An Adaptively Secure Mix-Net Without Erasures

Douglas Wikström[1,⋆] and Jens Groth[2,⋆⋆]

[1] ETH Zürich, Department of Computer Science
douglas@inf.ethz.ch
[2] UCLA, Computer Science Department
jg@cs.ucla.edu

Abstract. We construct the first mix-net that is secure against *adaptive* adversaries corrupting any minority of the mix-servers and any set of senders. The mix-net is based on the Paillier cryptosystem and analyzed in the universal composability model *without erasures* under the decisional composite residuosity assumption, the strong RSA-assumption, and the discrete logarithm assumption. We assume the existence of ideal functionalities for a bulletin board, key generation, and coin-flipping.

1 Introduction

Suppose a set of senders $S_1, \ldots, S_N$ each have an input m_i, and want to compute the sorted list $(m_{\pi(1)}, \ldots, m_{\pi(N)})$ of messages, but keep the identity of the sender of any particular message m_i secret. A trusted party can provide the service required by the senders. First it collects all messages. Then it sorts the inputs and outputs the result. A protocol, i.e., a list of machines $M_1, \ldots, M_k$, that emulates the service of the trusted party as described above is called a *mix-net*, and the parties $M_1, \ldots, M_k$ are referred to as *mix-servers*. The notion of a mix-net was introduced by Chaum [3].

Many mix-net constructions are proposed in the literature without security proofs, and several of these constructions have in fact been broken. The first rigorous definition of security of a mix-net was given by Abe and Imai [1], but they did not provide any construction that satisfies their definition. Wikström [16] gives the first definition of a universally composable (UC) mix-net, and also the first construction with a complete security proof. He recently presented a more efficient UC-secure scheme [17].

An important tool in the construction of a mix-net is a so called "proof of a shuffle". This allows a mix-server to prove that it behaved as expected without leaking knowledge. The first efficient methods to achieve this were given independently by Neff [12] and Furukawa and Sako [8]. Subsequently, other authors improved and complemented these methods, e.g. [9, 7, 17]. Our results seem largely independent of the method used, but for concreteness we use the method presented in [17].

⋆ Part of the work done while at Royal Institute of Technology (KTH), Stockholm, Sweden.

⋆⋆ Supported by NSF Cybertrust ITR grant No. 0456717. Part of the work done while at Cryptomathic, Denmark and BRICS, Dept. of Computer Science, University of Aarhus, Denmark.

M. Bugliesi et al. (Eds.): ICALP 2006, Part II, LNCS 4052, pp. 276–287, 2006.

1.1 Our Contribution

All previous works consider a static adversary that decides which parties to corrupt before the protocol is executed. We provide the first efficient mix-net that is secure against an *adaptive* adversary. The problem of constructing such a scheme has been an open problem since the notion of mix-nets was proposed by Chaum [3] two decades ago. The model we consider is the *non-erasure* model, i.e., every state transition of a party is stored on a special history tape that is handed to the adversary upon corruption. It is well known that it is hard to prove the security of protocols in this model, even more so for efficient protocols. Our analysis is novel in that we show that a mix-net can be proved UC-secure even if the zero-knowledge proofs of knowledge of correct re-encryption-permutations computed by the mix-servers are not zero-knowledge against adaptive adversaries and not even straight-line extractable as is often believed to be necessary in the UC-setting. We prove our claims in the full version of this paper.

1.2 Notation

We use $S_1, \ldots, S_N$ and $M_1, \ldots, M_k$ to denote the senders and the mix-servers. All participants are modeled as interactive Turing machines with a history tape where all state transitions are recorded. Upon corruption the entire execution history is given to the adversary. We abuse notation and use S_i and M_j to denote both the machines themselves and their identity. We denote by $k' = \lceil (k+1)/2 \rceil$ the number of mix-servers needed for majority. We denote the set of permutations of N elements by Σ_N. The main security parameter is κ. The zero-knowledge proofs invoked as subprotocols use two additional security parameters, κ_c and κ_r that determine the number of bits in challenges and the statistical distance between a simulated proof and a real proof. We denote by Sort the algorithm that given a list of strings as input outputs the same set of strings in lexicographical order.

