, \ldots, \mathcal{M}_{2k}$ that provide anonymity for a group of N senders $\mathcal{P}_1, \ldots, \mathcal{P}_N$. It is convenient to think of each pair of consecutive mix-servers as if they are executed by a single entity, i.e., $\mathcal{M}_{2j-1}, \mathcal{M}_{2j}$ is executed by the jth party.

We denote a cryptosystem by $\mathcal{CS} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where Gen, Enc, and Dec denote 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 $\mathcal{M}_{pk}$, $\mathcal{C}_{pk}$, and $\mathcal{R}_{pk}$ be the sets of 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 often 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: $\mathcal{M}_{pk}$, $\mathcal{C}_{pk}$, and $\mathcal{R}_{pk}$ are groups and for every $m_0, m_1 \in \mathcal{M}_{pk}$ and $r_0, r_1 \in \mathcal{R}_{pk}$ we have

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

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 $r \in \mathcal{R}_{pk}$, and the resulting ciphertext is identically, but independently, distributed to the original ciphertext.

We extend our notation to lists of keys. For a plaintext m and lists of public and private keys $pk = (pk_1, \ldots, pk_\ell)$ and $sk = (sk_1, \ldots, sk_\ell)$, we write

$$\begin{aligned} \mathsf{Enc}_{pk}(m) &= \mathsf{Enc}_{pk_1}(\mathsf{Enc}_{pk_2}(\cdots \mathsf{Enc}_{pk_\ell}(m) \cdots)) \ \text{ and} \\ \mathsf{Dec}_{sk}(c) &= \mathsf{Dec}_{sk_\ell}(\mathsf{Dec}_{sk_{\ell-1}}(\cdots \mathsf{Dec}_{sk_1}(c) \cdots)) \ . \end{aligned}$$

3 Randomized Partial Checking

In this section we provide a brief description of the mix-net with randomized partial checking (RPC) proposed by Jakobsson et al. [15] that focuses on the most common variations. We mostly borrow their notation in the following for easy reference.

The mix-net with RPC is not intended to provide full correctness or privacy; it gains in efficiency by relaxing the security requirements. The goal is to prevent mix-servers from undetectably modifying *many* inputs; it is easy to see that a malicious server can succeed in changing a small number of inputs with constant probability. Assuming that the penalty for being identified as a cheater is severe, the authors of [15] argue that

this suffices. The privacy guarantees are also relaxed: while the exact correspondence between senders and their inputs is hidden, some information may still be leaked (e.g., that a specific input did not originate from a specific sender with a probability different from what is ideally expected).

RPC is proposed as a general technique to verify the correctness of both homomorphic and Chaumian mix-nets. A key requirement on the underlying cryptosystem is that it allows a party to transform a ciphertext c into a ciphertext c' using a cryptographic transformation X_j, and then prove that it did so correctly. For a homomorphic mix-net, X_j denotes random re-encryption and proving that this was done correctly amounts to revealing the randomness used. For a Chaumian mix-net X_j denotes decryption. To prove that the transformation was applied a zero-knowledge proof of correct decryption can be used. However, if the cryptosystem allows using the secret key to recover the randomness used to form a ciphertext from the ciphertext itself, then the randomness can simply be revealed. This interesting feature is not mentioned in [15]. In the following we assume for concreteness that the cryptosystem has this special feature.

3.1 Key Distribution

There is a public key pk and a corresponding secret key sk for the mix-net. For the homomorphic mix-net, the public key is jointly generated, whereas for the Chaumian mix-net it is a list of $2k$ keys. For homomorphic mixing, the joint secret key is typically verifiably secret shared among the mix-servers [6] and not known by any subset smaller than a certain threshold. For the Chaumian mix-net, $\mathcal{M}_j$ knows the jth component of the secret key, sk_j, and each such key is verifiably secret shared among all the mix-servers. The threshold used determines the privacy and robustness of the mix-net.

