Security Analysis of Re-Encryption RPC Mix Nets

Ralf Küsters and Tomasz Truderung

University of Trier, Germany
{kuesters,truderung}@uni-trier.de

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 valuable 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 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., [22,18,7,24,10,11,23,1]), some have been shown to provide strong security properties. However, most of these schemes have not been deployed in real elections so far, with Verificatum [25] 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 have been implemented in several prominent e-voting systems, including Civitas [4] and Prêt à Voter [21], 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 [5]. Some systems, such as Scantegrity [3], which has also been employed in real political elections, have used 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 [10], 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 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 [16], after Khazaei and Wikström discovered attacks on the verifiability and privacy of Chaumian RPC mix nets and proposed fixes [12].

Despite their use in practice, so far no formal security analysis of *re-encryption* RPC mix nets has been carried out. In [12], Khazaei and Wikström pointed out attacks on re-encryption RPC mix nets as well, one of which generalizes an attack by Pfitzmann [20], 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,

[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.

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 (we call them risk-avoiding, 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., [15]) and which has also been employed for the analysis of Chaumian RPC mix nets in [16]. 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 [16], which in turn is based on a general, domain independent definition of accountability proposed in [14].

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.

As for privacy, in this paper we make the following key observation, which is related to our result of accountability. If an adversary does not follow the protocol in an essentially semi-honest way, e.g., he does not perform re-encryption of the ciphertexts, 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 semi-honestly. (Conversely, semi-honest adversaries are clearly also risk-avoiding.) Now, for such adversaries, we show that 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.

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 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. Full details and proofs are provided in the appendix.

2 Re-encryption RPC Mix Net

In this section, we first recall the definition of a re-encryption RPC mix net [10] with improvements suggested in [12] and then provide a formal model of this protocol.

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 [19] (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 only be decrypted 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),[2]
- 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;[3] this is straightforward for example in the case of ElGamal and, in fact, was used in the privacy proof of the Helios voting

[2] Note that here the zero-knowledge property and knowledge extraction are not necessary.

[3] 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.

system [2]. Precise security definitions for the used cryptographic primitives are provided in Appendix A.

Set of participants. The set of participants of the protocol consists of a public, append-only *bulletin board* B, *n senders* $\mathsf{S}_1, \ldots \mathsf{S}_n$, *m mix servers* $\mathsf{M}_0, \ldots, \mathsf{M}_{m-1}$, 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 will indeed assume, both for verifiability/accountability and for privacy, that at least one auditor is honest. We note that sometimes heuristics are implemented by which this assumption can be dropped (see [10]). However, as pointed out in [12], this may lead to some problems.

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 (with audit before or after decryption) we consider. Note, however, that in general it might matter; it is the case, for instance, for Chaumian mix nets [16].

Setup phase. In this phase, every mix server M_i runs the key generation algorithm of $\mathscr{S}$ to generate its private/public key pair (sk_i, pk_i). The public key pk_i is then posted on the bulletin board along with NIZKP of knowledge of the corresponding private key. The public keys $pk_1, \ldots, pk_m$ of all the mix servers are then combined to obtain the encryption key pk to be used by the senders.

Submit phase. In this phase, every (honest) sender 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 checked (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.

All the ciphertext 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.

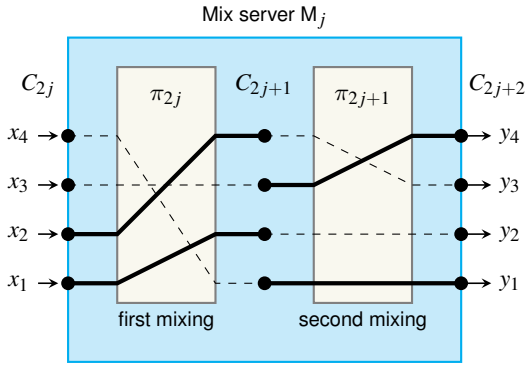

Fig. 1. Mixing by M_j. Solid bold lines represent audited links and dashed lines represent not audited links.

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 M_0 which processes it, as described below, and posts its output (which, again, is a sequence of ciphertexts) on the bulletin board B. This output becomes the input to the next mix server 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 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 M_j are as follows (see also Figure 1):

1. *Validation.* 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 and the whole protocol is aborted (except for the judging procedure; see Section 2.2).
2. *First mixing.* M_j uniformly at random chooses a permutation π_{2j} of $\{1,\ldots,l\}$ and posts the sequence C_{2j+1} of length l on 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.* M_j, again, uniformly at random chooses a permutation π_{2j+1} of $\{1,\ldots,l\}$ and posts the sequence C_{2j+2} of length l on 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 M_j on B.
4. *Posting commitments.* M_j posts two sequences of commitments on 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 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 every mix server is performed directly after this server produces 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 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 M_j is supposed to open (by posting appropriate information on B) the left link for i, i.e., M_j is supposed to open its i-th commitment from its first sequence of commitments, which should be a commitment on the value $\pi_{2j}(i)$. 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 mentioned before, 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., M_j is supposed to open its i-th commitment from its second sequence of commitments, which should be a commitment on the value $\pi^{-1}_{2j+1}(i)$. 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 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 [12], the second step, which often has been omitted in implementations and is not mentioned in [10], 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 (see Appendix A for the details of the distributed encryption scheme).

2.2 Modeling Re-encryption RPC Mix Nets

We now provide a formal model of re-encryption RPC mix nets, based on a computational model with interactive Turing machines. The computational model follows the one used

in [14,15], which we briefly recall before presenting our model of re-encryption RPC mix nets and which in turn is based on the IITM model [13,17].

Because, as we have mentioned, it does not matter for the results presented in this paper if the audit is done before or after decryption, we will focus here on the first variant (where auditing is carried out directly before the decryption phase).

The Computational Model A *process* is a set of probabilistic polynomial-time interactive Turing machines (ITMs, also called *programs*), which are connected via named tapes (also called channels). Two programs with channels of the same name but opposite directions (input/output) are connected by such channels. A process may have external input/output channels, those that are not connected internally. In a run of a process, at any time only one program is active. The active program may send a message to another program via a channel. This program then become active and after some computation can send a message to another program, and so on. A process contains a master program, which is the first program to be activated and which is activated if the active program did not produce output (and hence, did not activate another program). If the master program is active but does not produce output, a run stops.

We write a process π as $\pi = p_1 \parallel \cdots \parallel p_l$, where $p_1, \ldots, p_l$ are programs. If π_1 and π_2 are processes, then $\pi_1 \parallel \pi_2$ is a process, provided that the processes are connectible: two processes are *connectible* if common external channels, i.e., channels with the same name, have opposite directions (input/output).

A process π where all programs are given the security parameter ℓ is denoted by $\pi^{(\ell)}$. The processes we consider are such that the length of a run is always polynomially bounded in ℓ. Clearly, a run is uniquely determined by the random coins used by the programs in π.

Based on these notions of programs and processes, protocols and instances of protocols are defined as follows.

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 Π_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 a_i is *honest* in the instance π, if $\pi_{\mathsf{a}_i} = \hat{\pi}_{\mathsf{a}_i}$. A *run of* P (with security parameter ℓ) is a run of some instance of P (with security parameter ℓ). An agent a_i is honest in a run r, if r is a run of an instance of P with honest a_i.

A *property* γ *of* P is a subset of the set of all runs of P. By $\neg\gamma$ we denote the complement of γ.

As usual, a function f from the natural numbers to the interval $[0,1]$ is *negligible* if, for every $c > 0$, there exists ℓ_0 such that $f(\ell) \leq \frac{1}{\ell^c}$, for all $\ell > \ell_0$. The function f is *overwhelming* if the function $1 - f$ is negligible. A function f is λ-*bounded* if, for every $c > 0$ there exists ℓ_0 such that $f(\ell) \leq \lambda + \frac{1}{\ell^c}$, for all $\ell > \ell_0$.

Re-encryption RPC Mix Nets Modeled as Protocols We model a re-encryption RPC mix net as a protocol in the sense of Section 2.2. The set of agents of such a protocol is

as introduced in Section 2.1 plus two additional agents, the *judge* J and the scheduler Sch.

The programs of all agents are defined to have channels between each pair of agents. While not all channels are necessarily used by honest agents, they may be used by dishonest agents.

Scheduler. The honest program $\hat{\pi}_{\mathsf{Sch}}$ of the scheduler will be the master program. It triggers all agents in the appropriate order, according to the phases. It is part of every instance of the protocol and 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. Also, the scheduler triggers the judge after each protocol phase, to allow this agent to blame parties in the case of misbehavior. We also let $\hat{\pi}_{\mathsf{Sch}}$ create common reference strings (CRSs) for all the required NIZK proofs, by calling the setup algorithms of the non-interactive zero-knowledge proof systems used in the protocol, and provide them to all parties.

The bulletin board. The honest program of the bulletin board B accepts messages from all agents. A message received from an agent is stored in a list along with the identifier of the agent who posted the message. On request, B sends this list to an agent.

Auditors. For simplicity of presentation, we will simply assume one honest auditor A. The honest program $\hat{\pi}_{\mathsf{A}}$ of A, whenever triggered by the scheduler posts its random output on the bulletin board, as described in Section 2.1.

Sender. The honest program $\hat{\pi}_{\mathsf{S}}$ of a sender S implements the procedure described in Section 2.1: when triggered by the scheduler it first randomly picks a plaintext p according to some fixed probability distribution μ and then encrypts p as described and posts the resulting ciphertext along with an appropriate NIZKP on the bulletin board.[4] The honest program that is executed once p has been chosen is denoted by $\hat{\pi}_{\mathsf{S}}(p)$. As we will see, μ does not play any role for accountability, in which case we could simply assume the input to be provided by the adversary; this distribution, however, matters for our privacy result. It models prior knowledge of the adversary about the distribution of messages that honest senders send. In reality, in the context of e-voting, the adversary might not know this distribution precisely (only estimates according to election forecasts, for example). But assuming that the adversary knows this distribution precisely only makes the security guarantees that we prove stronger.

Mix server. The honest program $\hat{\pi}_{\mathsf{M}_j}$ of a mix server M_j implements the procedure describe in Section 2.1. It is triggered by the scheduler for the different phases (setup, mixing, auditing, decryption).

Judge. The honest program of the judge $\hat{\pi}_{\mathsf{J}}$ whenever triggered by the scheduler, reads data from the bulletin board and verifies it as described in Section 2.1. If a mix server M_i provides wrong output or if it simply declines to output the required data, the judge posts

[4] We will always assume that all plaintexts chosen by (honest) senders have the same length. This assumption is needed in order to prove privacy; it is not needed for accountability.

a message $\mathsf{dis}(\mathsf{M}_i)$, asserting that M_i misbehaved, i.e., M_i has not followed the prescribed protocol. More precisely, M_j is blamed in the following cases:

(a) the output produced by M_j has a wrong format (i.e. wrong number of entries, missing commitments or proofs, etc.),
(b) M_j has not output proper ciphertexts,
(c) M_j does not produce the required openings (which includes checking that the openings that come from one sequence of commitments have different values) or the proofs provided in the audit phase are invalid (i.e., do not verify),
(d) the proofs provided by M_j in the decryption or setup phase are invalid.

Trust assumptions. We assume that the scheduler, the bulletin board, the auditor, and the judge are honest. Formally, this means that the set Π_a of each such agent a consists of only the honest program $\hat{\pi}_a$ of that agent. All the other agents can (possibly) be dishonest. For a dishonest agent a, the set of its programs Π_a contains all probabilistic polynomially-bounded programs.

We denote re-encryption RPC mix nets modeled as above with m mix servers and n senders that use a probability distribution μ to determine their choices by $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$. To study privacy, by $\mathsf{P}^{j}_{\mathsf{mix}}(n,m,\mu)$ we denote the variant of the protocol, where the j-th mix server is assumed to be honest (which, again, formally means that the set of all programs of M_j contains its honest program only).

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 [16], which in turn is based on a general domain independent definition of accountability proposed in [14]. As demonstrated in [14], 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 [14] is stated with respect to a property γ of the protocol (recall the definition of a property from Section 2.2), called the *goal*, a parameter $\lambda \in [0,1]$, and an agent J of the protocol who is supposed to blame protocol participants in the case of misbehavior (resulting in the violation of the goal γ). 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 (see below for the formal definition):

(i) (*fairness*) J (almost) never blames protocol participants who are honest, i.e., run their honest program.
(ii) (*completeness*) If, in a run, the desired goal γ 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. The probability that the desired goal is not achieved but J nevertheless does not blame misbehaving parties should be bounded by λ.