1.3 Cryptographic Model

We use the UC-framework [2], but our notation differs from [2] in that we introduce an explicit "communication model" $\mathcal{C}_{\mathcal{I}}$ that acts as a router of messages between the parties. We define $\mathcal{M}_l^*$ to be the set of adaptive adversaries that corrupt less than l out of k parties of the mix-server type, and arbitrarily many parties of the sender type. We assume an ideal authenticated bulletin board functionality $\mathcal{F}_{\mathrm{BB}}$. All parties can write to it, but no party can erase any message from it. The adversary can prevent any party from reading or writing. We also need an ideal coin-flipping functionality $\mathcal{F}_{\mathrm{CF}}$ at some points in the protocol. It simply outputs random coins when asked to do so. We take the liberty of interpreting random strings as elements in groups, e.g., in the subgroup $\mathrm{QR}_{\mathbf{N}}$.

We use the discrete logarithm (DL) assumption for safe primes $p = 2q + 1$, which says that it is infeasible to compute a discrete logarithm of a random element $y \in G_q$, where G_q is the group of squares modulo p. We use the decision composite residuosity assumption (DCR), which says that given a product n

of two random safe primes of the same size, it is infeasible to distinguish the uniform distribution on elements in $\mathbb{Z}_{\mathsf{n}^2}^*$ from the uniform distribution on nth residues in $\mathbb{Z}_{\mathsf{n}^2}^*$. We use the strong RSA-assumption (SRSA) which says that given a product $\mathbf{N}$ of two random safe primes, and $\mathbf{g} \in \mathbb{Z}_{\mathbf{N}}^*$, it is infeasible to compute $(\mathbf{b}, \eta)$ such that $\mathbf{b}^\eta = \mathbf{g} \bmod \mathbf{N}$ and $\eta \neq \pm 1$.

1.4 Distributed Paillier

We use a combination of two threshold versions of the Paillier [13] cryptosystem introduced by Lysyanskaya and Peikert [10] and Damgård et al. [5], and also modify the scheme slightly. On input 1^κ the key generator $\mathsf{KG}^{\mathsf{pai}}$ chooses two $\kappa/2$-bit safe primes p and q randomly and defines the public key $\mathsf{n} = \mathsf{pq}$. We define $\mathsf{g} = \mathsf{n} + 1$ and $\mathsf{f} = (\mathsf{p} - 1)(\mathsf{q} - 1)/4$. Then it chooses a private key d under the restriction $\mathsf{d} = 0 \bmod \mathsf{f}$ and $\mathsf{d} = 1 \bmod \mathsf{n}$ and outputs (n, d). Note that $\mathsf{g}^m = 1 + m\mathsf{n} \bmod \mathsf{n}^2$. We define $L(u) = (u - 1)/\mathsf{n}$ and have $L(\mathsf{g}^m) = m$. To encrypt a message $m \in \mathbb{Z}_\mathsf{n}$ a random $r \in \mathbb{Z}_\mathsf{n}^*$ is chosen and the ciphertext is defined by $u = E_\mathsf{n}(m, r) = \mathsf{g}^m r^{2\mathsf{n}} \bmod \mathsf{n}^2$. The decryption algorithm is defined $D_\mathsf{d}(u) = L(u^\mathsf{d} \bmod \mathsf{n}^2)$. Let $\mathsf{g}_\mathsf{f} \in \mathbb{Z}_{\mathsf{n}^2}^*$ be an element of order f. Then there exists a $2\mathsf{n}$th root r_f of g_f modulo n, and an alternative encryption algorithm is $E_{\mathsf{g}_\mathsf{f},\mathsf{n}}(m, s) = \mathsf{g}^m \mathsf{g}_\mathsf{f}^s = \mathsf{g}^m (r_\mathsf{f}^s)^{2\mathsf{n}} \bmod \mathsf{n}^2$, where s is chosen randomly in $[0, 2^{\kappa+\kappa_r} - 1]$. Here κ_r is an additional security parameter that is large enough to make $2^{-\kappa_r}$ negligible. It should always be clear from the context what is meant.