3.2 Ballot Preparation and Encryption

Each sender $\mathcal{P}_i$ encrypts his plaintext m_i as $c_{i,0} = \mathsf{Enc}_{pk}(m_i)$, and sends it to the bulletin board. Pfitzmann [22,21] pointed out that this ciphertext must either be non-malleable (CCA2-secure) on its own, or augmented with a non-interactive zero-knowledge proof of knowledge of the plaintext to provide this property. For concreteness we assume that the latter approach is used.

3.3 Initial Ballot Checking

When all voters have submitted their ciphertexts, all duplicates are removed (preserving a single copy) and ciphertexts with invalid proofs are discarded. Without loss of generality, we assume that this results in a list of N ciphertexts $(c_{1,0}, \ldots, c_{N,0})$.

3.4 Permutation Commitment

Each server $\mathcal{M}_j$ selects a permutation π_j on N elements uniformly at random. The server publishes on the bulletin board a commitment to π_j or π_j^{-1} depending on j being odd or even. The commitment consists of N integer commitments of the form

$$\Gamma_j^{(In)} = \left(\zeta_{w_{i,j}}[\pi_j(i)]\right)_{i=1}^{N} \text{ or } \Gamma_j^{(Out)} = \left(\zeta_{w_{i,j}}[\pi_j^{-1}(i)]\right)_{i=1}^{N},$$

depending on the parity of j, where $\zeta_w[i]$ denotes a commitment to integer i under randomness w. We simply let $\gamma_{i,j}$ denote the ith commitment of mix-server $\mathcal{M}_j$, i.e., $\gamma_{i,j}$ is an element of $\Gamma_j^{(In)}$ or $\Gamma_j^{(Out)}$ depending on the parity of j.

3.5 Mix-Net Processing

Each server $\mathcal{M}_j$, in turn accepts a ciphertext list $(c_{1,j-1}, \ldots, c_{N,j-1})$ from the bulletin board as input, and computes a list $(c_{1,j}, \ldots, c_{N,j})$ as output and publishes it on the bulletin board, where input ciphertext goes through the cryptographic transformation

$$c_{i,j} = X_j(c_{\pi_j(i),j-1}) \ .$$

For the homomorphic mixing, the transformation X_j re-encrypts the input ciphertext using the joint public key. For the Chaumian mix-net, X_j is the decryption algorithm executed with the secret key sk_j.

3.6 Correctness Check

Each mix-server $\mathcal{M}_j$ is verified as follows. The mix-servers jointly select a collection of ciphertexts from its input or output list, depending on j being even or odd. The selection method is explained in the next section. Mix-server $\mathcal{M}_j$ is then asked to reveal a collection of input/output correspondences related to the selected ciphertexts. Suppose that $\mathcal{M}_j$ wishes to reveal information that allows anyone to verify that an input ciphertext $c_{k,j-1}$ maps to $c_{i,j}$. The mix-server $\mathcal{M}_j$ reveals the triple $(k, i, r_{k,i,j})$ where $r_{k,i,j}$ is the information required to validate the transformation $c_{i,j} = X_j(c_{k,j-1})$. In the case of the Chaumian mix-net, $r_{k,i,j}$ is the randomness chosen by a sender when encrypting $c_{i,j}$ to obtain $c_{k,j-1}$; in the case of the homomorphic mix-net, $r_{k,i,j}$ is the randomness value chosen by $\mathcal{M}_j$ itself for re-encrypting $c_{k,j-1}$.

Additionally, $\mathcal{M}_j$ reveals his commitment to the mapping from $c_{k,j-1}$ to $c_{i,j}$. An odd-numbered mix-server $\mathcal{M}_j$, for the selected ciphertext $c_{i,j}$, decommits $\gamma_{i,j}$ (revealing $k = \pi_j(i)$) whereas an even-numbered mix-server, for the selected ciphertext $c_{k,j-1}$, decommits $\gamma_{k,j}$ (revealing $i = \pi_j^{-1}(k)$).

If all the input/output correspondences verifies correctly, then the mix-server passes the correctness check. Otherwise, he is identified as a cheater.