This general definition of accountability is instantiated in [16] for Chaumian RPC mix nets, by fixing the specific goal γ and the parties who should be blamed if γ 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 an 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 describe this goal as a set of runs γ_0. Moreover, we generalize this goal by considering a family of goals γ_k, for $k \geq 0$, where γ_k is achieved if the output corresponds to the input up to k changed entries. In other words, for the goal γ_k we tolerate up to k changes. 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 γ_k, we consider a run r of an instance π of $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$ (with n senders). Let $s_1,\ldots,s_l$ (for $l \leq n$) be those senders that are honest in r, $x = x_1,\ldots,x_l$ be the plaintext input of these senders in r, and $y = y_1,\ldots,y_p$ (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 γ_k (in other words, γ_k is achieved in r), if there exists a subsequence x' of the honest inputs x of size $l-k$ such that x', treated as a multiset, is contained in y (again, treated as a multiset), i.e., for each element a of x', the number of a's in x' is less than or equal to the number of a's in 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 as a mix server refused to produce output), then r does not belong to γ_k, i.e., r does not achieve γ_k.[5]

Parties to be blamed. We require that if the goal γ_k is not achieved, then the judge should blame at least one mix server, i.e., post $\mathsf{dis}(\mathsf{M}_i)$ for at least one i. By the fairness property for accountability, it follows that at least this mix server definitely misbehaved. By this, every mix server risks to be blamed in the case it misbehaves, i.e., does not follow the prescribed protocol. Note that we do not require the judge to blame *all* misbehaving servers. This requirement would be too strong, because not all kinds of misbehavior (i.e., deviations from the prescribed protocol) can be detected by the judge. However, the above guarantees that at least one mix server is (rightly) blamed in the case that γ_k is not achieved. The above requirement also implies that a sender cannot break the goal γ_k: if γ_k is not achieved, this must be due to a misbehaving mix server. This is important for the robustness of the mix net: 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 π of a protocol P, we say that a verdict $\mathsf{dis}(a)$ *is true in* π if and only if a is not honest in π (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 γ_k is not satisfied, i.e., the run does not belong to γ_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 π.

Definition 1. ***(Accountability for RPC mix nets)*** *Let $P = \mathsf{P}_{mix}(n,m,\mu)$ 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 λ-accountability with tolerance k (and w.r.t. J)*, if the following two conditions are satisfied.*

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

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

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 γ_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 λ. We will see that for re-encryption RPC mix nets (the optimal/minimal) λ 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 λ, and hence, precisely measure the level of accountability re-encryption RPC mix nets provide.

Verifiability. Accountability and verifiability are tightly related as shown in [14]. Accountability is a stronger property than verifiability and subsumes it. While for verifiability one only requires protocol participants 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).

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. Our results show that level of accountability these mix nets have 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 it replaces an input ciphertext by an arbitrary other ciphertext, without actually performing re-encryption, and hence, 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 γ_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 [12]). For example, in the attack illustrated in Figure 2 a dishonest mix server 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 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 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).

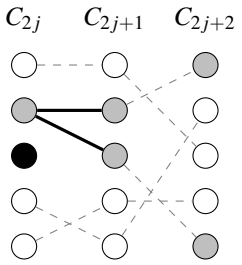

Fig. 2. Example attack on accountability.

This attack can be detected only with probability $\frac{1}{4}$, because detection requires that both p and q are audited to the left (both p and q belong to I_j). Furthermore, one mix server can apply this attack for multiple pairs of positions and it can also be performed by many mix servers in order to manipulate/replace many honest input ciphertexts. Performing the attack on $k+1$ different pairs of ciphertexts (by the same mix server or different mix servers) results in the violation of γ_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 γ_k) going unnoticed may be as high as $(\frac{3}{4})^{k+1}$, we prove that this probability is not higher, and hence, there are no worse attacks.

Security assumptions. Recall from Section 2.2 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. (Of course, privacy will require stronger properties, in fact the ones stated in Section 2.1.)

Now, the following theorem holds true for re-encryption RPC mix nets as modeled in Section 2.2, independently of whether auditing is done before and after decryption.

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

The proof of Theorem 1 follows a similar line of reasoning as the one for Chaumian RPC mix nets [16]. However, due to the different structures and cryptographic primitives used, the proofs of course differ in the details. Below we provide a sketch of the proof of Theorem 1; a complete proof can be found in Appendix B.

Proof (Proof sketch). Proving fairness (the first condition of the definition of accountability which in this context means that a mix server that runs the honest program is never blamed), is straightforward. Also, as we have already explained, the strategy of the adversary that drops exactly $k+1$ honest entries as described above breaks the goal γ_k. The probability that using this strategy the adversary successfully removes $k+1$ honest entries without this fact being noticed is λ_k. This shows that the constant in the theorem is optimal.

The main part of the proof is to show that no other strategy of the adversary is better than the aforementioned strategy. To this end, we first define the set (event) G of, so called, good runs of the system. Intuitively, G contains those runs where cryptography is not broken: no non-interactive proof of a false statement is accepted by the judge (soundness property of proofs) and no commitment is opened in more than one way (computational binding property of commitment schemes); the latter property, at first, does not seem to be a property of just one run, but by rewinding the adversary we can look at various behaviors of the adversary when mix servers are challenged in different ways during the audit. In fact, even for the non-interactive proofs different challenges are taken into account. Based on the assumptions for the cryptographic primitives, it is not hard to see that the set G has overwhelming probability. Hence, in what follows, it suffices to restrict our attention to runs in G only.

Before we proceed, we introduce the following notation. Recall that a run r is determined by the random coins the dishonest parties (the adversary) and the honest parties use. Let ω denote the random coins used in r and let ω_j denote the random coins

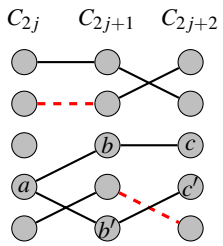

Fig. 3. An example configuration induces by ω_j. Dashed red lines represent unsafe links, while solid black lines represent safe links. Note that a safe (black) link indicates the correct re-encryption relation. Thus, for instance, a, b, b', c, and c' must be all encryptions of the same plaintext. Assuming that all the entries in the left column are honest, the right column might contain only two distinct honest entries. In this case, the mix server is nevertheless not blamed if no red link is audited and, moreover, if the two links pointing to a are not audited at the same time.

used in r, except for those used to audit the j-th mix server (these coins determine which links M_j has to open when M_j is audited). Alternatively, ω_j can be seen as the event containing all those ω' that coincide with ω, except for the coins used to audit M_j. We show that if $\omega_j \cap G \neq \emptyset$, then $\omega_j \subseteq G$. (This follows quite easily by the definition of G.)

We show that the proof of the theorem boils down to proving the following statement, for all $\omega_j \subseteq G$: Under the condition that runs are in ω_j, the probability p_{j,k_j} that the j-th mix server drops k_j honest entries (i.e., the number of honest entries in its output is by k_j smaller than the number of such entries in its input) and this mix server is nevertheless not blamed by the judge is not bigger than $\left(\frac{3}{4}\right)^{k_j}$.

The key observations for proving the above statement are the following. First, because the considered runs (that is runs in ω_j) are good, the commitments produced by M_j determine functions π_{2j} and π_{2j+1} (the commitments are consistently opened to the values of these functions, which do not have to be permutations, though, as the adversary might not have committed to permutations). This determines some fixed links between the ciphertexts of C_{2j} (the input to M_j) and C_{2j+1} as well as some fixed links between the ciphertexts of C_{2j+1} and C_{2j+2} (the output of M_j). Second, if such a link is not correct (the two ciphertext are not re-encryptions of each other), and this link is audited, then M_j is blamed by the judge (as for runs in G, M_j does not produce a proof of a false statement that nevertheless is accepted by the judge). We call such links *unsafe*.

By the above observations, every ω_j (more specifically the output of M_j in ω_j before the audit) determines a unique configuration, like, for example, the one presented in Figure 5. Now, the evaluation of the probability p_{j,k_j} is purely a matter of combinatorial, but still non-trivial reasoning, as the probability space is just the space of possible audit challenges for M_j. This mainly requires to calculate the probability of the adversary being (not) blamed in the audit, which depends on whether or not unsafe links or collisions (i.e., links pointing to the same entry) are discovered (see also the explanation in Figure 5, which illustrates the reasoning for the concrete example). □

5 Privacy and Risk-avoiding Adversaries

As observed in [12], and partly already in [20], 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 study the class of "risk-avoiding" adversaries. In this section, we formally define and discuss this class. We also prove that risk-avoiding adversaries are tightly connected with semi-honest adversaries, which we also formally introduce in the context of re-encryption RPC mix nets. In Section 7, we then prove that re-encryption RPC mix nets provide a reasonable level of privacy for risk-avoiding adversaries.

One key observation is that if an adversary does not follow the protocol in an (essentially) semi-honest way, then he will always be caught with a probability of at least $1/4$. We now first formalize what we mean by semi-honest behavior in our context and show that it is always risky in a run to deviate from this behavior. We later show that risk-avoiding adversaries are forced to behave in a semi-honest way.

Semi-honest behavior. In a nutshell, semi-honest behavior is a behavior which does not deviate from the protocol in important aspects.

Let π 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 π.

Now, we say that the j-th mix server behaves *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) M_j outputs its public key share along with a valid NIZK proof of knowledge of the private key shere, where "valid proof" (here and below) means that the proof is accepted by the judge in r.

(b) If M_j obtains a valid input C_{2j} (consisting of valid ciphertexts), then 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, M_j outputs two sequences of commitments to l values each. If M_j is audited 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, M_j produces output that the judge approves.

(c) If M_j obtains a valid input sequence C_{2j} of length l (consisting of valid ciphertexts), then M_j outputs C_{2j+1} and C_{2j+2} such that there exist permutations π_{2j} and π_{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, 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 *semi-honestly* in a run, if every dishonest mix server (which is controlled by the adversary) behaves semi-honestly in this run; honest mix servers obviously behave semi-honestly.

Intuitively, semi-honest behavior means (i) that a mix servers provides output in such a way that it is approved by the judge (Conditions (a), (b), and (d)). By the assumptions on the cryptographic primitives (see Section 2.1), this will imply that in an overwhelming set of runs the statements provided by the mix server are in fact true and knowledge can in fact be extracted from the mix server. Most importantly, (ii) that means the mix server in fact shuffles and re-encrypts the input ciphertexts (Condition (c)).

Now, the following lemma shows that under any circumstances not being semi-honest is always risky. Let B_j denote the event that M_j is blamed by the judge and let ω and ω_j be like in the proof of Theorem 1.

Lemma 1. *For all ω (and hence, for all runs), such that M_j violates Conditions (a), (b), or (d) in the run determined by ω, we have that*

$$\mathsf{Pr}\left[B_j \mid \omega\right] = 1. \tag{1}$$

Furthermore, for all ω, except for negligibly many, such that M_j violates Condition (c) in the run determined by ω, we have that

$$\mathsf{Pr}\left[B_j \mid \omega_j\right] \geq \frac{1}{4}. \tag{2}$$

Proof. The first statement of the lemma, by the judgment procedure, is trivial, so we will focus on the second one. We prove it using very similar reasoning as the one used in the proof of using Theorem 1 (see Section 4.2).

We proceed by showing that (2) holds true for all $\omega \in G$ such that, in the run induced by ω, M_j violates condition (c) of semi-honest runs, where G is the set of good runs defined in the proof of Theorem 1. Recall that this set has an overwhelming probability. For each such ω, we can follow the reasoning of the proof of Theorem 1 to conclude that ω_j induces a configuration such as the one presented in Figure 5.

Now, because we have assumed that M_j violates condition (2), one can quite easily conclude that there must be some unsafe links or some collisions in this configuration (see for the definitions of unsafe links and collisions in the proof Theorem 1). But this is caught with a probability of at least $1/4$: an unsafe link is detected with a probability of $1/2$ (this is the probability that such a link is audited, i.e., asked to be opened) and a collision is detected with a probability of at least $1/4$ (this is the probability that at list two of the links pointing to the same entry are asked to be opened). □

The interpretation of Lemma 1 is as follows: in (almost) any run if the adversary M_j decided to not shuffle and re-encrypt the ciphertexts in the expected way in this run, then (he knows that) in the audit phase he will be caught with a probability of at least $1/4$. In other words, in no such run there is a way for M_j to outsmart the system by not performing the expected task (namely shuffling and re-encrypting the input) but getting caught with only small ($< 1/4$) probability.

