# Security Analysis of Re-Encryption RPC Mix Nets

Ralf Küsters
*University of Trier*
*Germany*
*kuesters@uni-trier.de*

Tomasz Truderung
*Polyas GmbH*
*Germany*
*ttruderung@gmail.com*

*Abstract*—**Re-Encryption randomized partial checking (RPC) mix nets were introduced by Jakobsson, Juels, and Rivest in 2002 and since then have been employed in prominent modern e-voting systems and in politically binding elections in order to provide verifiable elections in a simple and efficient way. Being one of or even the most used mix nets in practice so far, these mix nets are an interesting and attractive target for rigorous security analysis.**

**In this paper, we carry out the first formal cryptographic analysis of re-encryption RPC mix nets. We show that these mix nets, with fixes recently proposed by Khazaei and Wikström, provide a good level of verifiability, and more precisely, accountability: cheating mix servers, who try to manipulate the election outcome, are caught with high probability. Moreover, we show that, under the assumption that at least one mix server is honest, all attacks that would break the privacy of voters' inputs are caught with a probability of at least $1/4$. In many cases, for example, when penalties are severe or reputation can be lost, adversaries might not be willing to take this risk, and hence, would behave in a way that avoids this risk. Now, for such a class of "risk-avoiding" adversaries, we show that re-encryption RPC mix nets provide a good level of privacy, even if only one mix server is honest.**

## 1. Introduction

Mix nets often play a central role in modern e-voting systems. In such systems, voters' ballots, which typically include the voters' choices in an encrypted form, are posted on a bulletin board. Then, the ballots are shuffled by a mix net, which consists of several mix servers, before they are decrypted. This is supposed to hide the link between a voter's ballot and her (plaintext) choice, and hence, guarantee the voter's privacy. In the context of e-voting, besides privacy, it is also crucial that potential manipulations are detected. That is, if ballots have been dropped or manipulated by a mix server, this should be detected. This property is called verifiability.

Many schemes have been proposed in the literature to obtain verifiable mix nets (see, e.g., [20], [16], [7], [22], [8], [9], [21], [2]). Some were shown to provide strong security properties. However, most of these schemes have not been deployed in real elections so far, with Verificatum [23] being a prominent exception of a provably secure scheme which has also been used in practice. The mix nets that are among or even the most used mix nets in practice to date are so-called re-encryption RPC (random partial checking) mix nets. These mix nets were implemented in several prominent e-voting systems, including Civitas [5] and Prêt à Voter [19], and used in politically binding elections. For example, in a variant of Prêt à Voter, re-encryption RPC mix nets were recently employed in an election of the Australian state of Victoria [6]. Some systems, such as Scantegrity [4], which has also been employed in real political elections, use a similar technique. Hence, it is important to understand and analyze the security of re-encryption RPC mix nets.

Re-encryption RPC mix nets were proposed in 2002 by Jakobsson, Juels, and Rivest [8], as particularly simple and efficient mix nets. Such mix nets consist of several mix servers, where the mix servers use a public key encryption scheme with distributed decryption, with ElGamal being a common choice. Roughly speaking, these mix nets work as follows. The input to a re-encryption RPC mix net is a list of ciphertexts (e.g., encrypted votes), where each ciphertext is obtained by encrypting a plaintext under a common public key. Now, the first mix server shuffles the ciphertexts and re-encrypts them.[1] The resulting ciphertexts form the output of this mix server and the input to the next one, which again shuffles and re-encrypts the ciphertexts, and so on, until the last mix server has done this. Then, the mix servers together, in a distributed way, decrypt each ciphertext in the list output by the last mix server. In order to check whether a mix server cheated, i.e., manipulated/replaced a ciphertext so that it carries a different plaintext, so-called random partial checking is performed for each mix server. For this purpose, every mix server is supposed to reveal some partial information (chosen by auditors) about the input/output relation. Jumping ahead, our results show that this does not require zero-knowledge proofs.

We note that in the same paper, Jakobsson, Juels, and Rivest also proposed Chaumian RPC mix nets, where the input ciphertexts are obtained by nested encryption (using

1. Re-encryption is an operation that can be performed without knowledge of the private key or the plaintext. Given a ciphertext $\mathsf{Enc}^{r}_{pk}(m)$ obtained using the public key $pk$, the plaintext $m$, and the random coins $r$, re-encryption yields a ciphertext of the form $\mathsf{Enc}^{r'}_{pk}(m)$, i.e., one with different random coins.

different public keys for each layer of encryption) and every mix server, instead of performing re-encryption, peels off one layer of encryption. However, to the best of our knowledge, this construction has not been used in practice so far.

From the design of RPC mix nets it is clear that they do not provide perfect security: there is some non-negligible probability that cheating goes undetected and some partial information about the input/output relation is revealed. As already argued by Jakobsson, Juels, and Rivest, in the context of e-voting the penalties for cheating would be so severe that being caught with some (even small) probability should deter a mix server from cheating.

Only very recently, Chaumian RPC mix nets have undergone first formal cryptographic analysis [14], after Khazaei and Wikström discovered attacks on the verifiability and privacy of Chaumian RPC mix nets and proposed fixes [10].

Despite their use in practice, so far no formal security analysis of *re-encryption* RPC mix nets has been carried out. In [10], Khazaei and Wikström pointed out attacks on re-encryption RPC mix nets as well, one of which generalizes an attack by Pfitzmann [18], and proposed fixes, but they did not carry out any formal analysis. In particular, it was left as an open question whether these fixes are sufficient for verifiability. In this paper, we prove that with the fixes, one obtains a good level of verifiability (see below). As for privacy, it was clear that the proposed fixes do not prevent the attacks. However, we can observe that in these attacks on privacy malicious mix servers risk to be caught with significant probability. In this paper, we formally prove that in fact *all* attacks that would break privacy will be caught with high probability and that cheating mix servers can be blamed individually, which should deter them from cheating, e.g., because of severe penalties they would face. We further prove that if mix servers want to avoid being caught (risk-avoiding mix servers, see below), then re-encryption RPC mix nets provide a high level of privacy. More precisely, the contributions of this paper are as follows.

**Contributions of this paper.** We provide the very first formal security analysis of re-encryption RPC mix nets. As mentioned, RPC mix nets by design can provide only restricted forms of verifiability and privacy. Therefore, we need security notions that allow us to measure the level of security re-encryption RPC mix nets provide. For this purpose, we use a definition of privacy which has been used in the context of e-voting before (see, e.g., [13]) and which has also been employed for the analysis of Chaumian RPC mix nets in [14]. This definition focuses on the level of privacy for individual senders/voters and basically requires that for every pair of messages an adversary should not be able to tell which of the two messages a sender has sent. As for verifiability, we study a stronger notion, namely accountability. While verifiability requires merely that misbehavior should be detectable, accountability, in addition, ensures that specific misbehaving parties can be blamed. This is crucial in order to deter parties from misbehaving. Our definition of accountability for re-encryption RPC mix nets follows the one proposed in [14], which in turn is based on a general, domain independent definition of accountability proposed in [12].

We show that re-encryption RPC mix nets, with the fixes proposed by Khazaei and Wikström, enjoy a reasonable level of accountability. Essentially, our accountability definition requires that the multiset of plaintexts in the input ciphertexts should be the same as the multiset of plaintexts in the output. We show that, if in the output $k$ or more plaintexts have been modified (compared to the input), then this remains undetected with a probability of at most $(\frac{3}{4})^k$. Conversely, if manipulation is detected (which happens with a probability of at least $1-(\frac{3}{4})^k$), then at least one mix server can (rightly) be blamed for misbehaving.

This also shows that re-encryption RPC mix nets have the same level of accountability as Chaumian RPC mix nets. The proof of our result follows a similar line of reasoning as the one for Chaumian RPC mix nets in [14]. However, due to the different structures and cryptographic primitives used, the proofs (which are highly non-trivial) of course differ in the details.

As for privacy, we first introduce the notion of an *(essentially) semi-honest adversary*,[2] a new notion, which, in particular, was not considered in [14] for Chaumian RPC mix nets, but is key to our analysis of privacy of re-encryption RPC mix nets. Such an adversary, who may control some mix servers, does not deviate from the protocol in crucial aspects. Most importantly, an essentially semi-honest adversary/mix server shuffles and re-encrypts its input as expected (although the shuffle and other choices might not be random). For these adversaries, we make the following key observation. If an adversary does not follow the protocol in an essentially semi-honest way, then he will be caught with a probability of at least $1/4$. Hence, whenever an adversary decides to deviate from this semi-honest behavior, he knows that he takes a relatively high risk of being caught. So, as mentioned, when penalties are severe and/or reputation can be lost, this risk will in many cases be sufficiently high to deter adversaries from deviating from this semi-honest behavior. Therefore, a risk-avoiding adversary, that is an adversary who wants to avoid being caught, but otherwise might be willing to cheat if this goes unnoticed, must behave essentially semi-honestly. (Conversely, an essentially semi-honest adversary is also risk-avoiding.) Now, for such adversaries, we show that, under the common assumption that at least one mix serves is honest, re-encryption RPC mix nets provide a reasonable level of privacy, which, in fact, is quite close to the ideal case, where the adversary only learns the final output of the mix net. This result is optimal for re-encryption RPC mix nets in the sense that adversaries who are not willing risk being caught, cannot break privacy and those adversaries who attempt to break privacy (using, for example, the attacks by Khazaei and Wikström or any other attempt) will be caught with high probability.

2. While in the literature semi-honest adversaries are adversaries who follow the protocol honestly, essentially semi-honest adversaries might not. For simplicity of terminology, we nevertheless often refer to these adversaries as "semi-honest".

We note that Chaumian RPC mix nets, as shown in [14], provide privacy even for arbitrary polynomial-time aderversaries, rather than only for risk-avoiding adversaries.

The privacy proof for re-encryption RPC mix nets is, just as the proof of accountability for these mix nets, again far from trivial. We also emphasize that this proof differs substantially from the one for Chaumian RPC mix nets, as further explained in Section 7.

**Structure of this paper.** In the next section, re-encryption RPC mix nets are explained in more detail. We also present a formal model of these mix nets. Accountability for re-encryption RPC mixnets is analyzed in Section 4, with the definition presented in Section 3. In Section 5, we introduce and discuss the notions of essentially semi-honest and risk-avoiding adversaries mentioned above. We then define and analyze privacy for re-encryption RPC mixnets in Sections 6 and 7. We conclude in Section 8. Further details are provided in the appendix, with full details and proofs provided in our technical report [1].

## 2. Re-Encryption RPC Mix Net

In this section, we first recall the definition of a re-encryption RPC mix net [8] with improvements suggested in [10] and then sketch the formal model of this protocol, with full details provided in [1].

### 2.1. Description of the Protocol

**Cryptographic primitives.** The protocol uses a commitment scheme, which we assume to be computationally binding and perfectly hiding, with Pedersen commitments being an example [17] (but a computationally binding and computationally hiding scheme would do as well) and an IND-CPA secure, distributed public-key encryption scheme $\mathscr{S}$, where a set of parties (in our case the mix servers) independently generate their public and private key shares and the public key shares are then combined to obtain the public key. Ciphertexts obtained using this public key can be decrypted only when all the above parties participate in the decryption process (all private key shares are necessary for decryption). We assume that given a public key and a ciphertext, it can be decided efficiently whether the ciphertext in fact belongs to the space of possible ciphertexts (this typically means that one has to be able to decide whether certain group elements in fact belong to a given group). As usual for re-encryption mix nets, the encryption scheme $\mathscr{S}$ is also assumed to allow for re-encryption (with the appropriate hiding property, i.e. semantic security under re-encryption) and the following standard non-interactive proofs, where for some we require merely the soundness and completeness property, for others we in addition require the zero-knowledge property (NIZKPs) or also the knowledge extraction property:

– a NIZKP of knowledge of the private key share (for a given public key share),
– a NIZKP of knowledge of the plaintext (for a given ciphertext and a public key),
– a non-interactive proof of correct re-encryption (such a proof would typically simply reveal the random coins used for re-encryption),[3]
– a NIZKP of correct decryption (more precisely, for distributed decryption, one needs to prove that a given decryption share is correct).