Inconsistent Permutation Commitments are Possible. We observe that Jakobsson et al. do not stress the importance of checking that the opened commitments of integers are *consistent with a permutation*. That is, all the revealed commitments $\gamma_{i,j}$'s (or $\gamma_{k,j}$'s) must open to *distinct* values. In fact, they emphasize that the verification of correspondences can be performed independently, i.e., it is easy to parallelize. In Section 6, we explore this issue in depth.

3.7 Selection Strategy

Jakobsson et al. propose different schemes for how to select a subset for each mix-server's inputs. The *pairwise dependent selection* scheme is favored and can be

described as follows. Two adjacent mix-servers are paired to ensure that no overlapping correspondences are revealed. The output list $(c_{1,2j-1}, \ldots, c_{N,2j-1})$ of an odd-numbered mix-server is divided in two groups of ciphertexts. Then $\mathcal{M}_{2j-1}$ is challenged with one group and $\mathcal{M}_{2j}$ with the other one.

The partitioning of the output list of an odd-numbered mix-servers is done as follows. Mix-servers jointly compute a random seed R. One way to do this is that each mix-server commits to a random value R_j before starting verification. Then, they open their commitments and compute R as the XOR of all R_j's. The seed R can be used to determine which challenges each mix-server needs to answer. For achieving universal verifiability, RPC combines the random seed R with the contents of the bulletin board, denoted by $\mathcal{BB}$, to compute a seed Q_{2j-1}, e.g., by computing the hash value $H(H(R, \mathcal{BB}), j)$ where H is a cryptographic hash function. The seed Q_{2j-1} is interpreted as a vector of N boolean values of which half are true which is used to divide $(c_{1,2j-1}, \ldots, c_{N,2j-1})$ in two disjoint groups.

The correctness check of mix-servers can be done in two ways: *in-phase with mixing*, i.e., right after each mix-server pair has published his output list, or *after all mixing has been performed*, i.e., the last mix-server has published his output list. Both schemes are proposed for homomorphic mixing, but if cheating is detected in the second scheme, then the culprit would be kicked out and the mixing restarts. Only when the mixing proceeds without any detected cheating does joint decryption take place. For Chaumian mixing the correctness check is suggested to take place in-phase. If a mix-server is identified as a cheater, then his secret key is recovered and his input decrypted in the open.

3.8 Ballot Decryption

For the homomorphic mix-net, once the mixing operation is complete without detecting any cheating mix-server, the holders of the secret key sk (the mix-servers or some other entities) jointly decrypt all output ciphertexts, yielding the full list of plaintext ballots. This decryption operation is not needed in the case of a decryption mix-net, since the X_j transformations have already performed all necessary decryptions.

4 Pfitzmann's Attack and a Generalization

It is easy to see that for the homomorphic mix-net with RPC, the attack of Pfitzmann [22,21] can be adopted to break the privacy of any given sender with probability $1/2$. This forms the basis of our attacks on privacy, so it is worthwhile to describe it in detail.

The first mix-server knows the correspondence between voters and ciphertexts. He targets a sender with a submitted ciphertext c. Then he chooses an integer δ of suitable size randomly and replaces one of his outputs by c^δ. With probability $1/2$ this is not detected during RPC. Then he waits until the mix-net produces an output, identifies two plaintexts m and m^* that satisfy $m^* = m^\delta$, and concludes that m was submitted by the targeted sender.

With more processing, Pfitzmann's basic attack can be generalized to break the privacy of s senders while keeping the probability of detection equal to $1/2$. To target

some ciphertexts $c_1, \ldots, c_s$, the first mix-server chooses random values $\delta_1, \ldots, \delta_s$ and replaces one of its outputs by the product $\prod_{i=1}^{s} c_i^{\delta_i}$. When the mix-net produces an output, the attacker identifies $s+1$ plaintexts $m_1, \ldots, m_s, m^*$ in the final list that satisfy $m^* = \prod_{i=1}^{s} m_i^{\delta_i}$. The running time of the attack is clearly exponential in δ, so δ should be viewed as a reasonably small constant. Then it concludes that the ith targeted ciphertext is an encryption of m_i.