Therefore, if for an adversary (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 behavior described in (a) to (d). 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 it 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. Risk-avoiding adversaries are adversaries that behave in such a way that the probably of their being caught is negligible. This class of adversaries was introduced in [16]. We now first recall this definition and then link this notion to semi-honest behavior. (This was not done in [16], but is crucial for our reasoning and result.)

Let π be an instance of $\mathsf{P}_{\mathsf{mix}}(n,m,\mu)$. We say that the adversary in π (who subsumes all dishonest parties in π) is *risk-avoiding* in this instance, if the probability that π produces a run where the judge states a verdict $\mathsf{dis}(a)$ for some dishonest party a is negligible as a function in the security parameter ℓ.

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. In our case, however, by Lemma 1, risk-avoiding adversaries are forced to behave semi-honestly:

Lemma 2. *Let π be an instance of $\mathsf{P}_{mix}(n,m,\mu)$. Then, the adversary in π is risk-avoiding if and only if he behaves semi-honestly with overwhelming probability.*

Proof. Lemma 1 implies that a risk-avoiding adversary must behave semi-honestly with overwhelming probability. Otherwise, if the adversary does not behave semi-honestly with non-negligible probability p, he will be caught (according to the judging procedure) with probability at least $p/4$ (up to some negligible factor). Conversely, it is easy to see that if in a run the adversary behaves semi-honestly, then in this run he will not be blamed by the judge. □

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.

Lemma 3. *Let π be an instance of $\mathsf{P}_{mix}(n,m,\mu)$ such that the adversary in π is risk-avoiding. There exists a simulator T which faithfully[6] simulates π and additionally*

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

outputs (on some distinct tape) permutations $\pi'_0, \ldots \pi'_{2m}$ such that in an overwhelming set of runs, for each mix server M_j, the permutations π'_{2j}, π'_{2j+1} satisfy Condition (c) of semi-honest runs, by which we mean that the properties stated for π_{2j} and π_{2j+1} in Condition (c) are satisfied for π'_{2j} and π'_{2j+1}, respectively.

Proof. The simulator T simulates honest parties, and in particular, honest mix servers directly itself, and hence, T just knows their permutations. We now show how T extracts permutations from the dishonest mix server M_j in such a way that, for an overwhelming set of runs, Condition (c) of semi-honest runs is satisfied for M_j with these permutations. The same method can be applied independently to all dishonest mix servers.

The simulator is defined in such a way that it faithfully simulates the system, and performs additional steps explicitly described below, to extract the permutations for M_j. That means, in particular, that the simulator keeps complete (simulated) state of the sytem it simulates and the simulated runs, when restricted to this simulated state, strictly correspond to the runs of the original system.

For a sequence of random coins ω, we define by $\bar{\omega}^j$ the sequence of random coins that coincide with ω, except for the random coins used by auditors to challenge M_j. These random coins are complementary in $\bar{\omega}^j$ in that, whenever in ω the j-th mix server is requested to open the i-th index to the left, then in $\bar{\omega}^j$ the j-th mix server is requested to open this index to the right, and vice versa. Note that $\bar{\omega}^j$ is in ω_j (as is ω).

Let us consider a simulated run for some $\omega \in \omega_j \subseteq G$, where G are defined just as in Lemma 1 (Theorem 1). The set of such ω's has an overwhelming probability, as stated in the proof of Theorem 1. Further, let us consider only those ω for which ω_j induces a good configuration—i.e. a configuration without unsafe links and collisions—and, moreover, where M_j correctly opens commitments and provides valid proofs for all audited links, both in ω and $\bar{\omega}^j$. This is still an overwhelming set of runs. Indeed, the probability that a run induces a bad configuration must be negligible for a risk-avoiding adversary (if the set of such runs had some non-negligible probability p, then the mix server would be blamed with a probability of at least $p/4$). Similarly, the probability that either ω or $\bar{\omega}^j$ does not correctly open a commitment or does not provide a valid proof must be negligible as well (if the set of such ω has probability q, then the mix server is blamed with probability at least $q/2$).

For all ω as above (and hence, for all ω except for some negligible set), the extraction procedure detailed below is well defined. This procudere allows the simulator to extract the configuration for M_j induced by the run (i.e., all links of this configuration). Since for the considered runs, the configurations are good, it follows that extracted links form permutations (no collisions). Also, the ciphertexts connected by a link are re-encryptions of each other (links are safe). Hence, Condition (c) of semi-honest runs is satisfied for M_j with for these extracted permutations, which completes the proof.

It remains to specify the extraction procedure. The additional steps carried out by the simulator are as follows. In the run determined by some ω as above, T halts the simulation when M_j is given an audit challenge and provides an answer (at this point the simulator gets to know one half the links of the configuration). Then the simulator rewinds the simulation to the point just before the audit and challenges the mix server with the complementary challenge from $\bar{\omega}^j$. In this way, the simulator obtains full knowledge of

the configuration (that is, all the links). The simulator then rewinds the simulation again to the point where it has been halted and resumes the simulation. □

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., [15]) and which has also been employed for the analysis of Chaumian RPC mix nets in [16].

As opposed to (very strong) simulation-based definitions, see, for instance, [11] and a related game-based definition in [2], 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. (See also [16] for more discussion on this topic).

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

As already mentioned in Section 2.2, for studying privacy, we consider the protocol $\mathsf{P}_{\mathsf{mix}}^{j}(n,m,\mu)$, 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 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 s and a plaintext p, the protocol $\mathsf{P}_{\mathsf{mix}}^{j}(n,m,\mu)$ 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 s under observation that takes p as its unencrypted input (as defined in Section 2.2) and π^* 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 according to the definition of $\mathsf{P}_{\mathsf{mix}}^{j}(n,m,\mu)$, if a is the scheduler, the auditor, the judge, or the j-th mix server, then Π_a contains only the honest program of that party, as they are assumed to be honest. All other parties are potentially dishonest and may run arbitrary (adversarial) probabilistic polynomial-time programs. Since these adversaries might not try to avoid accusations by the judge, this most general class of adversaries has been called *venturesome* in [16]. The attacks by Khazaei and Wikström demonstrate that for venturesome adversaries privacy of re-encryption RPC mix nets cannot be guaranteed. In order to define privacy w.r.t. risk-avoiding adversaries, we simply restrict the set of programs π^* to those that are risk-avoiding (see Section 5). In a system $(\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*)$ with a sender s under observation, π^* 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 (w.r.t. venturesome or risk-avoiding adversaries) is now defined as follows, where we use the following notation: $\mathsf{Pr}[(\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*)^{(\ell)} \mapsto 1]$ 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 ℓ 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 $\mathsf{P}^{j}_{mix}(n,m,\mu)$ as before let 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 *δ*-privacy (w.r.t. risk-avoiding adversaries)*, if*

$$\mathsf{Pr}[(\hat{\pi}_{\mathsf{s}}(p) \parallel \pi^*)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{\mathsf{s}}(p') \parallel \pi^*)^{(\ell)} \mapsto 1] \tag{3}$$

is δ-bounded as a function of the security parameter ℓ, for all valid input plaintexts[7] *p, p' and all (risk-avoiding) programs π^* of the remaining parties such that (at least) l senders are honest in π^*.*

Since δ typically depends on the number l of honest senders, privacy is formulated w.r.t. this number. Note that a smaller δ means a higher level of privacy. However, δ 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 s has sent, and hence, can easily distinguish between s having sent p or p'.

Privacy for the Ideal Mix Net Protocol. Before, we provide the analysis of the level of privacy provided by re-encrypted RPC mix nets, we first recall results for the ideal ideal mix net from [16], 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.

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 outputs the submitted messages after having applied a random permutation. Honest senders choose their inputs according to the distribution μ.

The level of privacy provided by the ideal mix net depends on the number l of honest senders and the probability distribution μ on valid input plaintexts.

To define $\delta^{id}_{l,\mu}$, we need the following terminology. Let $\{p_1,\ldots,p_k\}$ 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* $r = (r_1,\ldots,r_k)$ of the protocol, where r_i, $i \in \{1,\ldots,k\}$, 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 $r_1 + \cdots + r_k = l+1$ (l honest senders plus the sender under observation).

We denote by *Out* the set of all pure outputs. Let A^i_r denote the probability that the choices made by the honest senders yield the pure output r, given that the sender under observation submits p_i. Further, let $M_{j,j'} = \{r \in Out : A^j_r \leq A^{j'}_r\}$. Now, the intuition behind the definition of $\delta^{id}_{l,\mu}$ is as follows: If the observer, given a pure output 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 $r \in M_{j,j'}$, i.e., the pure output is more likely if the sender submitted $p_{j'}$.

[7] Recall that valid input plaintexts all have the same length.

This leads to the following level of privacy provided by the ideal mix net protocol with l honest senders and the probability distribution μ:

$$\delta^{id}_{l,\mu} = \max_{j,j' \in \{1,\ldots,k\}} \sum_{r \in M_{j,j'}} (A^{j'}_r - A^j_r),$$

with example values depicted in Figure 4. (Note that $A^{j'}_r$ - A^j_r depend on l and μ.)

7 Analysis of Privacy 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.

As already illustrated in Section 2.1, it is in the very nature of re-encryption RPC mix nets that some information about the input to a mix server is mapped to its output. Consequently, even a risk-avoiding adversary obtains some partial information about how the input of the honest mix server is mapped to its output. Hence, even in this case privacy cannot be as in the ideal case. Note that for the other mix servers (controlled by the adversary), the adversary has full knowledge about the mapping from the input to the output.

7.1 Privacy for Risk-Avoiding Adversaries

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

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

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

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

Example values for $\delta_{l,\mu}$ are depicted in Figure 4. 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. Recall that our result holds true under the pessimistic assumption that among all mix servers, there is one honest mix server only.

Proof. 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

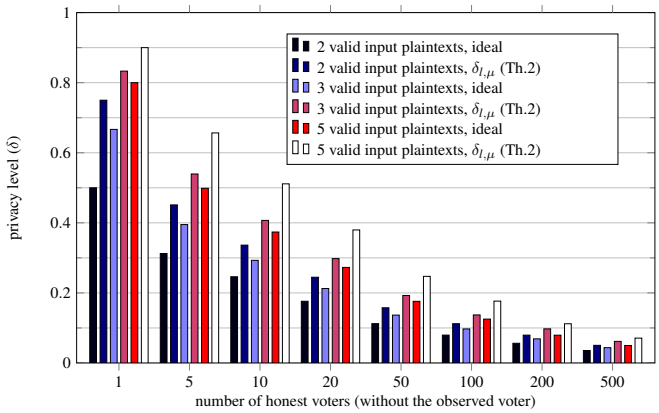

Fig. 4. Level of privacy ($\delta_{l,\mu}$) for $\mathsf{P}^{j}_{\mathsf{mix}}(n,m,\mu)$ w.r.t. risk-avoiding adversaries and in the ideal case $\delta^{id}_{l,\mu}$, uniform distribution of input plaintexts. These figures have been obtained by straightforward calculations using the δ-formulas as provided in the theorems. For non-uniform distributions, $\delta_{l,\mu}$ is close to ideal as well.

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 ℓ, 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 M_j opens the left link and those for which 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 M_j) with $i = (\pi_{2j} \circ \pi_{2j-1} \circ \cdots \circ \pi_0)(i')$, where π_k are permutations extracted from the run,[8] (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 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.

[8] 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.

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 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 denotes 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 server 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 μ) 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 ℓ. 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 $\mathsf{Adv}^{priv}_{A^*,P,p,p'} \leq_{neg} \delta_{l,\mu}$.

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

We will use the convention that the choice of the sender under observation depends on a bit b: if $b = 0$, this sender uses p, if $b = 1$, he uses p'. This bit is chosen with uniform probability. Using this convention, to conclude the proof, we have to show that

$$\mathsf{Pr}[A \parallel P \mapsto b] \leq_{neg} \mathsf{Pr}[A^* \parallel P \mapsto b], \tag{4}$$

where $(A \parallel P \mapsto b)$ denotes the event that A, interacting with P, correctly guesses the bit b, and similarly for $(A^* \parallel P \mapsto b)$.