Additionally, for the privacy result, we require that the used distributed encryption scheme allows for decryption share extractability;[4] this is straightforward for example in the case of ElGamal and, in fact, was used in the privacy proof of the Helios voting system [3]. We recall the precise security definitions for all mentioned cryptographic primitives in our technical report [1].

**Set of participants.** The set of participants of the protocol consists of a public, append-only *bulletin board* $\mathsf{B}$, *n senders* $\mathsf{S}_1, \dots \mathsf{S}_n$, *m mix servers* $\mathsf{M}_0, \dots, \mathsf{M}_{m-1}$, and some number of *auditors*. In the variant we consider here, we follow the most common practice to let mix servers play also the role of decryption servers. Alternatively, we could consider separate entities in the role of the decryption servers, which would not change the results provided in this paper.

The role of the auditors is to provide randomness for the auditing phase. Each auditor outputs a random bit string (more precisely, he first commits to his random bit string and later opens the commitment). An honest auditor outputs a bit string chosen uniformly at random. The bit strings produced by the auditors are combined to one bit string, say by the XOR operation. So, if at least one auditor is honest, the resulting bit string is chosen uniformly at random. We note that sometimes heuristics are implemented by which this assumption can be dropped (see [8]). However, as pointed out in [10], this may lead to some problems. In our formal model, we consider exactly one auditor, which we assume to be honest.

Typically, pairs of mix servers are audited. For the sake of presentation, it is therefore convenient to assume that one mix server performs two mixing steps. We will consider such mix servers in this paper.

Now, a re-encryption RPC mix net works in the following phases: setup, submit, input validation, mixing, and auditing, where the auditing may be carried out either before or after the decryption phase. It turns out that for the results presented in this paper, it does not matter which variant (auditing before or after decryption) we consider.

**Setup phase.** In this phase, every mix server $\mathsf{M}_i$ runs the key generation algorithm of $\mathscr{S}$ to generate a private/public key pair $(sk_i, pk_i)$. The public key $pk_i$ is then posted on the bulletin board along with a NIZKP of knowledge of the corresponding private key. The public keys $pk_0, \dots, pk_{m-1}$ of all the mix servers are then combined to obtain the encryption key $pk$ to be used by the senders.

3. We emphasize that here the zero-knowledge property and knowledge extraction are not necessary.

4. This property, roughly, states that, given a plaintext $m$, its encryption $c$, and all the private key shares but one, it is possible to compute all valid decryption shares, including the one for which the private key share is not given.

**Submit phase.** In this phase, every (honest) sender $\mathsf{S}_i$ chooses her input plaintext $m_i$ and encrypts it using the public key $pk$ to obtain her encrypted input. The sender also produces a NIZKP of knowledge of the plaintext. The ciphertext along with the zero-knowledge proof is posted by the sender on the bulletin board.

**Input validation.** It is checked whether the NIZKP of knowledge of the public keys are correct (otherwise, the protocol is aborted). It is also checked whether the ciphertexts are valid and whether their NIZKPs of knowledge of plaintexts are; invalid entries are eliminated (such entries might have been produced by dishonest senders). Moreover, for every set of entries with the same ciphertext, only one entry in this set is kept (the remaining ones are dropped). Note that input validation can be performed by any party.

The sequence of ciphertexts submitted by the senders and not rejected in the input validation phase constitute the input $C_0$ to the mixing phase, described below. Let $l$ be the number of entries in $C_0$.

**Mixing phase.** In what follows, we refer by $C_0[i]$ to the $i$-th element of the sequence $C_0$; similarly for other sequences. As mentioned above, the sequence of ciphertexts $C_0$ is the input to the first mix server $\mathsf{M}_0$ which processes it, as described below, and posts its output (which, again, is a sequence of ciphertexts) on the bulletin board $\mathsf{B}$. This output becomes the input to the next mix server $\mathsf{M}_1$, and so on. We will denote the input to the $j$-th mix server by $C_{2j}$ and its output by $C_{2j+2}$, reserving $C_{2j+1}$ for intermediate output (see Figure 1). Recall that one mix server performs two mixing steps.

The output $C_{2m}$ of the last mix server $\mathsf{M}_{m-1}$ is the output of the mixing stage. It is supposed to contain re-encryptions of the input $C_0$ (in random order).

The steps taken by every mix server $\mathsf{M}_j$ are as follows (see also Figure 1):

1. *Validation.* $\mathsf{M}_j$ checks whether its input $C_{2j}$ contains exactly $l$ entries and, if so, whether all these entries are valid ciphertexts (recall that we assume that this is possible). If this is not the case, the server stops without producing any output, the judge (see also below) will blame $M_{j-1}$ for misbehaving (if $j-1 \geq 0$), and the whole protocol is aborted.
2. *First mixing.* $\mathsf{M}_j$ uniformly at random chooses a permutation $\pi_{2j}$ of $\{1,\ldots,l\}$ and posts the sequence $C_{2j+1}$ of length $l$ on $\mathsf{B}$, where, for every $i \in \{1,\ldots,l\}$, $C_{2j+1}[i]$ is the result of the re-encryption of $C_{2j}[\pi_{2j}(i)]$.
3. *Second mixing.* $\mathsf{M}_j$, again, uniformly at random chooses a permutation $\pi_{2j+1}$ of $\{1,\ldots,l\}$ and posts the sequence $C_{2j+2}$ of length $l$ on $\mathsf{B}$, where $C_{2j+2}[i]$ is the result of the re-encryption of $C_{2j+1}[\pi_{2j+1}(i)]$. The sequence $C_{2j+2}$ is posted by $\mathsf{M}_j$ on $\mathsf{B}$.
4. *Posting commitments.* $\mathsf{M}_j$ posts two sequences of commitments on $\mathsf{B}$: commitments to the values $\pi_{2j}(1),\ldots,\pi_{2j}(l)$ and commitments to the values $\pi^{-1}_{2j+1}(1),\ldots,\pi^{-1}_{2j+1}(l)$ (in this order).

**Auditing phase.** The outputs of the mix servers are (partially) audited in order to detect potential misbehaviors. As

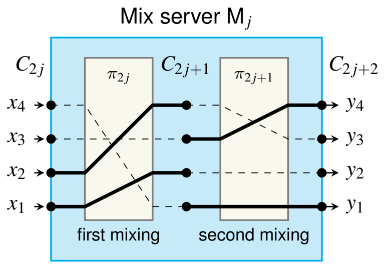

Figure 1. Mixing by $\mathsf{M}_j$. Solid bold lines represent audited links and dashed lines represent not audited links.

already noted, depending on the protocol variant, this phase may be performed before or after the decryption phase. In the former case, we can further consider a variant where auditing of a mix server is performed directly after the server has produced its output or a variant where all the mix servers are audited directly before the decryption phase. In either case, if auditing is done before decryption and misbehavior is detected, the decryption phase is not executed. As already noted, our results do not depend on which variant is chosen.

Independently of which variant is chosen, the steps taken in the audit for every individual mix server $\mathsf{M}_j$ are the same. First, using the randomness produced by the auditors, for an initial empty set $I_j$ and for every $i \in \{1,\ldots,l\}$ it is randomly decided, independently of other elements, whether $i$ is added to $I_j \subseteq \{1,\ldots,l\}$ or not. Provided that the random bit strings jointly produced by the auditors are distributed uniformly at random, the probably that $i$ belongs to $I_j$ is $\frac{1}{2}$. Now, for every $i \in \{1,\ldots,l\}$ the mix server $M_j$ does the following, depending on whether $i$ belongs to $I_j$ or not:

If $i \in I_j$, then the mix server $\mathsf{M}_j$ is supposed to open (by posting appropriate information on $\mathsf{B}$) the left link for $i$, i.e., $\mathsf{M}_j$ is supposed to open its $i$-th commitment from its first sequence of commitments, which should be a commitment to $\pi_{2j}(i)$. The mix server also has to post a non-interactive proof of correct re-encryption demonstrating that indeed $C_{2j+1}[i]$ is obtained by re-encrypting $C_{2j}[\pi_{2j}(i)]$. (As we will prove, this proof does *not* have to be zero-knowledge; it could simply reveal the random coins used to perform the re-encryption.)

If $i \notin I_j$, then, symmetrically, the mix server is supposed to open the right link for $i$, i.e., $\mathsf{M}_j$ is supposed to open its $i$-th commitment from its second sequence of commitments, which should be a commitment to $\pi^{-1}_{2j+1}(i)$. As before, the mix server also has to post a non-interactive proof of correct re-encryption demonstrating that indeed $C_{2j+2}[\pi^{-1}_{2j+1}(i)]$ is obtained by re-encrypting $C_{2j+1}[i]$.

An observer (or a judge) can now verify correctness of the data output by $\mathsf{M}_j$ in the audit phase. Firstly, the observer verifies that commitments are opened correctly. Secondly, one verifies that the opened indices (both from the first and the second sequence) do not contain duplicates (if they do, this means that the mix server has not committed to a permutation, but to some other, non-bijective function). Finally, one verifies the proofs of correct re-encryption. As pointed out in [10], the second step, which often has been omitted in implementations and is not mentioned in [8], is crucial for accountability and privacy.

The auditing described above guarantees that for a message from the sequence $C_{2j+1}$ either the connection to some message from $C_{2j}$ or to some message from $C_{2j+2}$ is revealed, but never both. Otherwise, an observer could follow the path of an input message to the corresponding output message (see also Figure 1 for an illustration). Nevertheless, some information about the link between the input and the output is revealed. For example, in Figure 1 an observer knows that the input values $x_1, x_2$ map to $y_2, y_3$ in some way and that $x_3, x_4$ map to $y_1, y_4$ in some way, and hence, for instance, she learns that $x_4$ does not map to $y_2$ or $y_3$.

**Decryption phase.** In this phase, the mix servers jointly decrypt every ciphertext from the output of the mixing phase (that is from $C_{2m}$) and provide NIZKP of correct decryption.

## 2.2. The Computational Model for Protocols

Our formal analysis is based on a computational model which follows the one used in [12], [13], which in turn is based on the IITM model [11], [15]. Below, we briefly recall this model.

A *program* is a probabilistic polynomial-time (ppt) interactive Turing machine (ITM) with named tapes (here also called channels). Two programs with channels of the same name but opposite directions (input/output) are connected by such channels, i.e., one program can send a message to the other one via the channel. Sending a message to another program via a channel triggers the other program. Only one program is active at a time.

By $\pi = \pi_1 \parallel \cdots \parallel \pi_n$ we denote a *system* of programs, also called a *process*, where the programs $\pi_i$ are connected via channels as described above. One of these programs is the master program (the one triggered first in a run and triggered if the active program did not produce output). In our modeling for re-encryption RPC mix nets, the scheduler (see below) will be the master. A process $\pi$ where all programs are given the security parameter $\ell$ is denoted by $\pi^{(\ell)}$.

A *protocol P* specifies a set of agents (also called parties or protocol participants) and the channels these agents can communicate over. Moreover, $P$ specifies, for every agent $a$, a set $\Pi_a$ of all programs the agent $a$ may run and a program $\hat{\pi}_a \in \Pi_a$, *the honest program of a*, i.e., the program that $a$ runs if $a$ follows the protocol.

Let $P$ be a protocol with agents $a_1, \ldots, a_n$. An *instance of P* is a process of the form $\pi = (\pi_{\mathsf{a}_1} \parallel \ldots \parallel \pi_{\mathsf{a}_n})$ with $\pi_{\mathsf{a}_i} \in \Pi_{a_i}$. An agent $\mathsf{a}_i$ is *honest* in the instance $\pi$, if $\pi_{\mathsf{a}_i} = \hat{\pi}_{\mathsf{a}_i}$. A *run of P* (with security parameter $\ell$) is a run $r$ of some instance $\pi$ of $P$ (with security parameter $\ell$); $\pi$ is in fact part of the description of $r$. An agent $\mathsf{a}_i$ is honest in a run $r$, if $\mathsf{a}_i$ is honest in the instance belonging to $r$. A *property $\gamma$ of P* is a subset of the set of all runs of $P$. By $\neg\gamma$ we denote the complement of $\gamma$.