5 On the Need for Duplicate Removal Everywhere

Recall that duplicate ciphertexts in the list of initial ciphertexts are removed (preserving only the first posted copy). Jakobsson et al. do not emphasize that each mix-server must perform this operation before processing its input. In this section we explore the consequences of failing to do so in a Chaumian mix-net with RPC.

For simplicity, assume that the adversary targets the first s ciphertexts $c_{1,0}, \ldots, c_{s,0}$ and corrupts $s(s+1)/2$ senders and the first and last mix-servers. For each i, the adversary removes the first layer of encryption by computing $c_{i,1} = \mathsf{Dec}_{sk_1}(c_{i,0})$, using the secret key of the first corrupted mix-server. He then makes i independent encryptions of $c_{i,1}$ under the public key of the first mix-server. This way, the adversary has prepared $1 + 2 + \ldots + s = s(s+1)/2$ ciphertexts which will be sent by the same number of corrupted senders — each submitting one ciphertext. By construction there are $i+1$ related ciphertexts of the ith targeted ciphertext. By looking at the input list of the last mix-server, the adversary can identify the targeted senders' ciphertexts (encrypted under the public key of the last mix-server) based on the number of duplicates. Since the last mix-server is corrupted, he learns the plaintexts of the targeted senders. As $s + s(s+1)/2 \leq N$ must hold, this attack can break privacy of at most $\mathcal{O}(\sqrt{N})$ senders.

Jakobsson et al. do suggest to employ a CCA2 secure encryption scheme like OAEP-based RSA [1] to make the initial encryption non-malleable. The problem illustrated by the above attack is that the composition of the cryptosystems of the mix-servers only remain CCA2 secure as long as the first mix-server remain uncorrupted.

Clearly, the attack is prevented if every mix-server removes the duplicates before processing its input list. Notice that the final output list, i.e., the mixed output, can contain duplicates.

We do not see any way to extend the above attack to a homomorphic mix-net with RPC.

6 Inconsistent Commitments Are Dangerous

Jakobsson et al. [15] do not mention that it is essential to verify that those parts of the permutation commitments $\Gamma_j^{(In)}$ (or $\Gamma_j^{(Out)}$) that are opened must be consistent with a permutation. Unfortunately, this also turns out to be the way that implementors have interpreted the paper. A prominent research group in electronic voting generously gave us access to their private source code of their homomorphic mix-net with RPC and it suffered from this flaw. Another notable and publicly available example of an implementation with this flaw is the Civitas [4] election system (version 0.7.1), implemented

by Clarkson et al. The Scantegrity scheme [3] employs a checking approach that resembles random partial checking and suffers from this flaw.

In this section we show how this flaw can be exploited to break either the correctness or the privacy, or a little bit of both, without detection.

6.1 Breaking Privacy without Detection

This attack applies only to the homomorphic mix-net with RPC. Recall our generalization of Pfitzmann's attack from Section 4 that breaks the privacy of s senders with detection probability $1/2$. We show how to mount a variation of this attack without being detected. The adversary targets some s ciphertexts $c_1, \ldots, c_s$. It chooses random values $\delta_1, \ldots, \delta_s$ and computes the product $c = \prod_{i=1}^{s} c_i^{\delta_i}$. Then it corrupts two senders and the first pair of mix-servers (here we assume that they are operated by a single entity). The two corrupt senders are asked to submit two ciphertexts that are re-encryptions of one another. The first mix-server behaves honestly and it is corrupted only in that it keeps track of the ciphertexts submitted by the corrupted senders and how they are re-encrypted. Let $c_{i_1,1}$ and $c_{i_2,1}$ be these ciphertexts in its output list and note that the adversary knows how to transform $c_{i_2,1}$ into $c_{i_1,1}$ by re-encryption.