It suffices to consider the (overwhelming) event *SHP* defined above. For such runs, the events introduced below are well defined. Such runs are, by definition, semi-honest and therefore, as already mentioned, not aborted (the mixing and decryption are completed) and all the entries make it to the output. As the systems $(A \parallel P)$ and $(A^*||P)$ diverge only in the way decisions are made, *SHP* applies to both systems, and a run in *SHP* for $(A \parallel P)$ corresponds to a run in *SHP* for $(A^*||P)$. Note that a run of $(A \parallel P)$ is semi-honest if and only if the corresponding run of $(A^*||P)$ is semi-honest.

W.l.o.g. let the sender under observation be the 0-th sender. 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). Moreover, we can observer that every run in *SHP* of $(A \parallel P)$ and $(A^* \parallel P)$ belongs to either $(0 \in L)$ or $(0 \in R)$. Therefore, to complete the proof it is enough to show both of the following inequalities:

$$\mathsf{Pr}[(A \parallel P \mapsto b), (0 \in L)] \leq_{neg} \mathsf{Pr}[(A^* \parallel P \mapsto b), (0 \in L)]$$

and

$$\mathsf{Pr}[(A \parallel P \mapsto b), (0 \in R)] \leq_{neg} \mathsf{Pr}[(A^* \parallel P \mapsto b), (0 \in R)].$$

In what follows, we prove the first inequality. The proof for the second one is very similar.

Let I_L be a set containing 0 and, possibly, some indices of honest senders, and Q by a multiset of plaintexts of size $|I_L|$. We will consider events of the form $X = I_L, Q$, 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).

Note that for different choices of the index set I_L and the multiset Q of plaintexts, one obtains a different event X. In this sense, X denotes an element of a family of events of the described form. Note that the number of elements in this family is fixed, finite and independent of the security parameter. (Clearly, the events represented by these elements depend on the security parameter.) Therefore, to complete the proof, it is enough to show that, for all X of non-negligible probability,

$$\mathsf{Pr}[(A \parallel P \mapsto b),\ X] \leq_{neg} \mathsf{Pr}[(A^* \parallel P \mapsto b),\ X] \tag{5}$$

The rest of this section is devoted to proving (5) for a fixed event X of non-negligible probability.

Let us observe that the event Q determines possible vectors $z_1, \ldots z_r$ 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 μ 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 $A^* \parallel P$.

The main technical result used in this proof is the following lemma (the proof of this lemma is given in Appendix C).

Lemma 5. *For each $z \in Z_Q$, we have*

$$\mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z] =_{neg} \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid Q]. \tag{6}$$

Because the above lemma works for any risk-avoiding adversary A, it holds, in particular, for A^*. Therefore we obtain:

Corollary 1. *For each $z \in Z_Q$, we have*

$$\mathsf{Pr}[(A^* \parallel P \mapsto 1), X \mid z] \equiv \mathsf{Pr}[(A^* \parallel P \mapsto 1), X \mid Q].$$

Note that the decision of A^*, by definition, is based solely and deterministically on Q and therefore this decision is the same for all runs in X. Let us assume that A^* outputs 1 for all runs in X; the proof for the case where A^* outputs 0 is analogous. In this case, by the definition of A^*, we know that $\mathsf{Pr}[b = 0 \mid Q] \leq \mathsf{Pr}[b = 1 \mid Q]$ and hence

$$\mathsf{Pr}[b = 0, Q] \leq \mathsf{Pr}[b = 1, Q] \tag{7}$$

Note that the events $(b = 0)$, $(b = 1)$, and Q, used in the above probabilities, are defined independently of whether we consider the system $A \parallel P$ or $A^* \parallel P$: besides the bit b, they are determined solely by the random coins used by the senders in the set I_L (which is a fixed set of indices) to determine their choices (according to the probability distribution μ or, in the case of the sender under observation, according to b).

In the following, we denote by Z_0 the set of those elements z in Z_Q for which the choice of the sender under observation is p (recall that z is a vector determining the choices of senders in I_L, including the choice of the sender under observation, compatible with Q; recall also that the choice of the sender under observation may be either p or p' according to b). Similarly, we denote by Z_1 the set of those elements z in Z_Q for which the choice of the sender under observation is p'. Note that by the definitions of Z_0 and Z_1 we have that

$$\sum_{z \in Z_0} \mathsf{Pr}[z] = \mathsf{Pr}[b = 0, Q] \quad \text{and} \quad \sum_{z \in Z_1} \mathsf{Pr}[z] = \mathsf{Pr}[b = 1, Q]. \tag{8}$$

Note that, again, all the events used in the above probabilities are defined independently of whether we consider the system $A \parallel P$ or $A^* \parallel P$. Now, using (7), (8), and Lemma 5, we obtain

$$\begin{aligned}
\mathsf{Pr}[(A \parallel P \mapsto b), X] &= \sum_{z \in Z_Q} \mathsf{Pr}[z] \cdot \mathsf{Pr}[(A \parallel P \mapsto b), X \mid z] \\
&= \sum_{z \in Z_0} \mathsf{Pr}[z] \cdot \mathsf{Pr}[(A \parallel P \mapsto 0), X \mid z] \\
&\quad + \sum_{z \in Z_1} \mathsf{Pr}[z] \cdot \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z] \\
&=_{neg} \sum_{z \in Z_0} \mathsf{Pr}[z] \cdot \mathsf{Pr}[(A \parallel P \mapsto 0), X \mid Q] \\
&\quad + \sum_{z \in Z_1} \mathsf{Pr}[z] \cdot \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid Q] \\
&=_{neg} \mathsf{Pr}[b = 0, Q] \cdot \mathsf{Pr}[(A \parallel P \mapsto 0), X \mid Q] \\
&\quad + \mathsf{Pr}[b = 1, Q] \cdot \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid Q] \\
&\leq \mathsf{Pr}[b = 1, Q] \cdot \mathsf{Pr}[(A \parallel P \mapsto 0), X \mid Q] \\
&\quad + \mathsf{Pr}[b = 1, Q] \cdot \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid Q] \\
&\leq_{neg} \mathsf{Pr}[b = 1, Q] \cdot \mathsf{Pr}[X \mid Q].
\end{aligned}$$

Note that by the definition of Z_Q, Z_0, and Z_1, the probability of z is bigger than 0, for all security parameters, and therefore the conditional probabilities above are always well defined. In a similar way, using Corollary 1 and the assumption that A^* outputs 1 for all runs in X (and hence in Q), we obtain:

$$\begin{aligned}
\mathsf{Pr}[(A^* \parallel P \mapsto b), X] &=_{neg} \mathsf{Pr}[b = 0, Q] \cdot \mathsf{Pr}[(A^* \parallel P \mapsto 0), X \mid Q] \\
&\quad + \mathsf{Pr}[b = 1, Q] \cdot \mathsf{Pr}[(A^* \parallel P \mapsto 1), X \mid Q] \\
&= \mathsf{Pr}[b = 1, Q] \cdot \mathsf{Pr}[X \mid Q],
\end{aligned}$$

Combining the above results, we obtain (5). □

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 showed that if an adversary does not follow the protocol in a semi-honest way, then he (knows that he) will be caught with a probability of at least $1/4$. This observation motivated us to consider the class of semi-honest or equivalently, as we prove, risk-avoiding adversaries, which are not willing to take this risk of being caught. In fact, severe penalties and the loss of reputation might be effective deterrents in many practical situations. For risk-avoiding adversaries, we showed that 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 these prominent mix nets provide.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. Philippe Golle, Markus Jakobsson, Ari Juels, and Paul F. Syverson. Universal re-encryption for mixnets. In *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, pages 163–178. Springer, 2004.
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. Jens Groth. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2010.
9. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect Non-interactive Zero Knowledge for NP. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, 2006.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.

21. 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.
22. 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.
23. 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.
24. 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.
25. Douglas Wikström. User Manual for the Verificatum Mix-Net Version 1.4.0. Verificatum AB, Stockholm, Sweden, 2013.

A Security Definitions for Cryptographic Primitives

A.1 Distributed Public-key Encryption

A distributed encryption scheme is a tuple $(\mathsf{KeyShareGen}, \mathsf{KeyCom}, \mathsf{Enc}, \mathsf{DecShareSDec})$ where all the algorithms implicitly take the security parameter as an argument such that:

- $\mathsf{KeyShareGen}$ is a ppt algorithm that generates a public-private key share (sk_i, pk_i) (where pk_i, besides the actual public key share, might contain additional information, such as a proof of knowledge of sk_i),
- $\mathsf{KeyCom}(pk_1, \ldots, pk_n)$ is a deterministic polynomial-time algorithm which returns the public key pk obtained from the public key shares; this algorithm may fail (produce $\perp$) if the public key shares are invalid,
- $\mathsf{Enc}(pk, m)$ is ppt algorithm that encrypts the message m under the public key pk,
- $\mathsf{DecShare}(c, pk, sk_i)$ is a deterministic polynomial-time algorithm which for a ciphertext c, a public key pk, and a secret-key share sk_i returns a decryption share h_i,
- $\mathsf{SDec}(c, h_1, \ldots, h_n)$ is a deterministic polynomial-time algorithms which returns a message or $\perp$, in the case that decryption fails.

We will assume that there are publicly known polynomial time algorithms to check if (for a given security parameters) a given message is a valid plaintext, a valid ciphertext and whether a given pair sk_i, pk_i is a valid key share pair.

For the *correctness* of such a scheme we require that (1) KeyCom does not fail when given valid key shares, and (2) if $pk = \mathsf{KeyCom}(pk_1, \ldots, pk_n)$ and $pk \neq \perp$, then for all plaintexts m and ciphertext c with $c \leftarrow \mathsf{Enc}(pk, m)$ and $h_i = \mathsf{DecShare}(c, pk, sk_i)$ (for $i \in \{1, \ldots, n\}$) we have that $\mathsf{SDec}(c, h_1, \ldots, h_n) = m$.

Let C^{enc} be a probabilistic polynomial-time algorithm, called a *challenger* that takes a bit b and a public key pk and serves the following challenge queries:

> for a pair of messages (x_0, x_1) of the same length, return $\mathsf{Enc}(pk, x_b)$, if $pk \neq \perp$, or $\perp$ otherwise.

Now, the distributed encryption scheme is *IND-CPA secure*, if

$$\mathsf{Pr}[S_A(0) \text{ outputs } 1] - \mathsf{Pr}[S_A(1) \text{ outputs } 1]$$

is a negligible function in the security parameter ℓ, for every polynomially bounded adversary $A = (A_1, A_2)$ where A_1 and A_2 share state and A_2 has oracle access to C^{enc}, and

$$\begin{aligned} S_A(b) = {} & (pk_1, sk_1) \leftarrow \mathsf{KeyShareGen}(); \\ & (pk_2, \ldots, pk_n) \leftarrow A_1(pk_1); \\ & pk \leftarrow \mathsf{KeyCom}(pk_1, \ldots, pk_n); \\ & b' \leftarrow A_2^{C^{enc}(b,pk)}() \\ & \text{output } b' \end{aligned}$$

The distributed encryption scheme is *decryption share extractable*[9] if the following is true: There exists a ppt algorithm which given public key shares $pk_1, \ldots, pk_n$ and corresponding private key shares $sk_2, \ldots, sk_n$ such that the pk_i, sk_i are valid key share pairs and given a valid ciphertext c, any plaintext $\tilde{m}$, computes $\tilde{h}_1$ such that for $h_i = \mathsf{DecShare}(c, pk, sk_i)$, for $i = 2, \ldots, n$, we have that $\mathsf{SDec}(c, \tilde{h}_1, h_2, \ldots, h_n) = \tilde{m}$ and, moreover, if c is an encryption of $\tilde{m}$, then $\tilde{h} = \mathsf{DecShare}(c, pk, sk_1)$.

A common instantiation of a distribtued encryption scheme is ElGamal.

A.2 Re-encryption

A distributed encryption scheme with re-encryption is a distributed encryption scheme with an additional ppt algorithm reencrypt that takes a public key and a ciphertext c and returns a re-encryption of c, i.e. a ciphertext c' such that decryption of c and c' yield the same plaintext; in other words, if $c = \mathsf{Enc}^r_{pk}(m)$, then $c' = \mathsf{Enc}^{r'}_{pk}(m)$, i.e., c' contains the same plaintext but was encrypted using different random coins r'.

Let C^{re} be a probabilistic polynomial time algorithm (a re-encryption challenger), that takes a bit b and a public key pk and serves the following queries:

> for two vectors of ciphertexts (x_0, x_1), where the ciphertexts have the same length and x_1 is a permutation of x_0, the challenger returns a re-encryption of x_b (i.e. a vector y of the same length such that $y[i] = \mathsf{reencrypt}(pk, x_b[i])$, if $pk \neq \perp$, or $\perp$ otherwise.