The cryptosystem is homomorphic, i.e., $E_\mathsf{n}(m_1)E_\mathsf{n}(m_2) = E_\mathsf{n}(m_1 + m_2)$. As a consequence it is possible to re-encrypt a ciphertext u using randomness $s \in \mathbb{Z}_\mathsf{n}^*$ by computing $uE_\mathsf{n}(0, s) = E_\mathsf{n}(m, rs)$, or alternatively using randomness $s \in [0, 2^{\kappa+\kappa_r} - 1]$ as $uE_{\mathsf{g}_\mathsf{f},\mathsf{n}}(0, s) = E_\mathsf{n}(m, rr_\mathsf{f}^s)$. Furthermore, given a ciphertext $K_1 = E_\mathsf{n}(1, R_1) = \mathsf{g}R_1^{2\mathsf{n}} \bmod \mathsf{n}^2$ of 1 an alternative way to encrypt a message m is to compute $E_{K_1,\mathsf{n}}(m, r) = K_1^m r^{2\mathsf{n}} \bmod \mathsf{n}^2$.

The scheme is turned into a distributed cryptosystem with k parties of which a majority k' are needed for decryption as follows. Let g and h be two random generators of a subgroup G_q of prime order q of $\mathbb{Z}_{2q+1}^*$ for a random prime $2q+1$ such that $\log_2 q > 2\kappa + \kappa_r$. Let v be a generator of the group of squares $\mathrm{QR}_{\mathsf{n}^2}$. Each party M_j is assigned a random element $\mathsf{d}_j \in [0, 2^{2\kappa+\kappa_r} - 1]$ under the restriction that $\mathsf{d} = \sum_{j=1}^k \mathsf{d}_j \bmod \mathsf{nf}$, and define $\mathsf{v}_j = \mathsf{v}^{\mathsf{d}_j} \bmod \mathsf{n}^2$. We also compute a Shamir-secret sharing [15] of each d_j to allow reconstruction of this value. More precisely we choose for each j a random $(k'-1)$-degree polynomial f_j over $\mathbb{Z}_q$ under the restriction that $f_j(0) = \mathsf{d}_j$, and define $\mathsf{d}_{j,l} = f_j(l) \bmod q$. A Pedersen [14] commitment $F_{j,l} = g^{\mathsf{d}_{j,l}} h^{\mathsf{t}_{j,l}}$ of each $\mathsf{d}_{j,l}$ is also computed, where $\mathsf{t}_{j,l} \in \mathbb{Z}_q$ is randomly chosen. The joint public key consists of $(\mathsf{n}, \mathsf{v}, (\mathsf{v}_j)_{j=1}^k, (F_{j,l})_{j,l \in \{1,\ldots,k\}})$. The private key of M_j consists of $(\mathsf{d}_j, (\mathsf{d}_{l,j}, \mathsf{t}_{l,j})_{l=1}^k)$.

To jointly decrypt a ciphertext u, the jth share-holder computes $u_j = u^{\mathsf{d}_j} \bmod \mathsf{n}^2$ and proves in zero-knowledge that $\log_u u_j = \log_\mathsf{v} \mathsf{v}_j$. If the proof fails, each M_l publishes $(\mathsf{d}_{j,l}, \mathsf{t}_{j,l})$. Then each honest party finds a set of $(\mathsf{d}_{j,l}, \mathsf{t}_{j,l})$ such that $F_{j,l} = g^{\mathsf{d}_{j,l}} h^{\mathsf{t}_{j,l}}$, recovers d_j using Lagrange interpolation, and computes $u_j = u^{\mathsf{d}_j} \bmod \mathsf{n}^2$. Finally, the plaintext is given by $L(\prod_{j=1}^k u_j) = m$.