It is straightforward to model re-encryption mix nets as a protocol, denoted by $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$, in the sense of this computational model (see [1] for the full model). The set of agents in the protocol in $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$ consists of the scheduler, the bulletin board, the auditor, the judge, $n$ senders, and the $m$ mix servers, where the first four agents are assumed to be honest, i.e., for each such agent $a$, $\Pi_a$ contains the honest program only. All other agents can run arbitrary ppt programs. Honest senders choose their plaintext input according to the distribution $\mu$. This models that the adversaries knows this distribution, which might not be completely true in reality but makes our results only stronger. The task of the scheduler is to trigger all agents (honest and dishonest) in the appropriate order according to the phases of the protocol. We assume that it is given information about which agents are honest and which are dishonest in order to schedule the agents in the appropriate way. In particular, the scheduler can schedule agents in a way advantageous for the adversary (dishonest agents) so that we obtain stronger security guarantees. For example, the scheduler first schedules honest senders to post their inputs on the bulletin board and then schedules dishonest senders. By this, the input of dishonest senders (the adversary) may depend on the input of honest senders.

In what follows, we use negligible and overwhelming functions in the security parameter, which are defined as usual. A function in the security parameter is *$\lambda$-bounded*, for $\lambda \in [0,1]$, if it is bounded by $\lambda$ plus some negligible function.

## 3. Defining Accountability of RPC Mix Nets

As mentioned in the introduction, our definition of accountability for re-encryption RPC mix nets follows the one proposed in [14], which in turn is based on a general domain independent definition of accountability proposed in [12]. As demonstrated in [12], accountability implies verifiability. Therefore, we mostly focus here on accountability, providing only a short discussion on verifiability.

The (general) definition of accountability of a protocol from [12] is stated with respect to a property $\gamma$ of the protocol, called the *goal*, a parameter $\lambda \in [0,1]$, and an agent $J$ of the protocol who is supposed to blame protocol participants in case of misbehavior (resulting in the violation of the goal $\gamma$). The agent $J$, sometimes referred to as a *judge*, can be a "regular" protocol participant or an (external) judge. It is worth noting that our results demonstrate that, for re-encryption RPC mix nets, every party (also external observers) can play the role of the judge, who needs to examine publicly available information only.

Informally speaking, accountability requires two conditions to be satisfied: i) $J$ (almost) never blames protocol participants who are honest, i.e., run their honest program (*fairness*); ii) if, in a run, the desired goal $\gamma$ of the protocol is not met—due to the misbehavior of one or more protocol participants—then $J$ should blame those participants who misbehaved, or at least some of them individually, where the probability that the desired goal is not achieved but $J$ nevertheless does not blame misbehaving parties should be bounded by $\lambda$ (*completeness*).

This general definition of accountability is instantiated in [14] for Chaumian RPC mix nets, by fixing the specific goal $\gamma$ and the parties who should be blamed if $\gamma$ is not achieved. We now provide a similar instantiation for re-encryption RPC mix nets.

**The goal.** As far as accountability (also verifiability) is concerned, we expect from a re-encryption RPC mix net that the output corresponds to the input, i.e., the plaintexts in the input ciphertexts and the plaintexts in the output of the mix net should be the same (as multisets). This, however, can be guaranteed only for input coming from honest senders. Dishonest senders, for example, might provide invalid NIZKPs of knowledge of the plaintexts, and hence, such input would be dropped during input validation. Below, we formally define this goal as a set of runs $\gamma_0$. Moreover, we generalize this goal by considering a family of goals $\gamma_k$, for $k \geq 0$, where $\gamma_k$ is achieved if the output corresponds to the input up to $k$ changed entries. In other words, for the goal $\gamma_k$ we tolerate up to $k$ manipulations. This is useful for the study of re-encryption RPC mix nets because, due to the nature of random partial checking, changing a small number of entries can go unnoticed with some probability. However, this probability should decrease very quickly with an increasing number $k$ of manipulated entries.

To formally specify the goal $\gamma_k$, we consider a run $r$ of an instance $\pi$ of $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$ (with $n$ senders). Let $s_1,\dots,s_l$ (for $l \leq n$) be those senders that are honest in $r$ (recall the definition of honest agents in a run from Section 2.2), $\vec{x} = x_1,\dots,x_l$ be the plaintext inputs of these senders in $r$ (chosen according to $\mu$), and $\vec{y} = y_1,\dots,y_p$ (with $p \leq n$) be the output of the mix net in $r$ (if any), i.e., the sequence of plaintexts output by the mix net after the decryption phase. We define $r$ to belong to $\gamma_k$ (in other words, $\gamma_k$ is achieved in $r$), if there exists a subsequence $\vec{x}'$ of $\vec{x}$ of size $l-k$ such that $\vec{x}'$, treated as a multiset, is contained in $\vec{y}$ (again, treated as a multiset), i.e., for each element $a$ of $\vec{x}'$, the number of $a$'s in $\vec{x}'$ is less than or equal to the number of $a$'s in $\vec{y}$. Hence, we require the output to contain $l-k$ elements from the honest input, while the remaining plaintexts, up to $n-(l-k)$, can be provided by the adversary. If in $r$ no final output was produced (because, for example, the process was stopped because a mix server refused to produce output), then $r$ does not belong to $\gamma_k$, i.e., $r$ does not achieve $\gamma_k$.[5]

**Parties to be blamed.** We require that, in a run $r$, if the goal $\gamma_k$ is not achieved ($r \notin \gamma_k$), then the judge should blame at least one mix server, i.e., post $\mathsf{dis}(\mathsf{M}_i)$ for at least one $i$ in $r$. This requirement also implies that a sender cannot break the goal $\gamma_k$: if $\gamma_k$ is not achieved, this must be due to a misbehaving mix server. This is important for the robustness of the mix net, as otherwise dishonest senders could spoil the mixing process.

5. Our proof of accountability shows that accountability holds true even for a slightly stronger goal, which says that *all* (but $k$) entries that made it through the input validation phase, have to make it to the output of the mix net. One can observe, using the cryptographic properties of the primitives, that honest entries will (except with negligible probability) always make it through the input validation. For Chaumian RPC mix nets this stronger goal cannot be achieved.

In the following formal definition of accountability for mix nets, we say that, if the judge posts $\mathsf{dis}(a)$, for some agent $a$, then the judge stated the *verdict* $\mathsf{dis}(a)$. Moreover, given an instance $\pi$ of a protocol $P$, we say that a verdict $\mathsf{dis}(a)$ *is true in* $\pi$ if and only if $a$ is not honest in $\pi$ (in the sense of Section 2.2). We write $\mathsf{Pr}[\pi^{(\ell)} \mapsto J : \mathsf{dis}(a)]$ to denote the probability that in a run of $\pi^{(\ell)}$ the judge $J$ states the verdict $\mathsf{dis}(a)$. We write $\mathsf{Pr}[\pi^{(\ell)} \mapsto \neg\gamma_k \wedge \neg(J : \mathsf{dis}(\mathsf{M}_i) \text{ for some } i)]$ to denote the probability that in a run of $\pi^{(\ell)}$ the goal $\gamma_k$ is not satisfied, i.e., the run does not belong to $\gamma_k$, and nevertheless $J$ does not state a verdict $\mathsf{dis}(\mathsf{M}_i)$ for any $i$. Both probabilities are taken over the runs of $\pi^{(\ell)}$, i.e., the random coins used by the agents in $\pi$.

**Definition 1.** ***(Accountability for RPC mix nets)*** *Let $P = P_{mix}(n,m,\mu)$ be a re-encryption RPC mix net protocol with an agent $J$ (the judge), $\lambda \in [0,1]$, and $k \geq 0$. We say that $P$* provides $\lambda$-accountability with tolerance $k$ (and w.r.t. $J$)*, if the following two conditions are satisfied.*

*(i)* (Fairness) *For all instances $\pi$ of $P$ and all verdicts $\mathsf{dis}(a)$ which are not true in $\pi$, the probability $\mathsf{Pr}[\pi^{(\ell)} \mapsto J : \mathsf{dis}(a)]$ is a negligible function in $\ell$.*

*(ii)* (Completeness) *For every instance $\pi$ of $P$, the probability $\mathsf{Pr}[\pi^{(\ell)} \mapsto \neg\gamma_k \wedge \neg(J : \mathsf{dis}(M_i) \text{ for some } i)]$ is a $\lambda$-bounded function in $\ell$.*

The above definition requires that the judge never (more precisely, only with negligible probability) blames mix servers that behave honestly, i.e., run their honest program. It also requires that the probability that the goal $\gamma_k$ is not satisfied, and hence, more than $k$ inputs of honest senders have been manipulated or no output was produced by the mix net, but the judge nevertheless does not blame any single mix server, is bounded by $\lambda$. We will see that for re-encryption RPC mix nets (the optimal/minimal) $\lambda$ will be bigger than 0. This is unavoidable because of the nature of random partial checking, some misbehavior might go unnoticed with some non-negligible probability. One of the important contributions of this work is to determine the optimal $\lambda$, and hence, precisely measure the level of accountability re-encryption RPC mix nets provide.

**Verifiability.** Accountability is a stronger property than verifiability and subsumes it [12]: While for verifiability one requires protocol participants only to be able to see whether something went wrong or not, accountability additionally demands that, if something went wrong, it is possible to blame specific misbehaving parties. Accountability therefore provides a strong incentive for parties (mix servers in our case) to carry out correct computations, which is of high practical importance. This cannot be said for verifiability alone. Accountability, as we will see, is a fundamental requirement that justifies the notion of risk-avoiding adversaries (see Section 5).

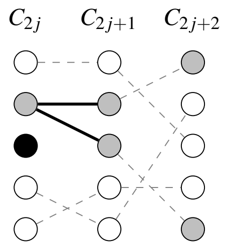

Figure 2. Example attack on accountability detected only with probability $\frac{1}{4}$. Both gray nodes in the middle colum are re-encryptionis of the same ciphertext in the left one. The ciphertext represented by the black node is dropped.

## 4. Analysis of Accountability of Re-Encryption RPC Mix Nets

In this section, we provide formal results for the level of accountability (and hence, verifiability) re-encryption RPC mix nets provide. This is the first rigorous analysis of accountability/verifiability for re-encryption RPC mix nets in the literature. The level of accountability is reasonably high and provides a strong deterrent for potentially malicious mix servers.

We start, in Section 4.1, with a description of some attacks on the accountability/verifiability of re-encryption RPC mix nets. We then present our formal results, which show that these mix nets have a good level of accountability/verifiability. In particular, they show that there are no worse attacks than those described in Section 4.1.

### 4.1. Attacks

The most obvious way in which a mix server can cheat is when, instead of performing re-encryption, the mix server replaces an input ciphertext by an arbitrary other ciphertext, possibly without preserving the plaintext. This kind of cheating is (not) detected with probability $\frac{1}{2}$, and if the mix server cheats in this way for $k+1$ input ciphertexts of honest senders at the same time (and hence, violates $\gamma_k$), its probability of not being caught is $(\frac{1}{2})^{k+1}$. There are, however, more subtle ways of cheating which result in dishonest mix servers being caught less likely (see [10]). For example, in the attack illustrated in Figure 2 a dishonest mix server $\mathsf{M}_j$, for two positions $p$ and $q$ in its intermediate sequence $C_{2j+1}$ of ciphertexts, sets both $C_{2j+1}[p]$ and $C_{2j+1}[q]$ to be re-encryptions of the same entry $C_{2j}[\pi_{2j}(p)]$ (an honest $\mathsf{M}_j$ would set $C_{2j+1}[q]$ to be a re-encryption of $C_{2j}[\pi_{2j}(q)]$). Moreover, in its first sequence of commitments, both at positions $p$ and $q$ it commits to the value $\pi_{2j}(p)$ (an honest $\mathsf{M}_j$ would at position $q$ commit to $\pi_{2j}(q)$). As a result of this manipulation, one of the entries from $C_{2j}$ is dropped (the black node in the example) and substituted by another one (the gray entry). This attack can be detected only with probability $\frac{1}{4}$, because detection requires that both $p$ and $q$ are audited to the left, i.e., both $p$ and $q$ belong to $I_j$. Performing the attack on $k+1$ different pairs of ciphertexts (by the same mix server or different mix servers) results in the violation of $\gamma_k$ and this remains undetected with probability $\left(\frac{3}{4}\right)^{k+1}$.