Recall that $\gamma_{i,2}$ denotes a commitment to $\pi_2^{-1}(i)$ if $\mathcal{M}_2$ behaves honestly. The second mix-server $\mathcal{M}_2$ behaves honestly except that:

1. It defines $\gamma_{i_2,2}$ to be a commitment to $\pi_2^{-1}(i_1)$. The remaining $N-1$ commitments, including $\gamma_{i_1,2}$, a commitment to $\pi_2^{-1}(i_1)$, are computed in the usual way.
   Note that $\mathcal{M}_2$ can not open both $\gamma_{i_1,2}$ and $\gamma_{i_2,2}$ in a way that is consistent with a permutation since they are commitments to the same index i_1.
2. It replaces $c_{i_2,1}$ with the maliciously constructed ciphertext c. The modified list is then re-encrypted and shuffled to give the output list $(c_{1,2}, \ldots, c_{N,2})$.

The attacker has replaced the ciphertext of one of the corrupted senders with a re-encryption of c. If the attack goes undetected, the adversary can clearly identify the targeted senders' plaintexts as in the generalization of Pfitzmann's attack.

To see that the attack is not detected by RPC, first note that both commitments $\gamma_{i_1,2}$ and $\gamma_{i_2,2}$ verify correctly. Then observe that $c_{\pi_2(i_1),2}$ is in fact a re-encryption of both $c_{i_1,1}$ and $c_{i_2,1}$ and $\mathcal{M}_2$ can provide the randomness needed to verify this. The attack can be extended to break the privacy of rs senders without detection by using $r+1$ commitments of i_1 and introducing $r+1$ ciphertexts submitted by corrupted senders that are re-encryptions of each other.

Depending on the method used to decode messages, the attack will be detected when the output plaintexts are interpreted. Thus, in practice, the above attack could probably only be executed once before implementors identified the issue.

Interestingly, even if decommitted values are verified to be distinct, the above attack with two ciphertexts originating from a single ciphertext input by a corrupted party performs better than the attacks considered by Jakobsson et al. We discuss this is Section 7.

6.2 Rigging an Election without Detection

This attack applies to the homomorphic mix-net with RPC, and if the duplicates are not removed (see Section 5), it is also applicable to the Chaumian mix-net. A straightforward adaptation of the attack applies to the Scantegrity voting system [3].

The adversary corrupts the first sender and the first mix-server. The first sender is asked to use m as his plaintext. During the attack all other encrypted votes are replaced by m. The first mix-server simply replaces all the ciphertexts by $c_{1,0}$, i.e., the submitted ciphertext by the first sender which is an encryption of m. The modified list is then correctly transformed and shuffled to produce the output list $(c_{1,2}, \ldots, c_{N,2})$. To avoid being detected, the first mix-server chooses all the $\gamma_{i,1}$'s as commitments to 1, i.e., all of them are opened to 1. Clearly, the first mix-server can then provide the evidence that $c_{1,0}$ maps to all $c_{i,2}$'s. Therefore, the output list of the last mix-server is N encryptions of m and the attack is not detected.

To make the resulting output look less unrealistic, the attacker can of course submit encryptions of several different plaintexts and choose a suitable distribution over these and apply the attack for each such plaintext.

We stress that there is no way to notice this attack except manually inspecting the list of decommitted integers. Thus, this attack could in fact already have been exploited in executions of mix-nets with RPC, so we suggest that transcripts of old executions of such implementations are inspected manually.

7 What Is the Best We Can Hope for?

Note that even if duplicates are removed everywhere and it is verified that all *opened* commitments contain distinct integers to prevent the attacks of the previous sections, then this does not guarantee that all the unopened commitments are consistent with a permutation. We show that we can still replace ciphertexts in the homomorphic mix-net or eliminate in Chaumian mix-net with notably better probability than the attack considered optimal by Jakobsson et al.

The attack is essentially the same as in Section 6.2 except that only *one* ciphertext submitted by an honest sender is replaced by a copy of the ciphertext of the corrupted sender. Then the probability of detection is $1/4$, since the attack is only detected if the two commitments containing the same integer are opened. The attack can be repeated independently t times to replace t ciphertexts with probability $(3/4)^t$.