2 The Ideal Relaxed Mix-Net

We use a slightly relaxed definition of the ideal mix-net in that corrupt senders may input messages with κ bits whereas honest senders may only input messages with $\kappa_m = \kappa - \kappa_r - 2$ bits. In the final output all messages are truncated to κ_m bits. The additional security parameter κ_r must be chosen such that $2^{-\kappa_r}$ is negligible. It decides the statistical hiding properties of some subprotocols. It is hard to imagine a situation where the relaxation is a real disadvantage, but if it is, it may be possible to eliminate this beauty flaw by an erasure-free proof of membership in the correct interval in the submission phase of the protocol.

Functionality 1 (Relaxed Mix-Net). The relaxed ideal mix-net, $\mathcal{F}_{\mathrm{RMN}}$, running with mix-servers $M_1, \ldots, M_k$, senders $S_1, \ldots, S_N$, and ideal adversary $\mathcal{S}$ proceeds as follows

1. Initialize a list $L = \emptyset$, a database D, $c = 0$, and $J_S = \emptyset$ and $J_M = \emptyset$.
2. Repeatedly wait for new inputs and do
   - Upon receipt of $(S_i, \texttt{Send}, m_i)$ from $\mathcal{C_I}$ do the following. If $i \notin J_S$ and S_i is not corrupted and $m_i \in [-2^{\kappa_m} + 1, 2^{\kappa_m} - 1]$ or if S_i is corrupted and $m_i \in [-2^{\kappa} + 1, 2^{\kappa} - 1]$ then set $c \leftarrow c + 1$, store this tuple in D under the index c, and hand $(\mathcal{S}, S_i, \texttt{Input}, c)$ to $\mathcal{C_I}$. Ignore other inputs.
   - Upon receipt of $(M_j, \texttt{Run})$ from $\mathcal{C_I}$, set $c \leftarrow c + 1$, store $(M_j, \texttt{Run})$ in D under the index c, and hand $(\mathcal{S}, M_j, \texttt{Input}, c)$ to $\mathcal{C_I}$.
   - Upon receipt of $(\mathcal{S}, \texttt{AcceptInput}, c)$ such that something is stored under the index c in D do
     (a) If $(S_i, \texttt{Send}, m_i)$ is stored under c and $i \notin J_S$, then append m_i to the list L, set $J_S \leftarrow J_S \cup \{i\}$, and hand $(\mathcal{S}, S_i, \texttt{Send})$ to $\mathcal{C_I}$.
     (b) If $(M_j, \texttt{Run})$ is stored under c, then set $J_M \leftarrow J_M \cup \{j\}$. If $|J_M| > k/2$, then truncate all strings in L to κ_m bits and sort the result lexicographically to form a list L'. Sort the list L to form a list L''. Then hand $((\mathcal{S}, M_j, \texttt{Output}, L''), \{(M_l, \texttt{Output}, L')\}_{l=1}^{k})$ to $\mathcal{C_I}$. Otherwise, hand $\mathcal{C_I}$ the list $(\mathcal{S}, M_j, \texttt{Run})$.

3 The Adaptively Secure Mix-Net

In this section we first describe the basic structure of our mix-net. Then we explain how we modify this to accommodate adaptive adversaries. We also discuss how and why our construction differs from previous constructions in the literature. This is followed by subsections introducing the subprotocols invoked in an execution of the mix-net. Finally, we give a detailed description of the mix-net.