### 4.2. Formal Analysis of Accountability

We now state and prove the precise level of accountability/verifiability re-encryption RPC mix nets have. While from the above it is clear that the probability of more than $k$ manipulations of honest entries (violation of $\gamma_k$) going unnoticed may be $(\frac{3}{4})^{k+1}$, we prove that this probability is not higher, and hence, there are no worse attacks.

**Security assumptions.** Recall that we assume that the scheduler, the judge, the auditor, and the bulletin board are honest. However, none of the mix servers nor the senders are assumed to be honest. The assumptions about the primitives used in a re-encryption RPC mix nets have already been summarized in Section 2.1. However, for accountability, weaker assumptions are sufficient. It suffices if the commitment scheme is computationally binding. The distributed public-key encryption does not have to be IND-CPA secure, but has to guarantee that encrypting the same message twice yields different ciphertexts with overwhelming probability (this is implied by IND-CPA security). The non-interactive proofs do not have to be zero-knowledge.

Now, the following theorem holds true for re-encryption RPC mix nets, independently of whether auditing is done before or after decryption.

**Theorem 1.** *Let $P = P_{mix}(n,m,\mu)$ be a re-encryption RPC mix net. Then, under the above security assumptions, $P$ provides $\lambda_k$-accountability with tolerance $k$, where $\lambda_k = \left(\frac{3}{4}\right)^{k+1}$; $P$ does not provide $\lambda$-accountability for any $\lambda < \lambda_k$, i.e., $\lambda_k$ is optimal.*

This theorem implies that even if all mix servers are dishonest, the probability that more than $k$ inputs of honest voters have been manipulated, but the judge nevertheless does not blame any mix server, is bounded by $\left(\frac{3}{4}\right)^{k+1}$. For example, the probability that more than 10 manipulations go undetected is less than 4.5%. Moreover, if manipulation is detected, at least one mix server is blamed (and rightly so) for its misbehavior.

As already mentioned in the introduction, this result in particular shows that re-encryption RPC mix nets have the same (good) level of accountability as Chaumian RPC mix nets. The (highly non-trivial) proof of Theorem 1, which is provided in our technical report [1], follows a similar line of reasoning as the one for Chaumian RPC mix nets [14]. However, due to the different structures and cryptographic primitives (re-encryption and distributed decryption instead of just nested encryption) used, the proofs differ in the details.

## 5. Risk-Avoiding and Essentially Semi-Honest Adversaries

As observed in [10], and partly already in [18], in the general case, that is, for arbitrary probabilistic polynomial-time adversaries, there are attacks on the privacy of votes for re-encryption RPC mix nets, which allow the adversary to see how one or more voters voted. These attacks use homomorphic properties of the encryption scheme and generate collisions (links that point to the same entry) as illustrated in Figure 2. In all these attacks, the adversary, however, risks to be caught cheating with a probability of at least $1/4$. As mentioned before, this risk of being caught should deter mix servers from dishonest behavior in many real-life applications (elections), as mix servers that are caught cheating would face severe penalties and, maybe just as deterrent, lose reputation.

As already mentioned in the introduction, this motivates us to consider "risk-avoiding" adversaries. Such adversaries were first introduced in [14]. We recall the definition of risk-avoiding adversaries below.

The concept of risk-avoiding adversaries is, however, not sufficient for the privacy analysis of re-encryption RPC mix nets. We rather need an additional important notion, namely essentially semi-honest adversaries. Below we formally introduce this new class of adversaries and show that whenever an adversaries deviates from the essentially semi-honest behavior, then he knows that he will be caught cheating with high probability. From this we obtain that risk-avoiding and essentially semi-honest adversaries are tightly connected, which in turn is an important tool to prove that re-encryption RPC mix nets provide a reasonable level of privacy for risk-avoiding adversaries (see Section 7).

We believe that the new notion of essentially semi-honest behavior and its connection to risk-avoiding adversaries might be relevant beyond the particular application to re-encryption RPC mix nets.

**Essentially semi-honest adversaries.** One key observation is that if an adversary does not follow the protocol in an essentially semi-honest way (as defined next), then he will always be caught with a probability of at least $1/4$, where, as before, for a given instance $\pi$ of $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$ or a run of this instance, the adversary is the set of all dishonest agents (in the sense defined in Section 2.2) in $\pi$.

Intuitively, we say that a mix server behaves *essentially semi-honestly* in a run $r$, if the server does not deviate from the protocol in crucial aspects in $r$. Most importantly, an essentially semi-honest mix server shuffles and re-encrypts its input as expected, i.e., input ciphertexts are re-encrypted and there exists a permutation such that the re-encrypted ciphertexts are shuffled according to this permutation. Now, an adversary is called *essentially semi-honest* in a run $r$, if every dishonest mix server (all of them are part of the adversary) behaves essentially semi-honestly in this run. Below, we provide a formal definition of essentially semi-honest behavior.

Let $\pi$ be an instance of the protocol $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$. Let us recall that an instance is a combination of programs of all parties, including potentially dishonest ones (the adversary). Let $r$ be a run of $\pi$.

Now, we say that the $j$-th mix server behaves *essentially semi-honestly* in the run $r$, if this mix server produces correct output in the setup phase, the mixing phase, and the decryption phase, that is:

(a) $\mathsf{M}_j$ outputs its public key share along with a valid NIZK proof of knowledge of the private key share, where "valid proof" (here and below) means that the proof is accepted by the judge in $r$ (see the auditing phase described in Section 2.1).

(b) If $\mathsf{M}_j$ obtains a valid input $C_{2j}$ (consisting of valid ciphertexts), then $\mathsf{M}_j$ outputs $C_{2j+1}$ and $C_{2j+2}$ such that the sequences $C_{2j}$, $C_{2j+1}$, and $C_{2j+2}$ all have the same length $l$. Moreover, $\mathsf{M}_j$ outputs two sequences of commitments to $l$ values each. If $\mathsf{M}_j$ is audited $\mathsf{M}_j$ provides valid non-interactive proofs of correct re-encryption and provides valid openings to the commitments that need to be opened without collisions. In other words, in the mixing phase in run $r$, $\mathsf{M}_j$ produces output that the judge approves.

(c) If $\mathsf{M}_j$ obtains a valid input sequence $C_{2j}$ of length $l$ (consisting of valid ciphertexts), then $\mathsf{M}_j$ outputs $C_{2j+1}$ and $C_{2j+2}$ such that there exist permutations $\pi_{2j}$ and $\pi_{2j+1}$ on the set $\{1,\ldots,l\}$ such that $C_{2j+1}[i]$ is a re-encryption of $C_{2j}[\pi_{2j}(i)]$ and $C_{2j+2}[i]$ is a re-encryption of $C_{2j+1}[\pi_{2j+1}(i)]$.

(d) In the decryption phase, for every ciphertext $m$ to be decrypted, $\mathsf{M}_j$ outputs a decryption share $h$ for this ciphertext and the (common) public key, along with a valid NIZKP of correctness of the decryption share.

We say that an adversary behaves *essentially semi-honestly* in a run, if every dishonest mix server (which is controlled by the adversary) behaves essentially semi-honestly in this run; honest mix servers obviously behave essentially semi-honestly. For simplicity, in what follows we simply refer to essentially semi-honest adversaries as *semi-honest adversaries*.

Now, the following lemma shows that under any circumstances not being semi-honest is always risky. For an instance $\pi^{(\ell)}$ of $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$ with security parameter $\ell$, let $B_j^{(\ell)}$ denote the set of runs of $\pi^{(\ell)}$ (an event) where $\mathsf{M}_j$ is blamed by the judge and let $NSH_j^{(\ell)}$ be the set of runs of $\pi^{(\ell)}$ where $\mathsf{M}_j$ does not behave semi-honestly. We say that a family $G = \{G^{(\ell)}\}_\ell$ of events is overwhelming if $\Pr[G^{(\ell)}]$ is an overwhelming function in $\ell$.

**Lemma 1.** *There exists an overwhelming family $G$ such that, for all $\ell \geq 0$, we have that* $\Pr\left[B_j^{(\ell)} \mid NSH_j^{(\ell)} \cap G^{(\ell)}\right] \geq \frac{1}{4}$ *if* $\Pr\left[NSH_j^{(\ell)} \cap G^{(\ell)}\right] > 0$.

The interpretation of Lemma 1 is as follows: except for some negligible set of runs, whenever $\mathsf{M}_j$ decides to not shuffle and re-encrypt the ciphertexts in the expected way, then (he knows that) in the audit phase he will be caught with a probability of at least $1/4$. In other words, there is no way for $\mathsf{M}_j$ to outsmart the system by not performing the expected task but getting caught with probably $< 1/4$. In our technical report [1], we present a lemma which implies Lemma 1 and makes this point even more explicit.

Therefore, if for an adversary (which includes all dishonest mix servers) the risk of being caught, and hence, the risk of facing severe penalties and loss of reputation, is too high, such a "risk avoiding" adversary would behave semi-honestly, i.e., would not deviate from the expected behavior. Of course, a mix server could flip a coin in order to decide whether to behave semi-honestly or not in certain stages of the run. This would bring the overall risk of being caught down, but only in a very artificial way. Indeed, for an adversary that decided not to take the (high) risk of being caught in the first place, such a behavior appears very unreasonable. If the coin flip made him behave in a non semi-honest way, he knows that he then will be caught with high probability, a risk the adversary wanted to avoid. Hence, it seems reasonable to assume that an adversary for whom the risk of being caught once he deviates from semi-honest behavior is too high (and this risk is always at least $1/4$), should never decide to deviate from the semi-honest behavior. Thus, it is reasonable to expect that such an adversary behaves semi-honestly with overwhelming probability. We will now see that this class of adversaries coincides with the class of risk-avoiding adversaries.

**Risk-avoiding adversaries.** For an instance $\pi$ of $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$, we say that the adversary in $\pi$ (consisting of all dishonest parties in $\pi$) is *risk-avoiding* in $\pi$, if the probability that $\pi$ produces a run where the judge blames a dishonest party is negligible as a function in the security parameter $\ell$. This class of adversaries was introduced in [14]. We now link this notion to semi-honest behavior.

Note that, in general, risk-avoiding adversaries do not need to behave semi-honestly: if in a protocol misbehavior goes unnoticed (because there are no, or insufficient, detection mechanisms), then an adversary can freely depart from the protocol and still never get blamed. For re-encryption RPC mix nets, however, by Lemma 1, we show that risk-avoiding adversaries are forced to behave semi-honestly:

**Lemma 2.** *The adversary in $\pi$ is risk-avoiding if and only if the set of runs of $\pi$ in which he behaves semi-honestly has overwhelming probability.*

In the proof of the privacy result (see Section 7), we will also use the following, technical result which takes the statement of Lemma 2 one step further: while Lemma 2 guarantees that a risk-avoiding adversary is semi-honest, and hence, shuffles and re-encrypts in every mixing step, the following result states in addition that, when such a system is simulated, suitable permutations can be extracted, by means of rewinding, by the simulator (see our technical report [1] for the proof).