We say that a decryption scheme with re-encryption provides *semantic security under re-encryption* [6], if

$$\mathsf{Pr}[S_A(0) \text{ outputs } 1] - \mathsf{Pr}[S_A(1) \text{ outputs } 1]$$

[9] This property is an abstract formulation of a property of the ElGammal distributed encryption scheme used in the privacy proof of the Helios voting system [2].

is a negligible function in ℓ, for every polynomially bounded adversary $A = (A_1, A_2)$ where A_1 and A_2 share state and A_2 has oracle access to C^{re}, and

$$\begin{aligned} S_A(b) = (pk_1, sk_1) &\leftarrow \mathsf{KeyShareGen}(); \\ (pk_2, \ldots, pk_n) &\leftarrow A_1(pk_1); \\ pk &\leftarrow \mathsf{KeyCom}(pk_1, \ldots, pk_n); \\ b' &\leftarrow A_2^{C^{re}(b,pk)}(); \\ &\text{output } b'. \end{aligned}$$

A.3 Commitments

As usual, a *commitment scheme* is a tuple (M, C, R, Comm), where, for each value ℓ of the security parameter, M_ℓ, C_ℓ and R_ℓ are sets of messages called the *message space*, the *commitment space*, and the *opening space*, respectively, and Comm is a deterministic, polynomial-time algorithm that for each ℓ, each $m \in M_\ell$ and $r \in R_\ell$, outputs $c = \mathsf{Comm}(1^\ell, m, r) \in C_\ell$. By $\mathsf{Comm}(1^\ell, m)$ we denote the probabilistic algorithm that chooses a random r from R_ℓ with uniform probability and returns $\mathsf{Comm}(1^\ell, m, r)$.

Such a commitment scheme is *perfectly hiding*, if for each ℓ and each $m, m' \in M_\ell$, we have that $\mathsf{Comm}(1^\ell, m)$ and $\mathsf{Comm}(1^\ell, m')$ have the same distribution.

A commitment scheme (M, C, R, Comm) is *computationally binding*, if for all polynomially bounded adversaries A

$$\begin{aligned} \mathsf{Pr}[(m, m', r, r') \leftarrow A(1^\ell) \ : r, r' \in R_\ell,\ m, m' \in M_\ell,\ m \neq m', \\ \mathsf{Comm}(1^\ell, m, r) = \mathsf{Comm}(1^\ell, m', r')] \end{aligned}$$

is a negligible function in ℓ.

A.4 Non-Interactive Zero-Knowledge Proofs

Following [8,9], we provide here a definition of non-interactive zero-knowledge proofs in the common reference string model.

Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$, we call x the statement and w the witness. Let $L_R = \{x : \exists w \text{ such that } (x, w) \in R\}$. A *non-interactive proof system* for a language L_R is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Setup}, \mathsf{Prover}, \mathsf{Verifier})$, where

- Setup (the common reference string generator) takes as input a security parameter 1^ℓ and the statement length n and produces a common reference string $\sigma \leftarrow \mathsf{Setup}(n)$,[10]
- Prover (the prover) takes as input the security parameter 1^ℓ, a common reference string σ, a statement x, and a witness w and produces a proof $\pi \leftarrow \mathsf{Prover}(\sigma, x, w)$,
- $\mathsf{Verifier}$ (the verifier) takes as input the security parameter 1^ℓ, a common reference string σ, a statement x, and a proof π and outputs $1/0 \leftarrow \mathsf{Verifier}(\sigma, x, \pi)$ depending on whether it accepts π as a proof of x or not,

[10] We omit the security parameter in the notation, also for the prover and the verifier, for simplicity of notation.

such that the following conditions (completeness and soundness) are satisfied.

Perfect completeness: For $n = \ell^{O(1)}$ and all adversaries A outputting $(x,w) \in R$ with $|x| = n$

$$\mathsf{Pr}[\sigma \leftarrow \mathsf{Setup}(n);\ (x,w) \leftarrow A(\sigma);\ \pi \leftarrow \mathsf{Prover}(\sigma,x,w);\ b \leftarrow \mathsf{Verifier}(\sigma,x,\pi) :\ b = 1] = 1.$$

This condition says that an honest prover should always be able to convince an honest verifier of a true statement.

Computational soundness: For $n = \ell^{O(1)}$ and all non-uniform polynomial time adversaries A, the probability

$$\mathsf{Pr}[\sigma \leftarrow \mathsf{Setup}(n);\ (x,\pi) \leftarrow A(\sigma);\ b \leftarrow \mathsf{Verifier}(\sigma,x,\pi) :\ \ x \notin L_R \text{ and } b = 1]$$

is a negligible function of the security parameter.

This condition captures that it should be infeasible for an adversary to come up with a proof of a false statement that is nevertheless accepted by the verifier.

We say that a non-interactive proof system $(\mathsf{Setup}, \mathsf{Prover}, \mathsf{Verifier})$ is *zero-knowledge* (NIZKP) if the following condition is satisfied:

Computational (single-theorem) zero-knowledge: There exists a polynomial-time simulator $S = (S_1, S_2)$ such that, for $n = \ell^{O(1)}$ and all stateful, interactive, non-uniform polynomial time adversaries $A = (A_1, A_2)$ outputting $(x,w) \in R$ with $|x| = n$, we have

$$\begin{aligned}&\mathsf{Pr}[\sigma \leftarrow \mathsf{Setup}(n);\ (x,w) \leftarrow A_1(\sigma);\ \pi \leftarrow \mathsf{Prover}(\sigma,x,w);\ b \leftarrow A_2(\pi)\ :\ b = 1] \\ &\quad \approx \mathsf{Pr}[(\sigma,\tau) \leftarrow S_1(n);\ (x,w) \leftarrow A_1(\sigma);\ \pi \leftarrow S_2(\sigma,\tau,x);\ b \leftarrow A_2(\pi)\ :\ b = 1]\end{aligned}$$

(where $\approx$ means that the difference between the two probabilities is a negligible function in the security parameter). In the latter system, S_1 outputs a simulated common reference string and a simulation trapdoor. S_2 takes the common reference string, the simulation trapdoor, and a statement (but not the witness) as input and produces a simulated proof.

We use here the single-theorem variant of the zero-knowledge property, where the common reference string is used to produce (and verify) only one ZK proof, as opposed to the (general) multi-theorem variant of the zero-knowledge property, where the same common reference string can be used to produce many proofs. This suffices for our application, because, in the mix net protocol we consider, the number of produced ZK-proofs is bounded and known a priori, which corresponds to the case, where A can only submit a bounded number of queries. In such a case, the single-theorem variant of the zero-knowledge property implies the multi-theorem variant (the length of σ can be expanded by factor of M, where M is the bound on the number of ZK-proofs).

We say that a non-interactive proof system $(\mathsf{Setup}, \mathsf{Prover}, \mathsf{Verifier})$ produces proofs of knowledge, if the following condition is satisfied:

Computational knowledge extraction: There exists a knowledge extractor $E = (E_1, E_2)$ such that for $n = \ell^{O(1)}$, the following conditions hold true: (a) for all non-uniform polynomial time adversaries A

$$\mathsf{Pr}[\sigma \leftarrow \mathsf{Setup}(n);\ b \leftarrow A(\sigma)\ :\ b = 1]\ \approx\ \mathsf{Pr}[(\sigma,\tau) \leftarrow E_1(n);\ b \leftarrow A(\sigma)\ :\ b = 1]$$

and (b), for all non-uniform polynomial time adversaries A, the probability

$$\mathsf{Pr}[(\sigma,\tau) \leftarrow E_1(n);\ (x,\pi) \leftarrow A(\sigma);\ w \leftarrow E_2(\sigma,\tau,x,\pi);\\ b \leftarrow \mathsf{Verifier}(\sigma,x,\pi)\ :\ b = 0 \text{ or } (x,w) \in R]$$

is an overwhelming function of the security parameter.

Note that (computational) knowlege extraction implies the existence of a witness and, therefore, it implies (computational) adaptive soudness.

A.5 (ZK) Proofs used in the Protocol

The (zero-knowledge) proof used in the protocol are formally defined as follows.

- *NIZKP of knowledge of the private key share*. For a given pk_i, the statement is: $\exists sk_i : (pk_i, sk_i)$ is a valid key share pair.
- *NIZKP of knowledge of plaintext*. For (c, pk), the statement is: $\exists m, r : c = \mathsf{Enc}^r_{pk}(m)$.
- *Non-interactive proof of correct re-encryption*. For (c, c', pk), the statement is: $\exists r : c' = \mathsf{ReEnc}^r_{pk}(c)$.
- *NIZKP of correct decryption share*. For input of the form $(h_i, c, pk_1, \ldots, pk_n)$, the statement is:

  $$\exists sk_i : (pk_i, sk_i) \text{ is a valid key share pair } \wedge\ \mathsf{DecShare}(c, pk, sk_i) = h_i.$$

  where $pk = \mathsf{KeyCom}(pk_1, \ldots, pk_n)$.

B Proof of Theorem 1

Proving fairness (the first condition of the definition of accountability which in this context means that a mix server that runs the honest program is never blamed), is easy, so the rest of this section is devoted to the proof of completeness.

As we have already explained, the strategy of the adversary that drops exactly $k+1$ honest entries, as described in Section 4.1, breaks the goal γ_k. The probability that using this strategy the adversary successfully removes $k+1$ honest entries without this fact being noticed is λ_k. Note that, using this strategy, one mix server can drop not more than half of its input entries. Therefore, if k is big, the adversary may need to use more than one mix server. We may simply assume that there are enough mix servers for the adversary to carry out this strategy.

In the remainder of the proof, we show that no other strategy of the adversary is better than the aforementioned strategy.

Let P denote the composition of the (honest) programs of those parties of the system that are assumed to be honest. The remaining parties are subsumed by an adversary A. Note that all mix servers are subsumed by A. Let us denote by X the event that, in the system $A \parallel P$, the goal γ_k is not achieved and no mix server is blamed. Our goal is to show that the probability $\mathsf{Pr}[X]$ is λ_k-bounded, where $\lambda_k = (\frac{3}{4})^{k+1}$.

All the probabilities in this proof are computed, if not stated explicitly otherwise, over the sample space Ω such that every atomic event $\omega \in \Omega$ is a composition of random

coins used by all the protocol participants of the system under consideration, that is the adversary and the parties in P. As usual, elements of Ω are sampled with uniform probability.

Counting honest entries. Let Plain_i denote the multiset of plaintexts obtained by decrypting all the elements of C_i. Note that some element of C_i may decrypt to $\perp$ (i.e. do not decrypt correctly). We now define the *number of honest entries in* C_i as the size of the (multiset) intersection of the multiset Plain_i and the multiset of the input plaintexts of honest senders.

It is easy to see that the number of honest entries in C_0 is l. It is also easy to see that, if j-th mix server follows the protocol specification, then the number of honest entries in its output (C_{2j+2}) is the same as the number of honest entries in its input (C_{2j}). Finally, let us notice that in the optimal strategy described above, whenever a dishonest mix server performs one manipulation, it decreases the number of honest entries by one.

Good runs. Let us consider an adversary A' which precisely simulates A, but carries out the following additional steps. When it produces output of a mix server M_j (that is messages C_{2j+1}, C_{2j+2}, comm_{2j}, and comm_{2j+1}, where comm_{2j}, comm_{2j+1} are supposed to be commitments to the permutations π_{2j} and π_{2j+1}, respectively), receives an audit challenge I_j, and produces its response to this challenge (in an honest run that would be appropriate openings to the challenged commitments along with required ZK proofs), A' additionally *simulates* the responses of A to *all possible alternative challenges* I'_j. For this, A' always rewinds its state to the one right before receiving a challenge. In such a simulation, only the challenges change, otherwise all random coins remain the same. If, during this process, A' discovers that (for different challenges) A opens the same commitment to two different values, it reports this conflict, by writing the commitment and the two different openings on a distinct tape. Note that, although the number of possible challenges may be very big, it is constant (it is not a function of the security parameter) and, therefore, this simulation can be done in a polynomial time. Note also that, for the same $\omega \in \Omega$, the runs of $A \parallel P$ and $A' \parallel P$ are identical, up to the additional simulation that A' performs and the possible reported conflicts.