3.1 Key Generation

The mix-servers use a joint κ-bit Paillier public key n and a corresponding secret shared secret key as described above. The public key n is the main public key in the mix-net, but we do need additional keys. We denote by $(\mathsf{n}', \mathsf{g}', \mathsf{d}')$ Paillier parameters generated as above but such that $\mathsf{n}' > \mathsf{n}$. We also need an RSA modulus $\mathbf{N}$ that is chosen exactly as the Paillier moduli n and n'. Finally, we need two Paillier ciphertexts $K_0 = E_{\mathsf{n}}(0, R_0)$ and $K_1 = E_{\mathsf{n}}(1, R_1)$ of 0 and 1 respectively. Below we summarize key generation as an ideal functionality.

Functionality 2 (Key Generation). The ideal key generation functionality, $\mathcal{F}_{\mathrm{PKG}}$, running with mix-servers $M_1, \ldots, M_k$, senders $S_1, \ldots, S_N$, and ideal adversary $\mathcal{S}$ proceeds as follows. It generates keys as described above and hands $((\mathcal{S}, \texttt{PublicKeys}, (\mathbf{N}, g, h, \mathsf{n}', \mathsf{n}, K_0, K_1, \mathsf{v}, (\mathsf{v}_l)_{l=1}^k, (F_{l,l'})_{l,l' \in \{1,\ldots,k\}})),$
$\{(M_j, \texttt{Keys}, (\mathbf{N}, g, h, \mathsf{n}', \mathsf{n}, K_0, K_1, \mathsf{v}, (\mathsf{v}_l)_{l=1}^k, (F_{l,l'})_{l,l' \in \{1,\ldots,k\}}),$
$(\mathsf{d}_j, (\mathsf{d}_{l,j}, \mathsf{t}_{l,j})_{l=1}^k))\}_{j=1}^k)$ to $\mathcal{C}_{\mathcal{I}}$.

3.2 The Overall Structure

Our mix-net is based on the re-encryption-permutation paradigm. Let $L_0 = \{u_{0,i}\}_{i=1}^N$ be the list of ciphertexts submitted by senders. For $l = 1, \ldots, k$ the lth mix-server M_l re-encrypts each element in $L_{l-1} = \{u_{l-1,i}\}_{i=1}^N$ as explained in Section 1.4, sorts the resulting list and publishes the result as L_l. Then it proves, in zero-knowledge, knowledge of a witness that L_{l-1} and L_l are related in this way. The mix-servers then jointly and verifiably re-encrypt the ciphertexts in L_k. Note that no permutation takes place in this step. The result is denoted by L_{k+1}. Finally, the mix-servers jointly and verifiably decrypt each ciphertext in L_{k+1} and sort the resulting list of plaintexts to form the output. Except for the joint re-encryption step this is similar to several previous constructions.

3.3 Accommodating Adaptive Adversaries

To extract the inputs of corrupt senders, each sender forms two ciphertexts $u_{0,i}$ and $u'_{0,i}$ and proves that the same plaintext is hidden in both. Naor and Yung's [11] double-ciphertext trick then allows extraction. Submissions of honest senders must be simulated without knowing which message they actually hand to $\mathcal{F}_{\mathrm{RMN}}$. A new problem in the adaptive setting is that the adversary may corrupt a simulated honest sender S_i that has already computed fake ciphertexts $u_{0,i}$ and $u'_{0,i}$. The ideal adversary can of course corrupt the corresponding dummy party $\tilde{S}_i$ and retrieve the true value m_i it handed to $\mathcal{F}_{\mathrm{RMN}}$. The problem is that it must provide S_i with a plausible history tape that convinces the adversary that S_i sent m_i already from the beginning. To solve this problem we adapt an idea of Damgård and Nielsen [6]. We have two public keys $K_0 = E_{\mathsf{n}}(0, R_0) = R_0^{2\mathsf{n}} \bmod \mathsf{n}^2$ and $K_1 = E_{\mathsf{n}}(1, R_1) = \mathsf{g} R_1^{2\mathsf{n}} \bmod \mathsf{n}^2$ and each sender is given a unique key $K'_i = E_{\mathsf{n}'}(a_i)$ for a randomly chosen $a_i \in \mathbb{Z}_{\mathsf{n}'}$. The sender of a message m_i chooses $b_i \in \mathbb{Z}_{\mathsf{n}}$, $r_i \in \mathbb{Z}_{\mathsf{n}}^*$ and $r'_i \in \mathbb{Z}_{\mathsf{n}'}^*$ randomly, and computes its two ciphertexts as follows $u_i = E_{K_1,\mathsf{n}}(m_i, r_i)$ and $u'_i = E_{(\mathsf{g}')^{b_i} K'_i, \mathsf{n}'}(m_i, r'_i)$. Then it submits (b_i, u_i, u'_i) and proves in zero-knowledge that the same message m_i is encrypted in both ciphertexts. Note that $D_{\mathsf{d}}(u_i) = m_i$ and $D_{\mathsf{d}'}(u'_i) = (a_i + b_i) m_i$ due to the homomorphic property of the cryptosystem.