**Lemma 3.** *Let $\pi$ be an instance of $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$ such that the adversary in $\pi$ is risk-avoiding. There exists a simulator $T$ which faithfully*[6] *simulates $\pi$ and additionally outputs (on some distinct tape) permutations $\pi'_0, \dots \pi'_{2m-1}$ such that in an overwhelming set of runs, for each mix server $M_j$, the permutations $\pi'_{2j}, \pi'_{2j+1}$ satisfy Condition (c) of semi-honest runs, by which we mean that the properties stated for $\pi_{2j}$ and $\pi_{2j+1}$ in Condition (c) are satisfied for $\pi'_{2j}$ and $\pi'_{2j+1}$, respectively.*

6. i.e., the simulated runs of $\pi$ are exactly the same as the runs of the original system $\pi$; note that the adversary (who subsumes all dishonest parties in $\pi$) is simulated in a black-box manner, while the honest parties in $\pi$ are explicitly given.

## 6. Definition of Privacy for RPC Mix Nets

As already mentioned in the introduction, we use a definition of privacy which has been used in the context of e-voting before (see, e.g., [13]) and which has also been employed for the analysis of Chaumian RPC mix nets in [14].

As opposed to (very strong) simulation-based definitions, see, for instance, [9] and a related game-based definition in [3], the above mentioned definition allows one to measure the level of privacy a protocol provides. The ability to measure the level of privacy is absolutely essential in the context of RPC mix nets, because such protocols do not achieve perfect privacy: it is in the very nature of these protocols that the adversary can learn *some* information from a protocol run. Therefore, it is essential to be able to precisely tell *how much* he can actually learn.

More specifically, in the context of e-voting, the privacy definition we adopt formalizes the inability of an observer to distinguish whether some voter $\mathsf{v}$ (called the voter under observation) voted for candidate $j$ or candidate $j'$, when running her *honest* voting program. Applied to RPC mix nets, we formalize privacy as the inability of an adversary to distinguish whether some sender under observation submitted plaintexts $p$ or $p'$, when running her honest program.

For studying privacy, we consider the protocol $\mathsf{P}^{j}_{\mathsf{mix}}(n,m,\mu)$, which coincides with $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$ but where the $j$-th mix server is assumed to be honest, all other mix servers may be dishonest. Among the $n$ senders, we consider one sender $\mathsf{s}$ to be under observation. (The task of the adversary is to figure out whether this sender sent plaintext $p$ or $p'$.)

Now, given a sender $\mathsf{s}$ and a plaintext $p$, the protocol $\mathsf{P}^{j}_{\mathsf{mix}}(n,m,\mu)$ induces a set of instances of the form $(\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*)$ where $\hat{\pi}_{\mathsf{s}}(p)$ is the honest program of the sender $\mathsf{s}$ under observation that takes $p$ as its unencrypted input and $\pi^*$ is the composition of programs of the remaining parties (scheduler, auditor, judge, senders, mix servers), one program $\pi \in \Pi_a$ for each party $a$. Recall that the honest voters in $\pi^*$ vote according to the probability distribution $\mu$. Also recall that according to the definition of $\mathsf{P}^{j}_{\mathsf{mix}}(n,m,\mu)$, if $a$ is the scheduler, the auditor, the judge, or the $j$-th mix server, then $\Pi_a$ contains the honest program of that party only, as they are assumed to be honest. All other parties are potentially dishonest and may run arbitrary (adversarial) probabilistic polynomial-time programs. For such general adversaries who do not avoid accusations by the judge, privacy of re-encryption RPC mix nets cannot be guaranteed, as demonstrated by the attacks in [10], [18]. In order to define privacy w.r.t. risk-avoiding adversaries, we simply restrict the set of programs $\pi^*$ to those that are risk-avoiding (see Section 5). In a system $(\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*)$ with a sender $\mathsf{s}$ under observation, $\pi^*$ is required to be risk avoiding for all choices of the plaintexts $p$ in the considered space of plaintexts.

Privacy for re-encryption RPC mix nets is now defined as follows, where we use the following notation: $\Pr[(\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*)^{(\ell)} \mapsto 1]$ denotes the probability that the adversary (i.e., some dishonest agent) writes the output 1 on some dedicated channel in a run of $\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*$ with security parameter $\ell$ and some plaintext $p$. The probability is over the random coins used by the agents in $\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*$.

**Definition 2.** *For $P^j_{mix}(n,m,\mu)$ as before let* $\mathsf{s}$ *be the sender under observation, $l < n-1$, and $\delta \in [0,1]$. We say that* $\mathsf{P}^j_{\mathsf{mix}}(n,m,\mu)$ with $l$ honest senders achieves $\delta$-privacy (w.r.t. risk-avoiding adversaries), *if*

$$\Pr[(\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*)^{(\ell)} \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{s}}(p') \parallel \pi^*)^{(\ell)} \mapsto 1] \qquad (1)$$

*is $\delta$-bounded as a function of the security parameter $\ell$, for all valid input plaintexts $p, p'$ and all (risk-avoiding) programs $\pi^*$ of the remaining parties such that (at least) $l$ senders are honest in $\pi^*$.*

Since $\delta$ typically depends on the number $l$ of honest senders, privacy is formulated w.r.t. this number. Note that a smaller $\delta$ means a higher level of privacy. However, $\delta$ cannot be 0, not even in an ideal protocol, as detailed in the following subsection: there is, for example, a non-negligible chance that all honest senders sent the same message. In this case, the adversary knows the message sender $\mathsf{s}$ has sent, and hence, can easily determine whether $\mathsf{s}$ sent $p$ or $p'$.

**Privacy for the Ideal Mix Net Protocol.** Before we state the level of privacy provided by re-encrypted RPC mix nets, we first briefly recall results for the ideal mix net from [14], where the optimal $\delta^{id}_{l,\mu}$ is determined in this case. The level of privacy for re-encryption RPC mix nets can be expressed in terms of this value (see Section 7).

In the ideal mix net, the senders submit their input plaintexts on a direct channel to the ideal mix net. The ideal mix net then immediately outputs the submitted messages after having applied a random permutation. Honest senders choose their inputs according to the distribution $\mu$.

The level of privacy provided by the ideal mix net depends on the number $l$ of honest senders and the probability distribution $\mu$ on valid input plaintexts. To define $\delta^{id}_{l,\mu}$, we need the following terminology. Let $\{p_1,\dots,p_k\}$ be the set of valid plaintexts. Since the adversary knows the input plaintexts of the dishonest senders, he can simply filter out these plaintexts from the final output and obtain the so-called *pure output* $\vec{r} = (r_1,\dots,r_k)$ of the protocol, where $r_i$, $i \in \{1,\dots,k\}$, is the number of times the plaintext $p_i$ occurs in the output after having filtered out the dishonest inputs. Note that, if $l$ is the number of honest senders, then $r_1 + \dots + r_k = l+1$ ($l$ honest senders plus the sender under observation).

We denote by *Out* the set of all pure outputs. Let $A^i_{\vec{r}}$ denote the probability that the choices made by the honest senders yield the pure output $\vec{r}$, given that the sender under observation submits $p_i$. Further, let $M_{j,j'} = \{\vec{r} \in Out : A^j_{\vec{r}} \leq A^{j'}_{\vec{r}}\}$. Now, the intuition behind the definition of $\delta^{id}_{l,\mu}$ is as follows: If the observer, given a pure output $\vec{r}$, wants to decide whether the observed sender submitted $p_j$ or $p_{j'}$, the best strategy of the observer is to opt for $p_{j'}$ if $\vec{r} \in M_{j,j'}$. This leads to the following level of privacy provided by the ideal mix net protocol with $l$ honest senders and the probability distribution $\mu$: $\delta^{id}_{l,\mu} = \max_{j,j' \in \{1,\dots,k\}} \sum_{\vec{r} \in M_{j,j'}} (A^{j'}_{\vec{r}} - A^j_{\vec{r}})$, with example values depicted in Figure 3. (Note that $A^{j'}_{\vec{r}}$ - $A^j_{\vec{r}}$ depends on $l$ and $\mu$.)

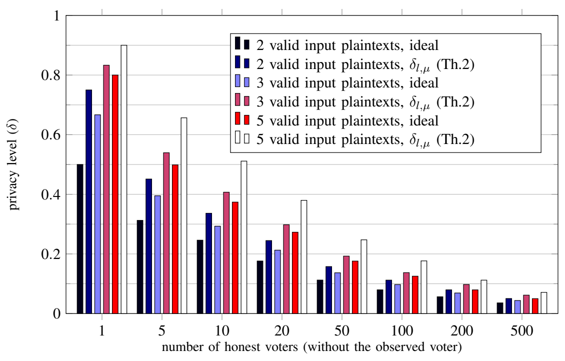

Figure 3. Level of privacy ($\delta_{l,\mu}$) for $\mathsf{P}^j_{\mathsf{mix}}(n,m,\mu)$ w.r.t. risk-avoiding adversaries and in the ideal case $\delta^{id}_{l,\mu}$, uniform distribution of input plaintexts. These figures have been obtained by straightforward calculations using the $\delta$-formulas as provided in the theorems. For non-uniform distributions, $\delta_{l,\mu}$ is close to ideal as well.

## 7. Analysis of Privacy of Re-Encryption RPC Mix Nets

We now provide the formal analysis of the level of privacy re-encryption RPC mix nets provide in the case of a risk-avoiding adversary. We note that in our analysis of privacy, we assume merely that one of the mix servers is honest; clearly, if all mix servers are dishonest there cannot be any privacy.

For the following result, our cryptographic assumptions are as described in Section 2.1. The result holds true independently of whether auditing of the mix servers is done before or after the decryption phase.

**Theorem 2.** *The protocol $P^j_{mix}(n,m,\mu)$ with $l$ honest senders achieves $\delta_{l,\mu}$-privacy w.r.t. risk-avoiding adversaries, where*

$$\delta_{l,\mu} = \frac{1}{2^l} \cdot \sum_{i=0}^{l} \binom{l}{i} \delta^{id}_{i,\mu} \quad .$$

*Moreover, $\delta_{l,\mu}$ is optimal, i.e., this protocol does not achieve $\delta$-privacy w.r.t. risk-avoiding adversaries for any $\delta < \delta_{l,\mu}$.*

Example values for $\delta_{l,\mu}$ are depicted in Figure 3. As can be seen, for risk-avoiding adversaries, the level of privacy provided by re-encryption RPC mix nets is only slightly worse than the level of privacy in the ideal mix net, even though only one honest mix server is assumed to be honest.

As mentioned before, the assumption that adversaries are risk-avoiding is necessary because otherwise privacy could be broken, as the attacks by Khazaei and Wikström on re-encryption RPC mix nets illustrate [10]. However, as we show in Lemma 2, being risk-avoiding is equivalent to behaving semi-honestly, and by Lemma 1, it follows that if an adversary deviates from semi-honest behavior, he (knows that he) will be caught cheating with high probability. So, altogether our results say that risk-avoiding adversaries cannot break privacy and those adversaries who (attempt to) break privacy will be caught with high probability. We note that while this provides a reasonable level of privacy, Chaumian RPC mix nets provide close to ideal privacy even for general polynomial-time adversaries, as was shown in [14].

We note that the proofs of privacy for re-encryption RPC mix nets on the one hand and Chaumian RPC mix nets on the other hand differ substantially as well: The main reason for this difference is that the encryption scheme used in re-encryption mix nets allow for re-encryption, and hence, merely provide IND-CPA-security (as opposed to IND-CCA2-security in the case of Chaumian mix nets); this is also the main reason for the weaker level of privacy. Consequently, we need the new machinery introduced in Section 5, in particular the notion of essentially semi-honest behavior and its connection to risk-avoiding adversaries, and the assumption that the adversary is risk-avoiding to establish the privacy result. On the technical level, when it comes to carrying out the reduction in the privacy proof, one cannot simply use idealized encryption and decryption oracle queries to simulate the decryption step performed by the mix net as done in [14]. Instead, the plaintexts have to be extracted from the input of the mix net and carefully traced throughout the (partially dishonest) mixing process. Note also that, unlike Chaumian RPC mix nets, re-encryption RPC mix nets use distributed decryption.