In the following, we will assume that there are independent and distinct subcomponents of $\omega \in \Omega$ that are used by the judge (the party who is supposed to blame dishonest protocol participant) as the source of random coins for the verification of distinct ZK proofs produced by the mix servers. Clearly, this assumption is safe, as it does not change the distribution of random coins used in those verification steps.

Let us denote by G the subset of Ω such that for $\omega \in G$ we have:

(i) in the run of the system $(A' \parallel P)$ for ω, no conflict is reported,

(ii) in the run of the system $(A' \parallel P)$ for ω, no proof of an invalid statement is produced that is accepted by the judge (this includes also all proofs produced in the simulation that A' performs and proofs of correct re-encryption and correct decryption).

We will call the runs of $(A \parallel P)$ for $\omega \in G$, *good runs*.

It is easy to see that the probability of G is overwhelming. Indeed, one can easily see that the set of those $\omega \in \Omega$ for which $A' \parallel P$ reports a conflict is negligible, as otherwise A' would break the computational binding property of the used commitment scheme. Also, the set of runs where mix nets output ZK proofs for invalid statements that are

accepted by the judge is, by the computational soundness of the used proof system, negligible.

By the above observation, to complete the proof, it is enough to show that $\mathsf{Pr}[X \mid G] \leq (\frac{3}{4})^{k+1}$.

Overall structure. Let us first notice that if a mix server produces an malformed output, that is an output that contains some entries which are not valid ciphertexts, such a server is blamed with probability one (and the event X does not hold). We also note that, considering good runs, if the decryption is not done correctly, then, some mix server is blamed as well. Therefore, given G, the goal γ_k is not achieved only if the number of honest entries in C_{2m} is smaller than $l-k$.

We know that the number of honest entries in C_0 is equal to the number l of honest senders. Let L be the set of vectors $l = (l_0, \ldots, l_{m-1})$, where $l_j \in \{0, \ldots, l\}$, and $l_{m-1} < l-k$ (recall that m is the number of mix servers). For $j \in \{0, \ldots, m-1\}$, by l_j^* we will represent the event that the j-th mix server produces a well-formed output (denoted by C_{2j+2}) containing exactly l_j honest entries. By abuse of notation, let l_j represent the event that, *additionally*, this mix server is not blamed (the audit procedure, done before or after the decryption step, does not discover any misbehaviour of this mix server). Now, $l = l_0, \ldots, l_{m-1}$ represents the event that no mix server is blamed for producing wrong output and, for each $j \in \{0, \ldots, m-1\}$, the number of honest entries in C_{2j+2} is l_j.

As we have noted, given G, the goal γ_k is *not* achieved if and only if the number of honest entries in C_{2m} is smaller than $l-k$. It follows that, (again, given G) by the definition of L (more precisely, by the assumption that $l_{m-1} < l-k$), the event X holds if and only if l holds, for some $l \in L$. Therefore, we have the following, where $L^* = \{l \in L \mid \mathsf{Pr}[l \mid G] \neq 0\}$:

$$\begin{aligned}
\mathsf{Pr}[X \mid G] &= \sum_{l \in L^*} \mathsf{Pr}[l \mid G] \\
&= \sum_{l \in L^*} \mathsf{Pr}[l_0^* \mid G] \cdot \mathsf{Pr}[l_0 \mid G, l_0^*] \cdot \mathsf{Pr}[l_1^* \mid G, l_0] \cdot \mathsf{Pr}[l_1 \mid G, l_0, l_1^*] \cdots \\
&\quad \cdots \mathsf{Pr}[l_{m-1}^* \mid G, l_0, \ldots, l_{m-2}] \cdot \mathsf{Pr}[l_{m-1} \mid G, l_0, \ldots, l_{m-2}, l_{m-1}^*] \\
&= \sum_{l \in L^*} \mathsf{Pr}[l_0^* \mid G] \cdot \mathsf{Pr}[l_1^* \mid G, l_0] \cdots \mathsf{Pr}[l_{m-1}^* \mid G, l_0, \ldots, l_{m-2}] \cdot \\
&\quad \cdot \mathsf{Pr}[l_0 \mid G, l_0^*] \cdot \mathsf{Pr}[l_1 \mid G, l_0, l_1^*] \cdots \mathsf{Pr}[l_{m-1} \mid G, l_0, \ldots, l_{m-2}, l_{m-1}^*]
\end{aligned}$$

We will show the following fact which means that, if a dishonest mix server drops k_j honest entries, then the probability that this goes undetected is at most $(\frac{3}{4})^{k_j}$.

Lemma 6. *For all $j \in \{0, \ldots, m-1\}$ and $l \in L^*$:*

$$Pr_j = \mathsf{Pr}[l_j \mid G, l_0, \ldots, l_{j-1}, l_j^*] \leq \left(\tfrac{3}{4}\right)^{k_j}, \tag{9}$$

where $k_j = \max(0, l_{j-1} - l_j)$ and where we put $l_{-1} = l$ (recall that l is the number of honest senders and hence the number of honest entries in the input to M_0).

We prove this lemma below. Now, using this lemma, we complete the proof of Theorem 1. First, we note that (9) implies the following fact.

$$\mathsf{Pr}[l_0 \mid G, l_0^*] \cdots \mathsf{Pr}[l_{m-1} \mid G, l_0, \ldots, l_{m-2}, l_{m-1}^*] \leq \left(\tfrac{3}{4}\right)^{k'}$$

where $k_j = \max(0, l_{j-1} - l_j)$ and $k' = k_0 + \cdots + k_{m-1}$. Now, because $l_{m-1} < l - k$, we have $k < k'$ and therefore we obtain:

$$\mathsf{Pr}[X \mid G] \leq \left(\tfrac{3}{4}\right)^{k+1} \cdot \sum_{l \in L^*} \mathsf{Pr}[l_0^* \mid G] \cdot \mathsf{Pr}[l_1^* \mid G, l_0] \cdots \mathsf{Pr}[l_{m-1}^* \mid G, l_0, \ldots, l_{m-2}]$$

One can easily see that the sum above is not bigger than 1, since this sum is equal to

$$\sum_{l_0} \mathsf{Pr}[l_0^* \mid G] \sum_{l_1} \mathsf{Pr}[l_1^* \mid G, l_0] \cdots \sum_{l_{m-1}} \mathsf{Pr}[l_{m-1}^* \mid G, l_0, \ldots, l_{m-2}],$$

where if conditionals have zero probability, we define the conditional probabilities to be zero. Therefore, we obtain

$$\mathsf{Pr}[X \mid G] \leq \left(\tfrac{3}{4}\right)^{k+1} \quad ,$$

which completes the proof.

Proof (Proof of Lemma 6). To prove (9), we need to first introduce some notation. For $\omega \in \Omega$, we define the event $\omega_j \subseteq \Omega$ where all random coins coincide with those of ω except possibly for the coins used by the auditors to generate the challenge for M_j. So, ω_j leaves it open how M_j is challenged, but otherwise the output of M_j is determined by ω_j. Let Ω_j be set of all ω_j as above. Similarly, for $\omega \in \Omega$ let $\omega_j^A \subseteq \Omega$ be the event where the random coins used by the auditors coincide with those of ω but other random coins are not fixed. Note that $\omega_j \cap \omega_j^A$ fixes a unique run of the system. In what follows, let $\Omega_j^* = \{\omega_j \mid \omega_j \in \Omega_j,\ \omega_j \cap G \cap l_0 \cap \cdots \cap l_{j-1} \cap l_j^* \neq \emptyset\}$. Now, we can represent the probability Pr_j as

$$Pr_j = \sum_{\omega_j \in \Omega_j^*} \mathsf{Pr}[\omega_j] \cdot \mathsf{Pr}[l_j \mid G, l_0, \ldots, l_{j-1}, l_j^*, \omega_j]. \tag{10}$$

We will show that for each $\omega_j \in \Omega_j^*$

$$\mathsf{Pr}[l_j \mid G, l_0, \ldots, l_{j-1}, l_j^*, \omega_j] \leq \left(\tfrac{3}{4}\right)^{k_j} \tag{11}$$

This implies that (10) is upper bounded by $(\frac{3}{4})^{k_j}$ and, hence, that (9) holds true. By this, proving (11) completes the proof.

In order to prove (11), we first observe several useful facts. Let us notice that ω_j determines messages output by the j-th mix server before it is challenged (in all runs in ω_j these messages are the same). Now, because $\omega_j \in \Omega_j^*$, and hence, in particular $\omega_j \cap l_j^* \neq \emptyset$, we know that the number of honest entries in the output C_{2j+2} equals l_j. So we know the following about the runs in ω_j:

(F1) The number of honest entries in the output C_{2j+2} is equal to l_j.

Similarly, we know the following:

(F2) The number of honest entries in C_{2j} is equal to l_{j-1}.

Since $\omega_j \cap G \neq \emptyset$, we obtain that $\omega_j \subseteq G$: By the definition of ω_j and G, and the construction of A', it is easy to see that if one run in ω_j satisfies the Conditions (i) and (ii) of the definition of G, then all runs in ω_j satisfy these conditions.

In particular, by Condition (i) of the definition of G, we obtain the following fact which says that commitments are opened in an unique way:

(F3) There exists a function f such that for every run in ω_j and for every commitment c produced by some mix server in this run, if some mix server produces an opening to c, then the opened value is $f(c)$.

Similarly, by Condition (ii) of the definition of G, we obtain the following fact.

(F4) For every run in ω_j if a ZK proof produced by some mix server is accepted by the verifier, then the statement is actually true.

Next, in order to prove (11), we first rule out some trivial cases. The remainder of the proof then boils down to some combinatorial arguments.

Trivial cases. In the trivial cases we consider the probability

$$\mathsf{Pr}[l_j \mid G, l_0, \ldots, l_{j-1}, l_j^*, \omega_j] \tag{12}$$

in (11).

If the output of M_j does not conform to the expected format, then this mix server is immediately blamed. In this case, the probability (12) is zero (as the event l_j implies that, in particular, M_j is not blamed) and (11) trivially follows. So, in the following, we will assume that:

(A1) The output of M_j conforms to the expected format, i.e. M_j outputs C_{2j+1}, C_{2j+2}, comm_{2j}, and comm_{2j+1}, where C_{2j+1} and C_{2j+2} contain the same number r of elements, and comm_{2j} and comm_{2j+1} also contain r elements each, where the latter are supposed to be the commitments to the permutations π_{2j} and π_{2j+1}^{-1}, respectively (but M_j might be cheating).

If $l_{j-1} - l_j \leq 0$, then $k_j = 0$ and (11) trivially holds. So we will assume that

(A2) $l_{j-1} - l_j > 0$ (and thus $k_j > 0$), i.e. some number of honest entries is dropped by M_j.

The non-trivial case. Now, we prove (12) under the assumptions (A1) and (A2).

For simplicity of the argument, we assume that for every run in ω_j if M_j is challenged to open a commitment, it will always open the commitment correctly and such that the opened value is in the range $\{1, \ldots, r\}$. The case where M_j does not do this for every run in ω_j (and hence, every challenge) is proven in a similar way. Note that if M_j does not open a commitment as required, it will be blamed by the judge. Hence, such a run does not belong to the event l_j.

Let us consider the output of M_j as determined by ω_j. For each $i \in \{1, \ldots, r\}$, we will denote by $link_L(i)$ the index $f(c)$, where $c = \mathsf{comm}_{2j}[i]$ is the commitment of M_j to the i-th element of the permutation π_{2j} and f is given as in (F3). Note that by our assumption $link_L(i) \in \{1, \ldots, r\}$. Similarly, we will denote by $link_R(i)$ the index $f(c) \in \{1, \ldots, r\}$,

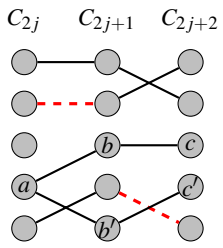

Fig. 5. An example configuration induces by ω_j. Dashed red lines represent unsafe links, while solid black lines represent safe links. Note that a safe (black) link indicates the correct re-encryption relation. Thus, for instance, a, b, b', c, and c' must be all encryptions of the same plaintext. Assuming that all the entries in the left column are honest, the right column might contain only two distinct honest entries. In this case, the mix server is nevertheless not blamed if no red link is audited and, moreover, if the two links pointing to a are not audited at the same time.

where $c = \mathsf{comm}_{2j+1}[i]$. Note that by (F3) the functions $link_L$ and $link_R$ are the same for all runs in ω_j.