During simulation we instead define $K_0 = E_{\mathsf{n}}(1, R_0) = \mathsf{g}R_0^{2\mathsf{n}} \bmod \mathsf{n}^2$ and $K_1 = E_{\mathsf{n}}(0, R_1) = R_1^{2\mathsf{n}} \bmod \mathsf{n}^2$. This means that u_i becomes an encryption of 0 for all senders. Furthermore, simulated senders choose $b_i = -a_i \bmod \mathsf{n}'$ which implies that also u'_i is an encryption of 0. The important property of the simulation is that given m_i and R_1 we can define $\bar{r}_i = r_i / R_1^{m_i}$ such that $u_i = E_{K_1,\mathsf{n}}(m_i, \bar{r}_i)$, i.e., we can open a simulated ciphertext as an encryption of an arbitrary message m_i. The ciphertext u'_i can also be opened as an encryption of m_i in a similar way when $b_i + a_i = 0 \bmod \mathsf{n}'$. Finally, the proof of equality we use can also be "opened" in a convincing way. This allows the simulator to simulate honest senders and produce plausible history tapes as required. Corrupt senders on the other hand have negligible probability of guessing a_i, so the simulator can extract the message submitted by corrupt senders using only the private key d' by computing $m_i = D_{\mathsf{d}'}(u'_i)/(a_i + b_i) \bmod \mathsf{n}'$. Before the mix-net simulated by the ideal adversary starts to process the input ciphertexts the ideal mix-net $\mathcal{F}_{\mathrm{RMN}}$ has handed the ideal adversary the list of plaintexts that should be output by the simulation. All plaintexts equal zero in the ciphertexts of the input in the simulation and the correct messages are introduced in the joint re-encryption phase. All mix-servers are simulated honestly during the re-encryption-permutation phase and the decryption phase.