## 7.1. Proof of Theorem 2

Let us consider an instance of the system $\mathsf{P}^j_{\mathsf{mix}}(n,m,\mu)$ with $l$ honest senders (where, by the definition of $\mathsf{P}^j_{\mathsf{mix}}$ the $j$-th mix server is honest). We will represent such an instance as $A \parallel P$, where $P$ represents all the honest programs, while the dishonest parties are represented by $A$, the risk-avoiding adversary. We have to show that for all valid input plaintexts $p$ and $p'$, we have that

$$|\mathsf{Pr}[(A \parallel P(p))^{(\ell)} \mapsto 1] - \mathsf{Pr}[(A \parallel P(p'))^{(\ell)} \mapsto 1]|$$

is a $\delta_{l,\mu}$-bounded function in the security parameter $\ell$, where $P(p)$ means that the sender under observation uses $p$ as its plaintext. We denote this function by $\mathsf{Adv}^{priv}_{A,P,p,p'}(\ell)$ and call it the advantage of $A$.

We first define what we call audit groups for an overwhelming set of runs.

Consider a run of the instance $A \parallel P$ which is semi-honest and for which the extractor from Lemma 3 can extract correct permutations, namely permutations that satisfy (c) in the definition of semi-honest behavior. This set of runs has overwhelming probability.

For such a run, we can split the input entries into two groups: those for which $\mathsf{M}_j$ opens the left link and those for which $\mathsf{M}_j$ opens the right link. More precisely, each input entry $C_0[i]$ is linked to $C_{2j+1}[i']$ (an entry in the middle column of $\mathsf{M}_j$) with $i = (\pi_{2j} \circ \pi_{2j-1} \circ \cdots \circ \pi_0)(i')$, where $\pi_k$ are permutations extracted from the run,[7] (and thus satisfying condition (c) of semi-honest runs) and $\circ$ denotes function composition ($(f \circ g)(x) = g(f(x))$). Note that, by the definition (c) of semi-honest runs, $C_{2j+1}[i']$ is a re-encryption of $C_0[i]$. Now, if the auditors request $\mathsf{M}_j$ to open the left link for the index $i'$, then we say that $i$ belongs to the left audit group $I_L$; otherwise we say that $i$ belongs to the right audit group $I_R$.

We further say that $I_L$ is the audit group of the sender under observation if the (index of the) entry of this sender belongs to $I_L$. Similarly for $I_R$.

From the program $A$, we derive a program $A^*$ in the following way: $A^*$ simulates $A$ and also runs the extractor from Lemma 3 in order to extract permutations for the mix servers subsumed by $A$ (as just mentioned, by Lemma 3, this can be done in such a way that, with overwhelming probability, for the extracted permutations, Condition (c) in the definition of semi-honest behavior is true). This allows $A^*$ to determine the audit group of the sender under observation and learn which (encrypted and then later decrypted) output entries are linked to this group (without knowing specifically which output entry is linked to which sender in this group). Let $Q$ denote the multi-set of all the plaintexts (decrypted entries) linked to this group. For example, if $x_3$ in Figure 1 is the (re-encrypted) entry of the sender under observation ($A^*$, knowing the extracted permutations, knows at which position the entry of the sender under observation is delivered), then $\{y_1, y_4\}$ are entries of the senders from the audit group of the sender under observation. This group of entries, given the extracted permutations, can be easily linked to the output of the mix net, when they get decrypted and the multi-set $Q$ is given.

Now, $A^*$ accepts the run (outputs 1) if and only if the following is true: the probability that the choices of $|Q|-1$ honest senders (made according to the probability distribution $\mu$) yield $Q$, given that the sender under observation chooses $p$, is bigger than the probability that the choices of $|Q|-1$ honest senders yield $Q$, given that the sender under observation chooses $p'$.

In the following lemma, we write $f \leq_{neg} f'$, if there exists a negligible function $\nu(\ell)$ such that $f(\ell) \leq f'(\ell) + \nu(\ell)$ for all $\ell$. Later we also use $f =_{neg} f'$ to mean $f \leq_{neg} f'$ and $f' \leq_{neg} f$. The lemma says that the advantage of $A$ is not bigger than the advantage of $A^*$.

**Lemma 4.** *For a risk-avoiding adversary $A$ and for all valid $p$ and $p'$, we have that*

$$\mathsf{Adv}^{priv}_{A,P,p,p'} \leq_{neg} \mathsf{Adv}^{priv}_{A^*,P,p,p'} \ .$$

The proof of this lemma is postponed to Section 7.2.

Now, the proof of Theorem 2 proceeds as follows. By Lemma 4, it suffices to prove that $\mathsf{Adv}^{priv}_{A^*,P,p,p'} \leq_{neg} \delta_{l,\mu}$.

7. Formally, to determine these permutations, one needs to consider the corresponding run of the system $A \parallel P$ simulated by the simulator from Lemma 3.

By Lemmas 2 and 3, there is an overwhelming set *SHP* of runs of $A^* \parallel P$ such that these runs are semi-honest and such that the permutations $A^*$ extracts from the dishonest mix servers satisfy Condition (c) of semi-honest behavior. For these runs, the mixing runs through successfully (as the runs are semi-honest) and the multi-set $Q$ contains the plaintexts chosen by the senders in the audit group of the sender under observation (as the permutations satisfy (c)).

By the above, it is easy to see that, the computations carried out by $A^*$ yield the constant from the theorem, i.e., $\mathsf{Adv}^{priv}_{A^*,P,p,p'} =_{neg} \delta_{l,\mu}$. Indeed, $i$ in the definition of $\delta_{l,\mu}$ represents the number of entries of honest senders that are, in a given run of the system, in the same audit group as the entry of the sender under observation. We consider all the possible cases, from $i = 0$ (the entry of the sender under observation is alone in its audit group, and hence, the adversary can easily see her choice) to $i = l$ (all the honest entries are in the same group as the entry of the sender under observation; in this case, privacy of the sender under observation is maximally protected). The probability that $i$ honest senders belong to the same audit group as the sender under observation is $\binom{l}{i}\frac{1}{2^l}$, as it is decided independently for every honest entry if it belongs to the audit group of the sender under observation or not. Moreover, under the condition that the sender under observation is in an audit group with $i$ honest senders, the situation corresponds to that of the ideal mix net with $i$ honest senders. Hence, in this case, the level of privacy is $\delta^{id}_{i,\mu}$. Moreover, for the given audit group, $A^*$ follows the best strategy as described for the ideal case (see Section 6). Therefore, we in fact obtain $\mathsf{Adv}^{priv}_{A^*,P,p,p'} =_{neg} \delta_{l,\mu}$.

Optimality of $\delta_{l,\mu}$ is obvious now since if $A$ is the benign adversary (which in particular is risk-avoiding), we have, by the above, that $\mathsf{Adv}^{priv}_{A^*,P,p,p'} =_{neg} \delta_{l,\mu}$. This concludes the proof of the theorem. □

## 7.2. Proof of Lemma 4

The full proof of Lemma 4 is given in our technical report [1]. The main technical tool used in this proof and in fact the main technical tool in the proof of Theorem 2 is the result stated below (Lemma 5). To formulate this result we need to first introduce some notation.

We can assume w.l.o.g. that the sender under observation has index 0. We denote by $(0 \in L)$ and $(0 \in R)$ the events that the entry of the sender under observation belongs to the left and the right audit group, respectively, with the notion of an audit group introduced above. Note that the events $(0 \in L)$ and $(0 \in R)$ are the same independently of whether we consider the system $(A \parallel P)$ or $(A^* \parallel P)$ (these systems diverge only after these events are determined).

Let $I_L$ be a set containing 0 and, possibly, some indices of honest senders, and $Q$ be a multiset of plaintexts of size $|I_L|$. We will consider events of the form $X = I_L \cap Q$, where $I_L$ and $Q$ (by abuse of notation) represent to the following events:

– $I_L$ denotes the set of all runs of $(A \parallel P)$ (and analogously for $(A^* \parallel P)$) where the set of indices of honest senders that are in the left audit group, including the sender under observation, is $I_L$. Note that $I_L \subseteq (0 \in L)$.

– $Q$ represents the set of all runs of $(A \parallel P)$ (and analogously for $(A^* \parallel P)$) where the multiset of plaintexts chosen by the senders in $I_L$ is $Q$.

Note again that the events $X$, $I_L$, and $Q$, are the same independently of whether we consider the system $A \parallel P$ or the system $A^* \parallel P$ (the systems $A$ and $A^*$ diverge only when all those events have already been determined).

Let us observe that the event $Q$ determines possible vectors $z_1, \dots z_r$ of plaintext input messages of senders in $I_L$ (which includes the sender under observation), that yield $Q$. Note that the length of each $z_i$ is $|I_L| = |Q|$ (where $|Q|$ is the number of elements in the multi-set $Q$). We will denote the collection of these vectors by $Z_Q$. More precisely, $Z_Q$ contains only those vectors which have a probably bigger than 0 according to the probability distribution $\mu$ that we consider. By abuse of notation, each $z \in Z_Q$ may be interpreted as the event containing all the runs where the senders in $I_L$ chose their plaintexts according to the vector $z$. Again, the event $z$ is defined independently of whether we consider the system $A \parallel P$ or the system $A^* \parallel P$.

Now, roughly speaking, the following key lemma says that the adversary cannot distinguish between two vectors of choices (which are permutations of each other) of senders that belong to the same audit group.

**Lemma 5.** *For each $z_0, z_1 \in Z_Q$, we have that*

$$\mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z_0] =_{neg} \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z_1]. \quad (2)$$

The proof of this lemma is given in Appendix A. This proof is constructed as a sequence of games where we use the security properties of the used NIZK proofs, the CPA property of the used encryption scheme, and the semantic security of the used encryption scheme under re-encryption.

## 8. Conclusion

In this paper, we carried out the first formal cryptographic analysis of re-encryption RPC mix nets, which are one of or even the most deployed mix nets in real elections so far. We proved that re-encryption RPC mix nets enjoy a good level of accountability and verifiability: manipulation of just a few (honest) entries is detected with quite high probability ($1 - (\frac{3}{4})^k$ for $k$ manipulations), even if all mix servers are dishonest. Importantly, if manipulation is detected, specific misbehaving servers can (rightly) be blamed. Moreover, we introduced the notion of essentially semi-honest behavior and showed that if an adversary does not follow the protocol in an essentially semi-honest way, then he (knows that he) will be caught with a probability of at least $1/4$. In particular, this implied that risk-avoiding adversaries would behave essentially semi-honestly. With this, we showed that for risk-avoiding adversaries re-encryption RPC mix nets provide a good level of privacy, even if only one mix server is honest. Altogether, our work, for the first time, precisely states the security guarantees (accountability/verifiability and privacy) these prominent mix nets provide.

**Acknowledgment.** This work was partially supported by *Deutsche Forschungsgemeinschaft* (DFG) under Grant KU 1434/6-3 within the priority programme 1496 "Reliably Secure Software Systems – RS$^3$".

## References

[1] Ralf Küsters and Tomasz Truderung. Security Analysis of Re-Encryption RPC Mix Nets. Technical Report 2015/295, Cryptology ePrint Archive, 2015. Available at http://eprint.iacr.org/2015/295.

[2] Stephanie Bayer and Jens Groth. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.

[3] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[4] R. Carback, D. Chaum, J. Clark, adn J. Conway, E. Essex, P.S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P.L. Vora. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding governmental Elecion with Ballot Privacy. In *USENIX Security Symposium/ACCURATE Electronic Voting Technology (USENIX 2010)*. USENIX Association, 2010.

[5] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (S&P 2008)*, pages 354–368. IEEE Computer Society, 2008.

[6] Chris Culnane, Peter Y. A. Ryan, Steve Schneider, and Vanessa Teague. vVote: a Verifiable Voting System (DRAFT). *CoRR*, abs/1404.6822, 2014. Available at http://arxiv.org/abs/1404.6822.

[7] Philippe Golle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels. Optimistic Mixing for Exit-Polls. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2002.

[8] M. Jakobsson, A. Juels, and R. L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *USENIX Security Symposium*, pages 339–353, 2002.

[9] Shahram Khazaei, Tal Moran, and Douglas Wikström. A Mix-Net from Any CCA2 Secure Cryptosystem. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 607–625. Springer, 2012.