For the homomorphic mix-net, replacing ciphertexts translates to replacing the final plaintexts, i.e., the output of the mix-net. For Chaumian mix-net our replacements corresponds to replacing the final plaintext if the last mix-server in the chain makes the replacements; otherwise, since the duplicates are removed, replacing ciphertexts results in eliminating plaintexts from the final mixed output.

8 On the Universal Verifiability of RPC

Recall that a mix-net is called universally verifiable if it ensures correctness even if the adversary corrupts all parties, and consider the case where checking takes place at the

end of the mixing. If all mix-servers are corrupted, then they can try to cheat by repeatedly: replacing t ciphertexts in their output, picking a random seed as in the protocol, and checking if the resulting challenges can be answered. The attack is successful if a replacement is found that passes the correctness check. Assuming the adversary repeats the procedure q times, Jakobsson et al. argue that the success probability in each attempt is bounded by 2^{-t} and then apply the union bound to conclude that the probability of success is roughly $1-(1-2^{-t})^q \leq q2^{-t}$.

8.1 An Improved Attack

There is a more clever attack on the public verifiability of homomorphic mixing with RPC. A straightforward application of the idea of Section 7 shows that the success probability can be increased to roughly $1-(1-(3/4)^t)^q$. In other words, the adversary can change $1/\log(4/3) \approx 2.4$ times as many ciphertexts as claimed for a given computational complexity. For example, to change $t = 192$ ciphertexts the adversary has to prepare and do 2^{80} hash queries. This should be compared with the claimed bound of 80 ciphertexts for corresponding complexity. This attack also applies to the Chaumian mixing even if the duplicates are removed. This requires that the replacement is done by the last mix-server.

8.2 When Checking Is Performed at the End of the Mixing

Universal verifiability is considerably weaker when in-phase checking is used. The problem is that in-phase checking allows the adversary to replace a few ciphertexts in each each mix-server. Replacing kt ciphertexts in this way goes undetected with probability roughly $\left(1-(1-(3/4)^t)^q\right)^k$ (instead of $1-(1-(3/4)^{tk})^q$) for homomorphic mixing. For example, with $k = 5$, $t = 192$, and roughly 2^{80} queries to the hash function, the adversary can replace almost one thousand ciphertexts. If we really care about public verifiability, then this is unlikely to be acceptable. This also holds for Chaumian mixing even if the duplicates are removed everywhere but the replaced ciphertexts are eliminated except for the last mix-server.

This leaves us with two options for Chaumian mix-nets: either we adopt checking in-phase with mixing and accept the weaker guarantee on universal verifiability, or we use checking after mixing.

There is a fairly obvious attack on a Chaumian mix-net with RPC if the checking is performed at the end of the mixing and this is why Jakobsson et al. do not suggest this type of checking. The problem is that the adversary can corrupt the first mix-server, replace some ciphertexts, and simply wait for the output of the mix-net. Then he will be caught, but before this happens the votes of the targeted voters have already been revealed.

To prevent this attack and still use checking after mixing, we propose to protect senders plaintexts with an innermost cryptosystem whose secret is shared between the mix-servers and only recovered after the checking has been successfully performed. This can also be implemented by letting each server generate an additional key pair and letting the joint key be the list of all the additional public keys. This avoids the need

for a costly distributed key generation protocol, but increases the size of ciphertexts. Another problem with this scheme is that the execution can only be aborted if cheating is detected.

Note that the problem with checking at the end of the mixing does not appear in a homomorphic mix-net with RPC since nothing is revealed about the plaintexts until after the checking is completed successfully.

9 On the Provable Security of RPC

Even cryptographic protocols proposed without a proof of security by experienced cryptographers can often be broken, and this seems to be particularly true for mix-nets. Historically the proposals of heuristically secure mix-nets [20,11,12,14,9] have been followed by discovery of security flaws [22,21,5,17,24].

A formal proof of security does not guarantee that no attack will ever be found (proofs can have subtle errors, assumptions can be wrong, and the adversarial model can be unrealistic), but it increases the confidence in the security of the scheme significantly.