The joint re-encryption is defined as follows. Before the mixing each mix-server is given a random ciphertext $\bar{K}'_j$ using the public key n'. Each mix-server M_j chooses random elements $m_{j,i} \in \mathbb{Z}_{\mathsf{n}}$ and commits to these by choosing $\bar{b}_j \in \mathbb{Z}_{\mathsf{n}'}$ and $s'_{j,i} \in \mathbb{Z}^*_{\mathsf{n}'}$ randomly and computing $w'_{j,i} = E_{(\mathsf{g}')^{\bar{b}_j} \bar{K}'_j,\mathsf{n}'}(m_{j,i}, s'_{j,i})$. When all mix-servers have published their commitments, it chooses $s_{j,i} \in \mathbb{Z}^*_{\mathsf{n}}$ randomly and computes $w_{j,i} = E_{K_0,\mathsf{n}}(m_{j,i}, s_{j,i})$. It also proves in zero-knowledge that the same random element $m_{j,i}$ is encrypted in both ciphertexts. The jointly re-encrypted elements $u_{k+1,i}$ are then formed as $u_{k+1,i} = u_{k,i} \prod_{l \in I} w_{l,i}^{\prod_{l' \neq l} \frac{l'}{l'-l}}$ where I is the first set of k' indices j such that the proof of M_j is valid. In the real execution this is an elaborate way to re-encrypt $u_{k,i}$, since K_0 is an encryption of 0. In the simulation on the other hand the ideal adversary chooses $\bar{b}_j = -\bar{a}_j \bmod \mathsf{n}'$ and sets $m_{j,i} = 0$ for simulated mix-servers and extracts the $m_{j,i}$ values of corrupt mix-servers from their commitments. It then redefines the $m_{j,i}$ values of simulated honest mix-servers such that $f_i(j) = m_{j,i}$ for a $(k'-1)$-degree polynomial f_i over $\mathbb{Z}_{\mathsf{n}}$ such that $f_i(0)$ equals $m_{\pi(i)}$ for some random permutation $\pi \in \Sigma_N$. Since $\bar{b}_j + \bar{a}_j = 0 \bmod \mathsf{n}'$ it can compute $\bar{s}'_{j,i}$ such that $w'_{j,i} = E_{(\mathsf{g}')^{\bar{b}_j} \bar{K}'_j,\mathsf{n}'}(m_{j,i}, \bar{s}'_{j,i})$. In the simulation K_0 is an encryption of 1 and each $u_{k,i}$ is an encryption of zero, which implies that $u_{k+1,i}$ becomes an encryption of $m_{\pi(i)}$ as required. The adversary can not tell the difference since it can only get its hands on a minority of the $m_{j,i}$ values directly, and the semantic security of the cryptosystem prevents it from knowing these values otherwise.

3.4 Some Intuition Behind Our Analysis

Intuitively, the soundness of the subprotocols ensure that each sender knows the message it submits and that the output of the mix-net is correct. The zero-knowledge properties and the knowledge extraction properties of the subprotocols are not used by the ideal adversary sketched above, but they are essential to prove that the ideal adversary produces an indistinguishable simulation.

The private key d corresponding to the Paillier modulus n is needed both in the ideal model and in the real protocol. Thus, even if the environment can distinguish the ideal model from the real model, we can not use it directly to reach a contradiction to the semantic security of the Paillier cryptosystem. To solve this problem we use the single-honest-player proof strategy to sample each of the two distributions, but without using the secret key d. The knowledge extractor of the proof of a shuffle is needed to be able to simulate the joint decryption, since although the set of plaintexts is known their order in the list of ciphertexts that are jointly decrypted is not. Due to the statistical zero-knowledge property of the proof of a shuffle and the fact that in the ideal model all plaintexts are zero from the start we can use the same type of simulation also when sampling the ideal model without changing its distribution more than negligibly. A hybrid argument allows us to assume that the simulated honest senders use the correct plaintexts already from the start. If there is a gap between the resulting distributions this can be used to distinguish a ciphertext of a zero from a ciphertext of a one, i.e., we can break the semantic security of the Paillier cryptosystem.

3.5 Differences with Previous Constructions

Most previous schemes are based on the ElGamal cryptosystem. We need the Paillier cryptosystem to allow adaptive corruption of the senders in the way explained above. The joint re-encryption step which no previous construction has is needed to insert the correct messages in the simulation and still be able to construct plausible history tapes of any adaptively corrupted mix-server.

In [16, 17] the mix-net is given in a hybrid model with access to ideal zero-knowledge proof of knowledge functionalities. These functionalities are then securely realized, and the composition theorem of the UC-framework invoked. The modular approach simplifies the analysis, but the strong demands on subprotocols make them hard to securely realize efficiently. We avoid this problem by showing that a zero-knowledge proof of knowledge of correct re-encryption-permutation in the classical sense is sufficient, i.e., the protocol can *not be simulated to an adaptive adversary* and extraction is *not straight-line*.