[10] Shahram Khazaei and Douglas Wikström. Randomized Partial Checking Revisited. In Ed Dawson, editor, *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013. Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2013.

[11] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 309–320. IEEE Computer Society, 2006. See http://eprint.iacr.org/2013/025/ for a full and revised version.

[12] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and Relationship to Verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS 2010)*, pages 526–535. ACM, 2010.

[13] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *32nd IEEE Symposium on Security and Privacy (S&P 2011)*, pages 538–553. IEEE Computer Society, 2011.

[14] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Formal Analysis of Chaumian Mix Nets with Randomized Partial Checking. In *35th IEEE Symposium on Security and Privacy (S&P 2014)*, pages 343–358. IEEE Computer Society, 2014.

[15] Ralf Küsters and Max Tuengerthal. The IITM Model: a Simple and Expressive Model for Universal Composability. Technical Report 2013/025, Cryptology ePrint Archive, 2013. Available at http://eprint.iacr.org/2013/025.

[16] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *8th ACM Conference on Computer and Communications Security (CCS 2001)*, pages 116–125. ACM, 2001.

[17] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference (CRYPTO 1991)*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[18] Birgit Pfitzmann. Breaking Efficient Anonymous Channel. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques*, volume 950 of *Lecture Notes in Computer Science*, pages 332–340. Springer, 1994.

[19] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. The Prêt à Voter Verifiable Election System. Technical report, University of Luxembourg, University of Surrey, 2010. http://www.pretavoter.com/publications/PretaVoter2010.pdf.

[20] K. Sako and J. Kilian. Receipt-Free Mix-Type Voting Scheme — A practical solution to the implementation of a voting booth. In *Advances in Cryptology — EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer-Verlag, 1995.

[21] Björn Terelius and Douglas Wikström. Proofs of Restricted Shuffles. In Daniel J. Bernstein and Tanja Lange, editors, *Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa*, volume 6055 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2010.

[22] Douglas Wikström. A Universally Composable Mix-Net. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 317–335. Springer, 2004.

[23] Douglas Wikström. User Manual for the Verificatum Mix-Net Version 1.4.0. Verificatum AB, Stockholm, Sweden, 2013.

## Appendix A. Proof of Lemma 5

We show that for all $z_0, z_1 \in Z_Q$:

$$\Pr[(A \parallel P \mapsto 1), X \mid z_0] =_{neg} \Pr[(A \parallel P \mapsto 1), X \mid z_1]. \quad (3)$$

Let us first notice that $z_i$ subsumes $Q$. Therefore (3) is equivalent to

$$\Pr[(A \parallel P \mapsto 1), I_L \mid z_0] =_{neg} \Pr[(A \parallel P \mapsto 1), I_L \mid z_1]. \quad (4)$$

Let $T$ be the simulator of $A \parallel P$, given by Lemma 3, which extracts, with an overwhelming probability, permutations $\pi_0, \ldots, \pi_{2m-1}$ that satisfy Condition (c) of semi-honest runs. Note that the permutations of the honest mix server do not have to be extracted by the simulator; the simulator simply knowns them, as they are generated using the (simulation of the) honest mix server program. For convenience, however, we will also call these permutations extracted. By $\pi^*$ we will denote the composition $\pi_{2m-1} \circ \cdots \circ \pi_0$ of all the permutations extracted in a simulated run (where the operator $\circ$ denotes composition of functions $((f \circ g)(x) = g(f(x)))$.

One can easily see that in the system $T$, with an overwhelming probability (more precisely, for all those runs where the extraction works correctly), the $i$-th entry in the decrypted output of the mix net is the same as the $\pi^*(i)$-th input entry (after the proof check and duplicate elimination). We will use this fact later.

Now, because the simulation is faithful, for $p \in \{0,1\}$ we have

$$\mathsf{Pr}[(A \parallel P \mapsto 1), I_L \mid z_p] = \mathsf{Pr}[(T \mapsto 1), I_L \mid z_p] \qquad (5)$$

where the set of indices $I_L$ is interpreted as an event for the system $T$ in the same way as for the system $(A \parallel P)$.

Let $T'_p$, for $i \in \{0,1\}$, be defined as the system $T$, but with the following differences. First, $T'_p$ uses $z_p$ as the (unencrypted) input of the senders in $I_L$. Second, it outputs 1 if $(A \parallel P)$ outputs 1 and the event $I_L$ is true in the run (which can be easily checked by $T'_p$). It is easy to see that

$$\mathsf{Pr}[(T \mapsto 1), I_L \mid z_p] =_{neg} \mathsf{Pr}[T'_p \mapsto 1]. \qquad (6)$$

**Simulating NIZKPs and Extracting** Let $Q_p$, for $p \in \{0,1\}$, be the program works exactly like $T'$, which includes simulation of the system $A \parallel P$, and diverges from the faithful simulation (as done in $T'$) only in the following points. Note that we must simulate $A$ in a black-box manner, while the honest component ($P$) is known and does not need to be simulated as a black-box.

Q1. Instead of using the (honest) setup algorithm to generate common reference strings $\sigma_k$ for NIZKPs of knowledge of the secret key shares corresponding to the published public key shares of the dishonest mix servers, $Q_p$ uses (the first component of) an extractor algorithm (that exists by the computational knowledge extraction property) to generate $\sigma_k$ (which is given to the adversary) along with a trapdoor $\tau_k$.

Q2. Instead of using the (honest) setup algorithm to generate common reference strings $\sigma_e$ for NIZKPs of knowledge of the plaintexts to be used by the dishonest senders (subsumed by the adversary), $Q_p$ uses (the first component of) an extractor algorithm (that exists by the computational knowledge extraction property) to generate $\sigma_e$ (which is given to the adversary) along with a trapdoor $\tau_e$.

Q3. Instead of using the (honest) setup algorithm to generate common reference strings $\sigma_{I_L}$ for NIZKPs of knowledge of the plaintexts to be used by the honest senders in $I_L$, $Q_p$ uses a simulator algorithm (that exists by the computational zero-knowledge property) to generate these CRSs $\sigma_{I_L}$ along with a trapdoor $\tau_{I_L}$.
These CRSs and the trapdoors are then used to generate (simulated) NIZKP of knowledge of the plaintexts by the honest senders in $I_L$.

Q4. Instead of using the (honest) setup algorithm to generate common reference strings $\sigma_d$ for NIZKPs of correct decryption share of the honest mix server $\mathsf{M}_j$, $Q_p$ uses a simulator algorithm (that exists by the computational zero-knowledge property) to generate $\sigma_d$ along with the trapdoor $\tau_d$.
These CRSs and the trapdoors are then used by $Q_p$ to generate (simulated) proofs of correct decryption of the honest mix server (so that the private key is not used in this step).

By the construction of $Q_p$ and by the properties of the interactive zero-knowledge proofs used in the system (computational zero-knowledge and computational knowledge extraction) we obtain:

$$\mathsf{Pr}[T'_p \mapsto 1] =_{neg} \mathsf{Pr}[Q_p \mapsto 1], \qquad (7)$$

Note that, as is necessary for use of the zero knowledge property, the system $Q_p$ only produces simulated proofs for true statements (honest sender produce ciphertexts of plaintexts they know and the honest mix server $\mathsf{M}_j$ produces a valid decryption share).

Moreover, the permutation $\pi^*$ computed by $Q_p$ is still "correct" in that, with an overwhelming probability, the $i$-th entry in the decrypted output is the same as the $\pi^*(i)$-th input entry (after the proof check and duplicate elimination), as otherwise, because this is true for $T$ and $T'$, and can be easily tested by the simulator, one could easily construct a distinguisher breaking zero-knowledge or extraction properties.

**CPA Game Simulator.** Given $z_0$, $z_1$ as above, let $S_p$, for $p \in \{0,1\}$, be the system that uses a CPA challenger $C^{enc}$ as an oracle, defined as follows:

S1. $S_p$ generates all the common reference strings to be used in the system in the same way as this is done in the system $Q_p$ (hence, some of the these CRSs are generated by simulators / extractors).

S2. $S_p$ first calls the encryption oracle $C^{enc}$ ($|z_0| = |z_1|$ times) to obtain the encrypted input $\vec{y}_{I_L}$ of senders in $I_L$, that is encrypted $z_b$, where $b$ is the secret bit used by the oracle (the CPA challenger). Then, as it was done in Q3, $S_p$ uses the simulator algorithm and the trapdoor $\tau_{I_L}$ to produce (simulated) NIZKP of knowledge of the plaintexts for the obtained vector $\vec{y}_{I_L}$ (without knowing which plaintexts have been encrypted and without knowing the used randomness).

S3. It then simulates honest senders not in $I_L$ to generate their unencrypted input $\vec{x}_h$ and then their encrypted input $\vec{y}_h$ along with the required ZK proofs (note that "real" zero knowledge proofs are produced here, using honestly generated CRSs).

S4. $S_p$ gives the encrypted entries produced so far to the adversary $A$ and simulates $A$ up to the point where it produces its (dishonest) input $\vec{y}_d$.

S5. With the ciphertexts $\vec{y}_{I_L}$, $\vec{y}_h$, and $\vec{y}_d$, $S_p$ now first performs the input validation phase of the mix net. As a result, some entries of the proofs provided by the adversary might be dropped, because the adversary might have provided invalid proofs. (Honest senders provide valid proofs only.) So, we will have a subset of entries from $\vec{y}_d$. We denote the new set of entries of the adversary by $\vec{y}'_d$. Also, some ciphertexts provided by the adversary might coincide with those provided by the honest senders, i.e., with those in $\vec{y}_{I_L}$, $\vec{y}_h$. (Since the encryption scheme used is IND-CPA secure, the probably that their are duplicate ciphertexts among those provided by the honest senders is negligible.) So, some more of the ciphertexts in $\vec{y}'_d$ might be dropped.[8]

Hence, the ciphertexts in $\vec{y}_{I_L}$ and $\vec{y}_h$ will all make it to the actual mixing phase. Only some of the entries in $\vec{y}_d$ might be dropped, and hence, only a subset $\vec{y}''_d$ may actually make it to the mixing phase. For simplicity of notation, we will, instead of referring to these ciphertexts by $\vec{y}''_d$, still refer to them by $\vec{y}_d$.

After having simulated the input validation phase, $S_p$ uses the knowledge extractor from Q2 with the trapdoor $\tau_e$ to extract the vector of plaintexts $\vec{x}_d$ from $\vec{y}_d$. (Note that by now all the entries have valid NIZKPs of knowledge of plaintexts.)

At this point the simulator $S_p$ has—up the to choices of the senders in $I_L$—complete knowledge of the input of each of the senders (honest and dishonest), except for the exact order of plaintexts for the honest senders in $I_L$. The simulator knows that it is $\vec{z}_o$ or $\vec{z}_1$. Let $\vec{x}_p$ denote the vector of plaintexts consisting of the vectors $\vec{z}_p$, $\vec{x}_h$ and $\vec{x}_d$. Hence, $\vec{x}_0$ and $\vec{x}_1$ differ only at positions corresponding to the honest senders in $I_L$. The simulator also knows the corresponding ciphertexts, which we denote by the vector $\vec{y}$, consisting of the elements of $\vec{y}_{I_L}$, $\vec{y}_h$ and $\vec{y}_d$.

S6. $S_p$ then simulates the mixing phase on the input $\vec{y}$. Doing this, $S_p$ extracts the permutations used by the mix nets in the same way, as this is done in $T$ and in $Q_p$. As previously, we will denote by $\pi^*$ the composition of these permutations.

S7. Finally, $S_p$ simulates the decryption process in such a way that it outputs $\pi^*\{\vec{x}_p\}$, by which we denote the vector $\vec{v}$ such that $\vec{v}[i] = \vec{x}_p[\pi^*(i)]$. This is the output vector one would obtain by shuffling $\vec{x}_p$ according to the extracted permutations used by the mix servers. (Note, however, that this is not necessarily the "correct" output vector, as the bit $b$ used by the CPA challenger $C^{enc}$ might not coincide with $p$.)