Further, for each $i \in \{1,\ldots,r\}$, we will say that the index i is *left-unsafe* if $C_{2j+1}[i]$ is not a re-encryption of $C_{2j}[i']$, where $i' = link_L(i)$. Note that if a left-unsafe link is audited (as requested by ω_j^A), then by (F4) the mix server is blamed. Similarly, we will say that the index i is *right-unsafe* if $C_{2j+2}[i']$ is not a re-encryption of $C_{2j+1}[i]$, where $i' = link_R(i)$. Again, if a right-unsafe link is audited, the mix server is blamed.

With the above definition, ω_j induces a configuration as the one presented in Figure 5. To complete the proof, we need to show that, for all possible such configurations which drop exactly k_j honest entries, the probability that the mix server is not blamed is bounded by $(\frac{3}{4})^{k_j}$. Computing this probability is, essentially, a purely combinatorial argument, as presented in what follows.

If two or more indices from the middle column point to the same index in the left column, we call it a left collision group. Formally, a *left collision* is a maximal set of indices L such that there exists a with $link_L(i) = a$ for all $i \in L$. Analogously, we define a *right collision group* as a maximal set of two or more indices from the middle column pointing to the same index in the right column. As an example, the configuration in Figure 5 contains a left collision group of size 2 (nodes labeled by b and b' point to the same left entry).

Let A be a subset of all runs in $G \cap l_0 \cap \cdots \cap l_{j-1} \cap l_j^* \cap \omega_j$ such that at most one index of every left collision group is right-unsafe. Let $\bar{A}$ denote its complement. As $\bar{A} \cap G \cap l_0 \cap \cdots \cap l_{j-1} \cap l_j^* \cap \omega_j$ implies that either two indices of a left collision group are challenged to the left, or a right-unsafe link is challenged to the right, and hence, that M_j is blamed, we get that $l_j \cap G \cap l_0 \cap \cdots \cap l_{j-1} \cap l_j^* \cap \omega_j \subseteq A$ since l_j requires that M_j is not blamed. If $\mathsf{Pr}[A \mid G, l_0, \ldots, l_{j-1}, l_j^*, \omega_j] = 0$, we are done. Therefore, in the following we assume $\mathsf{Pr}[A \mid G, l_0, \ldots, l_{j-1}, l_j^*, \omega_j] \neq 0$. It suffices to show that

$$\mathsf{Pr}[l_j \mid G, l_0, \ldots, l_{j-1}, l_j^*, \omega_j, A] \leq \left(\frac{3}{4}\right)^{k_j}.$$

That is, in the following we assume that at most one index of every left collision group is right unsafe.

We say that an index in the middle column is *honest* if one of the following holds true:

1. it is neither left- nor right-unsafe nor belonging to a left collision group,

2. if it belongs to a left collision group that contains at least two indices that are not left-unsafe, then it is the lowest index of this collision group that is neither left- nor right-unsafe. (Note that, because of the event A, we know that one of the above mentioned indices is not right-unsafe.)

Let h_1 denote the number of indices that are not honest in the middle column.

Intuitively, the indices that are not honest in the middle column correspond to potentially dropped (honest) entries of M_j. (They do not have to actually drop entries because an adversary might for instance use right-unsafe indices to undo the effect of a left-unsafe link, which, however, would not be a good strategy of cheating.) Dishonest indices might not be the only reason that entries are dropped. Entries might also be further dropped due to right collision groups.

Now, the structure of the proof is roughly as follows: We show that the probably that the adversary is not blamed due to having produced dishonest indices is bounded by $\left(\frac{3}{4}\right)^{h_1}$. We then show that to every honest index i, except for some number h_2 of honest indices, we can assign a unique index k in the output of M_j such that the entry pointed to by k is a re-encryption of the entry pointed to by i. The last step of the proof is to show that the probability of not getting blamed due to right collision groups, given that M_j is not blamed due to producing dishonest indices, is bounded by $\left(\frac{3}{4}\right)^{h_2}$. The theorem then follows by the observation that the number of dropped entries is at most $h_1 + h_2$ (that is the number of dishonest indices plus the number of honest indices that did not get an assignment).

Now, given the facts and the assumptions listed above, it is easy to see that M_j is not blamed if and only if no unsafe link is challenged (otherwise, by (F4), the ZK proof would not verify) and if at most one link from each left (or right) collision group is challenged to the left (or the right), as otherwise it is visible that M_j did not commit to a permutation.)

Let B be a subset of all runs in A such that no left-unsafe index is challenged to the left, no right-unsafe index is challenged to the right, and no two indices of a left collision group are challenged to the left. In other words, in runs in B the mix server M_j is not blamed due to having produced dishonest indices. (Still, in a run in B the mix server M_j can be blamed if the configuration contains a right collision group and this group is discovered.) We now show that

$$\mathsf{Pr}[B \mid G \cap l_0 \cap \cdots \cap l_{j-1} \cap l_j^* \cap \omega_j \cap A] \leq \left(\frac{3}{4}\right)^{h_1}. \tag{13}$$

The set of dishonest indices can be partitioned as follows: A dishonest index might i) belong to a left collision group which contains an honest index (and hence, at least two indices that are not left unsafe), ii) belong to a left collision group that contains exactly one index that is not left unsafe (and hence, this left collision group does not contain an honest index), or iii) it might not belong to any left collision group but is left- and/or right-unsafe or it might belong to a left collision group which, however, contains only indices that are left unsafe. We now look at these (disjoint) sets separately.

The probably that at most one index in a collision group of size k as in i) is challenged to the left is $\frac{k+1}{2^k}$ (we consider $k+1$ cases, each occurring with probability $\frac{1}{2^k}$: one case

if no index from the group is chosen for audit to the left, and k cases if exactly one index is chosen). An elementary calculation shows that $\frac{k+1}{2^k} \leq \left(\frac{3}{4}\right)^{k-1}$.

Now, let us consider left collision groups of the form ii). The probability that no index from such a left collision group of size k (≥ 2) that is not left unsafe is challenged to the left is $\left(\frac{1}{2}\right)^{k-1} \leq \left(\frac{3}{4}\right)^{k}$.

The dishonest indices in the set iii) are left- and/or right unsafe. So, for each such index, the probability that it is not challenged to its unsafe side is at most $\frac{1}{2} < \frac{3}{4}$.

As the indices are audited independently and the cases above do not overlap, we immediately get (13).

In the following, we assign to all but h_2 honest indices in the middle column (with h_2 being defined below) a unique index in the output column. More precisely, to every honest index i in the middle column (except for h_2 honest indices), we assign a unique (and different) index a in the output column of M_j such that the entry pointed to by a is a re-encryption of the entry pointed to by i. We call these honest indices *completely honest*. So, except for h_2 honest indices, all honest indices are completely honest.

Then, we have that $k_j \leq h_1 + h_2$, because for every completely honest index i, the input entry b pointed to by i (i.e., the entry in the left column at the index $link_L(i)$) has a valid re-encryption in the output column, namely at the index a assigned to i by the provided construction. Hence, informally speaking, b makes it through M_j. Moreover, the above assignment (from the input entries to the output entries) is one-to-one, because there is at most one honest index in a left collision group and two different completely honest indices are not assigned the same index. Therefore, the number of dropped entries of M_j is at most as high as the number of indices that are not completely honest. And, as the set of indices that are not completely honest is the disjoint union of the h_1 indices that are not honest in the middle column and the h_2 indices that are honest in the middle column but not completely honest, $k_j \leq h_1 + h_2$ follows.

Let C be a subset of all runs in A such that at most one index of a right collision group in M_j is challenged to the right. (Note that this is trivially true if there is no right collision group.) In other words, in runs in C the mix server M_j is not blamed due to the cheating done by using right collision groups.

Then, $C \cap B$ is the event that M_j is neither blamed for using left collision groups nor for right collision groups nor for left-unsafe indices nor for right-unsafe indices. And, as already mentioned, given the facts and assumptions listed before, M_j is not blamed exactly in this event, i.e. $C \cap B = l_j \cap A$.

Let $\{B_i : i = 1, \ldots, p\}$ denote a partition of B such that one B_i contains all runs in B where for every index that is left or right unsafe or belongs to a left collision groups it is fixed in the same way (in the different runs in B_i) whether this index is challenged to the left or to the right. In other words, one B_i determines one pattern of how indices in left collision groups and indices with left or right unsafe links are audited. (If $p = 0$, then this means that B is empty. In this case, we are done. Otherwise (if $p > 0$), all B_i are non-empty, by the definition of a partition.)

In what follows, we define for every B_i an h_2^i following the above sketched intuition such that $k_j \leq h_1 + h_2^i$. Let $h_2^{min} = \min\{h_2^1, \ldots, h_2^p\}$. We then show that

$$\mathsf{Pr}[C \mid B_i] \leq \left(\frac{3}{4}\right)^{h_2^i}. \tag{14}$$

This then completes the proof:

$$\begin{aligned}
\mathsf{Pr}[C \cap B] &= \mathsf{Pr}[B] \cdot \mathsf{Pr}[C \mid B] \\
&= \mathsf{Pr}[B] \cdot \sum_{i=1}^{p} \mathsf{Pr}[C \cap B_i \mid B] \\
&= \mathsf{Pr}[B] \cdot \sum_{i=1}^{p} \mathsf{Pr}[C \mid B_i] \cdot \mathsf{Pr}[B_i \mid B] \\
&\leq \mathsf{Pr}[B] \cdot \sum_{i=1}^{p} \left(\frac{3}{4}\right)^{h_2^i} \cdot \mathsf{Pr}[B_i \mid B] \\
&\leq \mathsf{Pr}[B] \cdot \left(\frac{3}{4}\right)^{h_2^{min}} \\
&\leq \left(\frac{3}{4}\right)^{h_1} \cdot \left(\frac{3}{4}\right)^{h_2^{min}} \\
&\leq \left(\frac{3}{4}\right)^{k_j}.
\end{aligned}$$

So, it remains to define h_2^i and prove (14). In what follows, let $B^* = B_i$. The following assumption (P1) is, as argued below, made w.l.o.g. Also, the fact (P2) will be useful.

(P1) No right collision group contains two links that are opened to the right according to B^*.
Otherwise, if some right collision group contains two indices of which the right links are opened according to B^*, then the server is blamed. Hence the probability of not getting blamed (given B^*) is 0. In particular, $\mathsf{Pr}[C \mid B^*] = 0$, and hence, we would be done with proving (14).

(P2) For every left collision group that contains an honest index there is, according to B^*, an opened index to the right which is neither left nor right unsafe.
Indeed, as every left collision group with an honest index contains at least two indices that are not left-unsafe, given B, (at least) one of them must be open to the right. This index, again given B, cannot be right-unsafe.

For assigning unique outputs to honest indices in the middle column, we proceed as follows:

I) If an honest index i does not belong to a right collision group, we assign i to its right index $link_R(i)$.

II) Let i be an honest index that belongs to a left collision group L (so i is *the* honest index in L) and to a right collision group G. Note that in B^* it is fixed whether i is opened to the right or to the left (since i belongs to a left collision group).

a) If i is opened to the right according to B^*, we assign to i the index to which G is linked to (i.e., the index $link_R(i)$).

b) If i is opened to the left according to B^*, by (P2), there is another index i' in L which is neither left nor right unsafe and that is opened to the right according to B^*. We assign i to $link_R(i')$.

III) The remaining honest indices are those that do not belong to a left collision group but to a right collision group. While, by I) and II), all honest indices considered above have obtained an assignment, the remaining ones have not obtained an assignment yet. For these indices the assignments (if any) are defined as follows:

So, let i be an honest index that does not belong to a left collision group but to a right collision group. Let G denote the right collision group i belongs to. If $link_R(i)$ (which is the index to which all indices in G are linked to) has already been used for an assignment (i.e., some of the honest indices in I) or II) have been assigned to $link_R(i)$), then i is not assigned any index. In particular, this means that i is not completely honest, and hence, it contributes to h_2^i.

Otherwise (if $link_R(i)$) has not been assigned to), if i is the minimal index in the group G which does not belong to a left collision group, then i is assigned to $link_R(i)$.[11] If i is not minimal, it is not assigned any index, which, in particular, means that i is not completely honest, and hence, it contributes to h_2^i.

As, by (P1), in every right collision group, there is at most one index that by B^* is opened to the right, we indeed have for every completely honest index a different assignment. Also, by the construction it is clear that the entry corresponding to the index, say a, to which a completely honest index i is assigned is the result of correct re-encryption of the entry corresponding to $link_L(i)$ (in the two mixing steps in M_j). Moreover, $link_L(i)$ is different for different completely honest indices (because two honest indices do not belong to the same left collision group). Hence, in this sense the entry at $link_L(i)$ makes it to the output of M_j, namely at index a. So, indeed we have that not more than $h_1 + h_2^i$ honest entries could have been dropped, and hence, $k_j \leq h_1 + h_2^i$, where h_2^i is defined to be the number of honest indices which have not received an assignment.