9.1 Homomorphic Mix-Net with RPC

We argue informally that homomorphic mix-nets with RPC, e.g., based on El Gamal cannot be proven secure using a blackbox reduction in the simulation paradigm, even if the issues explored in this paper are handled correctly.

A definition of security in the simulation paradigm requires that no efficient distinguisher can tell a suitable ideal model with a simulator from a real model with a real adversary. Suppose that there is a real adversary and a distinguisher that contradicts this claim. We must use them to break the semantic security of the cryptosystem. The first step would be to do a hybrid argument such that two hybrids only differ in that in one of the hybrids the ith voter encrypts his true message and in the other he encrypts some bogus message, e.g. zero. Then to exploit the adversary in a blackbox way we would:

1. Accept a public key from the semantic security experiment as input and somehow embed this into the public key used by voters. We could for example pretend that the public key belongs to one of the honest senders and simulate the verifiable secret sharing of the secret key without knowing the secret key at all.
2. Simulate the execution until the ith voter prepares its ciphertext and interrupt the execution at this point. Then we hand the true plaintext and the bogus plaintext to the experiment and wait for a ciphertext in return which is used as the ciphertext of the ith sender in the continued simulation.
3. Simulate the decryption of the given ciphertext to the true plaintext, and output the output of the distinguisher. (In a homomorphic mix-net we must keep track of how it is permuted through the mix-net to be able to do this.)

The problem with the homomorphic mix-net with RPC is that a corrupted mix-server with probability $3/4$ can replace a ciphertext by *any* ciphertext and before the distinguisher outputs its result, the adversary expects to see the plaintext of this ciphertext

in the output of the mix-net. Thus, the adversary is given access to a restricted decryption oracle *before* the distinguisher guesses which model it is interacting with. In other words, for the adversary to be useful to the reduction, the simulator must solve almost the same problem as the reduction is intended to do.

9.2 Chaumian Mix-Net with RPC

Proving the security of the Chaumian mix-net with RPC based on a CCA2 secure cryptosystem where the vulnerabilities explored in this paper are resolved seems hard, but not impossible. The challenge is to capture the restricted forms of privacy and soundness that it exhibits. The obvious way to resolve the problem with limited privacy is to assume that there are sufficiently many mix-servers to get full privacy, but it turns out to be non-trivial to determine how many mix-servers are needed.

Gomulkiewicz et al. [10] study the probability distribution of the permutations linking the input and outputs of a mix-net with RPC given the information revealed. They show that the distance between this distribution and the uniform distribution is $\mathcal{O}(\frac{1}{N})$ even when there is only a constant number of mix-servers. Contrary to what is claimed, this result does not capture the privacy of a mix-net with RPC, but it may well be a useful result.

10 Interpretation and Discussion

It is easy to add consistency checks to the random partial checking protocol, but such consistency checks are missing in the description of Jakobsson et al. [15]. Implementers should of course follow the description of a cryptographic protocol strictly to make sure that their implementation capture the intentions of the protocol designers, so it is not surprising that the consistency checks are missing in all implementations we have considered. Furthermore, even if the needed consistency checks are added the protocol still does not achieve the claimed security guarantees. Thus, we think it is fair to consider the issues we have identified as *flaws in the protocol* and not as mere implementation bugs, although we realize that different interpretations are possible. Both the Civitas [4] and the Scantegrity [3] teams have reported that they have already mended, or are about to mend, their implementations based on our insights.

Our attack on correctness is easily generalized to be very difficult for a human to discover by a manual sanity check of the values revealed during random partial checking. Thus, we choose to view our attack as *undetectable* in practice, despite that it can be detected using an algorithm that performs the needed consistency checks.

It is hard to exaggerate how fortunate we are to be able to *retroactively verify* that the attack on correctness did not take place in a given execution. We strongly suggest that all implementers of random partial checking (or similar schemes) perform the needed verifications for all conducted elections. The Scantegrity team has already reported that no tampering took place in elections using their scheme.