To this end, the simulator, using the trapdoor and the (second component of the) extractor algorithm from Q1, extracts the private keys of the dishonest mix servers. Then the simulator manipulates the decryption share of the honest mix server in the following way. Using the private keys of all the dishonest mix server and the property of decryption share extractability, the simulator, for each output target entry $\tilde{m}$ it wants to output, produces the appropriate $\tilde{h}_j$ that together with the decryption shares of the remaining mix servers yields $\tilde{m}$ (more precisely, it yields $\tilde{m}$ with overwhelming probability, that is for those runs where the adversary is semi-honest and produces correct decryption shares). The simulator also outputs simulated ZK proofs of correctness of $\tilde{h}_j$, using the trapdoor from Q4.

S8. Finally, after the output is produced, $S_p$ computes its decision as $T'$ does.

One can see that, by construction, the systems $Q_p$ and $S_P^{C^{enc}(p)}$, where $C^{enc}(p)$ is the encryption oracle (the CPA challenger for the used encryption scheme) with the challenge bit fixed to $p$, coincide, except for the decryption step. Because this step does does not affect the computation of $\pi^*$, we know that the permutation $\pi^*$ as computed by $S_p^{C^{enc}(p)}$ is the same as $\pi^*$ computed by $Q_p$.

By the above, with overwhelming probability (for all those runs where $\pi^*$ is correct, as defined for the system $Q_p$), the $i$-th entry output by $Q_p$ (obtained by decrypting the $i$-th encrypted output) is the same as the $\pi^*(i)$-th input entry $\vec{x}_p[\pi^*(i)]$ which, by construction, is the $i$-th entry output by $S_p^{C^{enc}(p)}$. Hence, the decrypted output of $S_p^{C^{enc}(p)}$ and $Q_p$ is the same. Therefore, the output of the decryption in $S_p^{C^{enc}(p)}$ is correct.

Now, because we consider risk-avoiding adversaries, that is adversaries that behave semi-honestly with overwhelming probability (Lemma 2), we know that, again with overwhelming probability, the decryption shares produced by the dishonest mix servers are correct. Furthermore, because in this case $\tilde{h}_j$ yields, along with the remaining decryption shares, the correct plaintext, by decryption share extractability, the faked decryption share $\tilde{h}_j$ produced in $S_p^{C^{enc}(p)}$ is the same as the honest decryption share $h_j$ produced in $Q_p$. Altogether, we can conclude that these two systems coincide also in the decryption step and, therefore, coincide completely. Hence we have

$$\Pr[Q_p \mapsto 1] = \Pr[S_p^{C^{enc}(p)} \mapsto 1], \tag{8}$$

By the IND-CPA property of the used encryption scheme, we immediately obtain

$$\Pr[S_1^{C^{enc}(0)} \mapsto 1] =_{neg} \Pr[S_1^{C^{enc}(1)} \mapsto 1]. \tag{9}$$

Therefore, to complete the proof, it suffices to show that

$$\Pr[S_0^{C^{enc}(0)} \mapsto 1] =_{neg} \Pr[S_1^{C^{enc}(0)} \mapsto 1]. \tag{10}$$

**Re-encryption Game Simulator.** To prove (10), we will use the semantic security of the used encryption scheme under re-encryption. Let $R$ be the system that uses a re-encryption oracle $C^{re}$ and works as follows

R1. $R$ generates all the common reference strings to be used in the system, as it is done in $Q$ (and hence in $S_p$).

8. Since in the rest of the mix net the NIZKPs in the entries are no longer used (only the actual ciphertexts are used), it does not matter whether a ciphertext in $\vec{y}'_d$ or its duplicate in $\vec{y}_{I_L}$ or $\vec{y}_h$ (if any) is dropped. In any case, one ciphertext of each set of duplicates will "survive".

R2. $R$ takes $\vec{z}_0$ as the plaintext input of senders in $I_L$, encrypts these plaintext to obtain encrypted input $\vec{y}_{I_L}$ and produces a simulated NIZKP of knowledge of plaintexts for these ciphertexts (as $S_p$ does in S2).

R3. It simulates the honest senders not in $I_L$ as $S_p$ does in S3.

R4. It produces the input of the adversary as $S_p$ does in S4.

R5. $R$ simulates the input validation steps as $S_p$ does in Step S5. $R$ also extracts the plaintexts from the ciphertexts provided by the adversary as $S_p$ does in S5.

Note that the unencrypted input, after validation, produced by $R$ is the same as the unencrypted input $\vec{x}_0$ produced by $S_0^{C^{enc}(0)}$ and $S_1^{C^{enc}(0)}$.

R6. $R$ simulates the mixing phase (including permutation extraction) in the same way as $S_p$ in Step S6, with the exception of the second mixing step of the honest mix server $\mathsf{M}_j$ which is simulated in the following way:

Let $\vec{y}'$ be the input to the second mixing step of $\mathsf{M}_j$. By this point, $R$ has extracted some permutations $\pi_0, \ldots, \pi_{2j-1}$ (from dishonest mix server before $\mathsf{M}_j$). Also, $R$ has chosen a permutation $\pi_{2j}$ for the first mixing step of $\mathsf{M}_j$ itself. Let $\pi_1^*$ be the composition of these permutations.

Let $\rho$ be the permutation (on the set of input indices) that maps $\vec{x}_1$ into $\vec{x}_0$, that is $\vec{x}_1[i] = \vec{x}_0[\rho(i)]$. Such a permutation exists, because of the way $\vec{x}_0$ and $\vec{x}_1$ are constructed (they are the same as multisets). Moreover, this permutation only permutes indices corresponding to the senders in $I_L$ (where the elements of $\vec{z}_0$ are located) and keeps intact the remaining indices, that is, for $i \notin I_L$ we have $\rho(i) = \rho^{-1}(i) = i$.

To simulate the second mixing step of the honest mix server, $R$ picks a random permutation $\pi_{2j+1}$ (as $\mathsf{M}_j$ would do). Additionally, $R$ computes the permutation $\tilde{\rho} = \pi_1^* \circ \rho^{-1} \circ (\pi_1^*)^{-1}$, and $\tilde{\pi}_{2j+1} = \pi_{2j+1} \circ \tilde{\rho}$. The simulator $R$ then uses the re-encryption oracle to obtain $\vec{y}'' = C^{re}(\pi_{2j+1}\{\vec{y}'\}, \tilde{\pi}_{2j+1}\{\vec{y}'\})$.

Notice that, for all indices $i$ such that $\pi_1^*(i) \notin I_L$, these two permutations work in exactly the same way, that is $\pi_{2j+1}^{-1}(i) = \tilde{\pi}_{2j+1}^{-1}(i)$. Let us denote the set of such indices $i$ by $I'_L$ (this set, intuitively, contains indices at the point of the input to the second mixing step of $\mathsf{M}_j$ that do not map (via $\pi_1^*$) to indices in $I_L$).

Now, $R$ computes a vector $\vec{y}'''$ from $\vec{y}''$ by substituting every element of $\vec{y}''$ that does not map to $I_L$ (that is, every element at position $k$ such that $\tilde{\pi}_{2j+1}[k] = \pi_{2j+1}[k] \in I'_L$) by a (freshly obtained) re-encryption of $\vec{y}'[\pi_{2j+1}(k)]$. This vector $\vec{y}'''$ is output as the resulting ciphertexts of the second mixing step of $M_j$. In addition, $R$ commits to $\pi_{2j+1}$ (note that this commitment may be wrong if the used permutation was $\tilde{\pi}_{2j+1}$).

We can now see that, if the event $I_L$ holds true, which implies that no index required in the audit phase for $\mathsf{M}_j$ to be opened to the right is mapped via $\pi_1^*$ to $I_L$, then $R$ can easily output the required proofs of correct re-encryption, as it was $R$ who generated the re-encryptions. Otherwise, $R$ does not output the required ZK proofs (this is, however, not important for the property we prove).

Let us observe that, altogether, $R$, for the second mixing step of $\mathsf{M}_j$, outputs a re-encryption of $\pi_{2j+1}\{\vec{y}'\}$ if the challenge bit of the re-encryption oracle is 0, or a re-encryption of $\tilde{\pi}_{2j+1}\{\vec{y}'\}$ if this bit is 1. Jumping ahead, the whole system, again depending on the bit $b$, uses the permutation $\pi^* = \pi_2^* \circ \pi_{2j+1} \circ \pi_1^*$ or $\tilde{\pi}^* = \pi_2^* \circ \tilde{\pi}_{2j+1} \circ \pi_1^*$, with $\pi_2^*$ being the composition of all the permutations applied after the second mixing step of $\mathsf{M}_j$. Let us also observe that $\tilde{\rho}$ is constructed in such a way that $\tilde{\pi}^*(i) = \rho^{-1}(\pi^*(i))$ and therefore $\pi^*\{\vec{x}_0\} = \tilde{\pi}^*\{\vec{x}_1\}$ (that is $\vec{x}_0[\pi^*(i)] = \vec{x}_1[\tilde{\pi}^*(i)]$).

R7. $R$ simulates the decryption step, similarly to S7, to produce $\pi^*\{\vec{x}_0\}$.

R8. $R$ ouputs the decision of $T'$.

Note that in the system $T$ the extraction of permutations succeeds (that is produces some permutation) with an overwhelming probability. This property carries over through all the systems, to the system $R$, as it is easily checkable by each simulator. Therefore all the operations in the above definition are well defined with overwhelming probability.

One can see that, by construction of $R$ and $S$,

$$\mathsf{Pr}[S_0^{C^{enc}(0)} \mapsto 1] = \mathsf{Pr}[R^{C^{re}(0)} \mapsto 1]. \qquad (11)$$

(It is in fact easy to construct a bijection between the runs in the two events, and hence, the probabilities are equal.)

Let $\tilde{S}_1$ be the system that works as $S_1$, but when it simulates the second mixing step of the honest mix server $\mathsf{M}_j$, it uses the permutation $\tilde{\pi}_{2j+1}$, as defined in Step R6 and it also commits to this permutation. Because $\tilde{\pi}_{2j+1}$ has the same distribution as a random permutation, we immediately have that

$$\mathsf{Pr}[S_1^{C^{enc}(0)} \mapsto 1] = \mathsf{Pr}[\tilde{S}_1^{C^{enc}(0)} \mapsto 1]. \qquad (12)$$

Let $\tilde{R}$ be the system that works as $R$ but instead of committing to the permutation $\pi_{2j+1}$, it commits to $\tilde{\pi}_{2j+1}$. Using our assumption that the commitment scheme is perfectly hiding (recall that, for runs in $I_L$, $R/\tilde{R}$ is not required to open commitments which are wrong), it easily follows that

$$\mathsf{Pr}[\tilde{R}^{C^{enc}(1)} \mapsto 1] = \mathsf{Pr}[R^{C^{re}(1)} \mapsto 1]. \qquad (13)$$

Now, using the observation we have already made, namely that $\tilde{\pi}^*\{\vec{x}_1\}$ (the output of the system $\tilde{S}_1^{C^{enc}(0)}$) is the same as $\pi^*\{\vec{x}_0\}$ (the output of $R^{C^{re}(1)}$, and hence, $\tilde{R}^{C^{re}(1)}$), we have

$$\mathsf{Pr}[\tilde{S}_1^{C^{enc}(0)} \mapsto 1] = \mathsf{Pr}[\tilde{R}^{C^{re}(1)} \mapsto 1]. \qquad (14)$$

Therefore, we obtain

$$\mathsf{Pr}[S_1^{C^{enc}(0)} \mapsto 1] = \mathsf{Pr}[(R^{C^{re}(1)} \mapsto 1]. \qquad (15)$$

Finally, by the hiding property of re-encryption, we have

$$\mathsf{Pr}[R^{C^{re}(0)} \mapsto 1] =_{neg} \mathsf{Pr}[R^{C^{re}(1)} \mapsto 1], \qquad (16)$$

which, together with the above, proves (10) and concludes the proof of Lemma 5.