In what follows, let $h_2^* = h_2^i$. It remains to show that $\mathsf{Pr}[C \mid B^*] \leq \left(\frac{3}{4}\right)^{h_2^*}$. To this end, let $G_1, \ldots, G_z$ denote the right collision groups. For $i = 1, \ldots, z$, let t_i denote the number of honest indices in G_i that did not get an assignment, i.e., those honest indices that are not completely honest. Then, we have that $\sum_{i=1}^{z} t_i = h_2^*$.

Note that in B^* the random coins of all indices that belong to a left collision group or that have a left or right unsafe link are fixed in a specific way. For all honest indices that are not completely honest, B^* does not fix how they are audited because these indices do not belong to left collision groups and neither have left nor right unsafe links. How these indices are audited is chosen independently and uniformly at random even given B^*.

[11] Minimality is not actually important. If $link_R(i)$) has not been assigned yet, we could take any index in G that does not belong to a left collision group and assign it to $link_R(i)$. However, exactly one of these indices should be assigned to $link_R(i)$.

W.l.o.g., we assume that the $G_1,\ldots,G_z$ are ordered in such a way that $G_1,\ldots G_y$ are those right collision groups that contain an index opened to the right (according to B^*) and $G_{y+1},\ldots,G_z$ are those right collision groups that do not contain such an index.

Then, G_i (for $i = y+1,\ldots,z$) contains at least t_i+1 indices for which it is not determined by B^* how they are challenged (and for which the challenge is chosen independently and uniformly at random, even given B^*), as argued next: Clearly, as already mentioned, since G_i contains t_i honest but not completely honest indices, we know that for these t_i indices it is not determined by B^* how they are audited. Moreover, clearly none of these indices is assigned to the index q that G_i is linked to. This implies, by (III), that q is assigned to some other honest index k. If k does not belong to a left collision group, then it must have obtained its assignment according to (III). But then the way k is audited is not determined by B^* and we are done, because, in this case, we altogether have t_i+1 indices in G_i for which the way they are audited is not determined by B^*. Now, consider the case that k belongs to a left collision group. This case is not possible: According to (II), k can only be assigned to q if q is connected to an opened linked. But, by assumption, G_i does not contain indices that are opened to the right.

For $i = 1,\ldots,y$, let C_i contain all runs in B^* such that no index of those indices in G_i that are not fixed by B^* is challenged to the right. Note that G_i contains at least t_i such indices, namely, the honest but not completely honest indices in G_i. Hence, we have that $\mathsf{Pr}[C_i \mid B^*] \leq \left(\frac{1}{2}\right)^{t_i} \leq \left(\frac{3}{4}\right)^{t_i}$.

For $i = y+1,\ldots,z$, let C_i contain all runs in B^* such that at most one index of those indices in G_i that are not fixed by B^* is challenged to the right. We know by the above that there are at least t_i+1 such indices in G_i. Hence, we have that $\mathsf{Pr}[C_i \mid B^*] \leq \frac{t_i+1+1}{2^{t_i+1}} \leq \left(\frac{3}{4}\right)^{t_i}$.

By definition of C and the construction of the C_i, we have that $C \cap B^* = \bigcap_{i=1}^{z} C_i$. Also, since different C_i talk about different sets of (independent) indices, we know that $\{C_i : i = 1,\ldots,z\}$ are independent, given B^*. Hence, we obtain the following:

$$\mathsf{Pr}[C \mid B^*] = \mathsf{Pr}[\bigcap_{i=1}^{z} C_i \mid B^*] = \prod_{i=1}^{z} \mathsf{Pr}[C_i \mid B^*] \leq \prod_{i=1}^{z} \left(\frac{3}{4}\right)^{t_i} \leq \left(\frac{3}{4}\right)^{h_2^*}.$$

This proves (14), and concludes the proof. □

Remark 1. As one can see from the proof (and as shortly remarked in Section 2.2), the probability distribution μ does not play any role in the above result. Indeed, we could allow the adversary to provide unencrypted input for the honest senders and the result would still work.

C Proof of Lemma 5

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

$$\mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z_0] =_{neg} \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z_1], \tag{15}$$

From this, equivalence (6) easily follows, because

$$\begin{aligned}\mathsf{Pr}[(A \parallel P \mapsto 1), X \mid Q] &= \sum_{z' \in Z_Q} \mathsf{Pr}[z' \mid Q] \cdot \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z'] \\ &=_{neg} \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z] \cdot \sum_{z' \in Z_Q} \mathsf{Pr}[z' \mid Q] \\ &= \mathsf{Pr}[(A \parallel P \mapsto 1), X \mid z],\end{aligned}$$

Hence, to complete the proof, we need to prove (15).

Let us first notice that z_i subsumes Q. Therefore (15) is equivalent to

$$\mathsf{Pr}[(A \parallel P \mapsto 1), I_L \mid z_0] =_{neg} \mathsf{Pr}[(A \parallel P \mapsto 1), I_L \mid z_1]. \tag{16}$$

Let T be the simulator of $A \parallel P$, given by Lemma 3, which extracts, with an overwhelming probability, permutations $\pi_0, \ldots, \pi_{2m}$ 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 π^* 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

$$\mathsf{Pr}[(A \parallel P \mapsto 1), I_L \mid z_p] = \mathsf{Pr}[(T \mapsto 1), I_L \mid z_p] \tag{17}$$

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

$$\mathsf{Pr}[(T \mapsto 1), I_L \mid z_p] = \mathsf{Pr}[T'_p \mapsto 1]. \tag{18}$$

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 σ_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 σ_k (which is given to the adversary) along with a trapdoor τ_k.

Q2. Instead of using the (honest) setup algorithm to generate common reference strings σ_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 σ_e (which is given to the adversary) along with a trapdoor τ_e.

Q3. Instead of using the (honest) setup algorithm to generate common reference strings σ_{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 σ_{I_L} along with a trapdoor τ_{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 σ_d for NIZKPs of correct decryption share of the honest mix server M_j, Q_p uses a simulator algorithm (that exists by the computational zero-knowledge property) to generate σ_d along with the trapdoor τ_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:

$$\mathsf{Pr}[T'_p \mapsto 1] =_{neg} \mathsf{Pr}[Q_p \mapsto 1], \tag{19}$$

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 M_j produces a valid decryption share).

Moreover, the permutation π^* 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 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 τ_{I_L} to produce (simulated) NIZKP of knowledge of the plaintexts for the obtained vector 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 x_h and then their encrypted input 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 y_d.

S5. With the ciphertexts y_{I_L}, y_h, and 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 y_d. We denote the new set of entries of the adversary by y'_d. Also, some ciphertexts provided by the adversary might coincide with those provided by the honest senders, i.e., with those in y_{I_L}, 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 y'_d might be dropped.[12]
Hence, the ciphertexts in y_{I_L} and y_h will all make it to the actual mixing phase. Only some of the entries in y_d might be dropped, and hence, only a subset y''_d may actually make it to the mixing phase. For simplicity of notation, we will, instead of referring to these ciphertexts by y''_d, still refer to them by y_d.

After having simulated the input validation phase, S_p uses the knowledge extractor from Q2 with the trapdoor τ_e to extract the vector of plaintexts x_d from 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 z_o or z_1. Let x_p denote the vector of plaintexts consisting of the vectors z_p, x_h and x_d. Hence, x_0 and 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 y, consisting of the elements of y_{I_L}, y_h and y_d.

S6. S_p then simulates the mixing phase on the input 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 π^* the composition of these permutations.

S7. Finally, S_p simulates the decryption process in such a way that it outputs $\pi^*\{x_p\}$, by which we denote the vector v such that $v[i] = x_p[\pi^*(i)]$. This is the output vector one would obtain by shuffling 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

[12] 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 y'_d or its duplicate in y_{I_L} or y_h (if any) is dropped. In any case, one ciphertext of each set of duplicates will "survive".

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 π^*, we know that the permutation π^* as computed by $S_p^{C^{enc}(p)}$ is the same as π^* computed by Q_p.

By the above, with overwhelming probability (for all those runs where π^* 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 $x_p[\pi^*(i)]$ 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

$$\mathsf{Pr}[Q_p \mapsto 1] = \mathsf{Pr}[S_p^{C^{enc}(p)} \mapsto 1], \tag{20}$$

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

$$\mathsf{Pr}[S_1^{C^{enc}(0)} \mapsto 1] =_{neg} \mathsf{Pr}[S_1^{C^{enc}(1)} \mapsto 1]. \tag{21}$$

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

$$\mathsf{Pr}[S_0^{C^{enc}(0)} \mapsto 1] =_{neg} \mathsf{Pr}[S_1^{C^{enc}(0)} \mapsto 1]. \tag{22}$$

Re-encryption Game Simulator. To prove (22), 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).

R2. R takes z_0 as the plaintext input of senders in I_L, encrypts these plaintext to obtain encrypted input 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 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 M_j which is simulated in the following way:

Let y' be the input to the second mixing step of M_j. By this point, R has extracted some permutations $\pi_0, \ldots, \pi_{2j-1}$ (from dishonest mix server before M_j). Also, R has chosen a permutation π_{2j} for the first mixing step of M_j itself. Let π_1^* be the composition of these permutations.

Let ρ be the permutation (on the set of input indices) that maps x_1 into x_0, that is $x_1[i] = x_0[\rho(i)]$. Such a permutation exists, because of the way x_0 and 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 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 π_{2j+1} (as 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 $y'' = C^{re}(\pi_{2j+1}\{y'\}, \tilde{\pi}_{2j+1}\{y'\})$.

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 M_j that do not map (via π_1^*) to indices in I_L).

Now, R computes a vector y''' from y'' by substituting every element of 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 $y'[\pi_{2j+1}(k)]$.
This vector y''' is output as the resulting ciphertexts of the second mixing step of M_j. In addition, R commits to π_{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 M_j to be opened to the right is mapped via π_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 M_j, outputs a re-encryption of $\pi_{2j+1}\{y'\}$ if the challenge bit of the re-encryption oracle is 0, or a re-encryption of $\tilde{\pi}_{2j+1}\{y'\}$ 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 π_2^* being the composition of all the permutations applied after the second mixing step of 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 $\pi^*\{x_0\} = \tilde{\pi}^*\{x_1\}$ (that is $x_0[\pi^*(i)] = x_1[\tilde{\pi}^*(i)]$).

R7. R simulates the decryption step, similarly to S7, to produce $\pi^*\{x_0\}$.

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,

$$\mathsf{Pr}[S_0^{C^{enc}(0)} \mapsto 1] = \mathsf{Pr}[R^{C^{re}(0)} \mapsto 1]. \tag{23}$$

(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 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

$$\mathsf{Pr}[S_1^{C^{enc}(0)} \mapsto 1] = \mathsf{Pr}[\tilde{S}_1^{C^{enc}(0)} \mapsto 1]. \tag{24}$$

Let $\tilde{R}$ be the system that works as R but instead of committing to the permutation π_{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

$$\mathsf{Pr}[\tilde{R}^{C^{enc}(1)} \mapsto 1] = \mathsf{Pr}[R^{C^{re}(1)} \mapsto 1]. \tag{25}$$

Now, using the observation we have already made, namely that $\tilde{\pi}^*\{x_1\}$ (the output of the system $\tilde{S}_1^{C^{enc}(0)}$) is the same as $\pi^*\{x_0\}$ (the output of $R^{C^{re}(1)}$, and hence, $\tilde{R}^{C^{re}(1)}$), we have

$$\mathsf{Pr}[\tilde{S}_1^{C^{enc}(0)} \mapsto 1] = \mathsf{Pr}[\tilde{R}^{C^{re}(1)} \mapsto 1]. \tag{26}$$

Therefore, we obtain

$$\mathsf{Pr}[S_1^{C^{enc}(0)} \mapsto 1] = \mathsf{Pr}[(R^{C^{re}(1)} \mapsto 1]. \tag{27}$$

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

$$\mathsf{Pr}[R^{C^{re}(0)} \mapsto 1] =_{neg} \mathsf{Pr}[R^{C^{re}(1)} \mapsto 1], \tag{28}$$

which, together with the above, proves (22) and concludes the proof of Lemma 5.