We stress that all the issues described in this paper were found in an attempt to prove the security of random partial checking and we are convinced that any attempt to do so would have revealed the same issues. Thus, we think that our work illustrates the

importance of precise descriptions and rigorous proofs of security. Proofs of security can have subtle bugs and models can be unrealistic, but we think that protocols without proofs of security should not be trusted.

Acknowledgments. Tal Moran contributed to our initial discussions on possible approaches to prove the security of random partial checking. Johan Håstad gave helpful comments. Members of the Civitas and the Scantegrity teams quickly confirmed our findings and provided valuable feedback on a draft of this paper.

References

1. Bellare, M., Rogaway, P.: Optimal Asymmetric Encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995)
2. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM 24(2), 84–88 (1981)
3. Chaum, D., Essex, A., Carback, R., Clark, J., Popoveniuc, S., Sherman, A., Vora, P.: Scantegrity: End-to-end voter-verifiable optical- scan voting. In: IEEE Security and Privacy, vol. 6, pp. 40–46 (2008)
4. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: IEEE Symposium on Security and Privacy, pp. 354–368. IEEE Computer Society (2008)
5. Desmedt, Y., Kurosawa, K.: How to Break a Practical MIX and Design a New One. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 557–572. Springer, Heidelberg (2000)
6. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: FOCS, pp. 427–437. IEEE Computer Society (1987)
7. Furukawa, J., Sako, K.: An Efficient Scheme for Proving a Shuffle. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 368–387. Springer, Heidelberg (2001)
8. Gabber, E., Gibbons, P.B., Matias, Y., Mayer, A.J.: How to Make Personalized Web Browising Simple, Secure, and Anonymous. In: Hirschfeld, R. (ed.) FC 1997. LNCS, vol. 1318, pp. 17–32. Springer, Heidelberg (1997)
9. Golle, P., Zhong, S., Boneh, D., Jakobsson, M., Juels, A.: Optimistic Mixing for Exit-Polls. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 451–465. Springer, Heidelberg (2002)
10. Gomułkiewicz, M., Klonowski, M., Kutyłowski, M.: Rapid Mixing and Security of Chaum's Visual Electronic Voting. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 132–145. Springer, Heidelberg (2003)
11. Jakobsson, M.: A Practical Mix. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 448–461. Springer, Heidelberg (1998)
12. Jakobsson, M.: Flash mixing. In: PODC, pp. 83–89 (1999)
13. Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: Okamoto [19], pp. 162–177
14. Jakobsson, M., Juels, A.: An optimally robust hybrid mix network. In: PODC, pp. 284–292. ACM Press, New York (2001)
15. Jakobsson, M., Juels, A., Rivest, R.L.: Making mix nets robust for electronic voting by randomized partial checking. In: Boneh, D. (ed.) USENIX Security Symposium, pp. 339–353. USENIX (2002)
16. Jakobsson, M.: Mix-Based Electronic Payments. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 157–173. Springer, Heidelberg (1999)

17. Mitomo, M., Kurosawa, K.: Attack for flash mix. In: Okamoto [19], pp. 192–204
18. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: CCS 2001: Proc. of the 8th ACM Conference on Computer and Communications Security, pp. 116–125. ACM, New York (2001)
19. Okamoto, T. (ed.): ASIACRYPT 2000. LNCS, vol. 1976. Springer, Heidelberg (2000)
20. Park, C., Itoh, K., Kurosawa, K.: Efficient Anonymous Channel and All/Nothing Election Scheme. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 248–259. Springer, Heidelberg (1994)
21. Pfitzmann, B.: Breaking an Efficient Anonymous Channel. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 332–340. Springer, Heidelberg (1995)
22. Pfitzmann, B., Pfitzmann, A.: How to Break the Direct RSA-Implementation of Mixes. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 373–381. Springer, Heidelberg (1990)
23. Sako, K., Kilian, J.: Receipt-Free Mix-Type Voting scheme — A Practical Solution to the Implementation of a Voting Booth. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 393–403. Springer, Heidelberg (1995)
24. Wikström, D.: Five Practical Attacks for "Optimistic Mixing for Exit-Polls". In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 160–175. Springer, Heidelberg (2004)