3.6 Subprotocols Invoked by the Main Protocol

Some of our subprotocols satisfy a weaker notion of proof of knowledge called "computationally convincing proof (of knowledge)" introduced by Damgård and Fujisaki [4]. Informally, this means that extraction is possible with overwhelming probability over the randomness of a special input that is given to both parties.

Proof of Knowledge of Re-encryption-Permutation. Denote by $\pi_{\mathrm{prp}} = (P_{\mathrm{prp}}, V_{\mathrm{prp}})$ the 5-move protocol for proving knowledge of a witness of a re-encryption and permutation of a list of Paillier ciphertexts given by Wikström

[17]. The parties accept as special parameters an RSA-modulus $\mathbf{N}$ and random $\mathbf{g}, \mathbf{h} \in \mathrm{QR}_{\mathbf{N}}$, and random $g_1, \ldots, g_N \in G_q$. The re-encryption-permutation relation $R_{\mathrm{RP}}^{\mathsf{n},\mathsf{g_f}}$ and the security properties of π_{prp} are stated below.

Definition 1 (Knowledge of Correct Re-encryption-Permutation). *Define for each N, n and $\mathsf{g_f}$ a relation $R_{\mathrm{RP}}^{\mathsf{n},\mathsf{g_f}} \subset (\mathrm{QR}_{\mathsf{n}^2}^N \times \mathrm{QR}_{\mathsf{n}^2}^N) \times [-2^{\kappa+\kappa_r} + 1, 2^{\kappa+\kappa_r} - 1]^N$, by $((\{u_i\}_{i=1}^N, \{u_i'\}_{i=1}^N), (a, (x_i)_{i=1}^N)) \in R_{\mathrm{RP}}^{\mathsf{n},\mathsf{g_f}}$ precisely when $a < \sqrt{\mathsf{n}}/4$ equals one or is prime and $(u_i')^a = \mathsf{g_f}^{x_{\pi(i)}} u_{\pi(i)}^a \mod \mathsf{n}^2$ for $i = 1, \ldots, N$ and some permutation $\pi \in \Sigma_N$ such that the list $\{u_i'\}_{i=1}^N$ is sorted lexicographically.*

Proposition 1 ([17]). *The protocol π_{prp} is an honest verifier statistical zero-knowledge computationally convincing proof of knowledge for the relation $R_{\mathrm{RP}}^{\mathsf{n},\mathsf{g_f}}$ with respect to the distribution of $(\mathbf{N}, \mathbf{g}, \mathbf{h})$ and $(g_1, \ldots, g_N)$, and it has overwhelming completeness.*

Proof of Equality of Plaintexts. When a sender submits its ciphertexts u_i and u_i' it must prove that they are encryptions of the same $(\kappa - 2)$-bit integer under two distinct public keys. The protocol $\pi_{eq} = (P_{\mathrm{eq}}, V_{\mathrm{eq}})$ used to do this is given below. The security parameters κ_c and κ_r decide the soundness and statistical zero-knowledge property of the protocol.

Protocol 1 (Proof of Equal Plaintexts Using Distinct Moduli)
COMMON INPUT: $\mathsf{n} \in \mathbb{Z}$, $K, u \in \mathbb{Z}_{\mathsf{n}^2}^*$, $\mathsf{n}' \in \mathbb{Z}$, $K', u' \in \mathbb{Z}_{(\mathsf{n}')^2}^*$, $\mathbf{N} \in \mathbb{N}$, *generators* $\mathbf{g}$ *and* $\mathbf{h}$ *of* $\mathrm{QR}_{\mathbf{N}}$.
PRIVATE INPUT: $m \in [-2^{\kappa_m} + 1, 2^{\kappa_m} - 1]$, $r \in \mathbb{Z}_{\mathsf{n}}^*$, *and* $r' \in \mathbb{Z}_{\mathsf{n}'}^*$ *such that* $u = E_{K,\mathsf{n}}(m, r)$ *and* $u' = E_{K',\mathsf{n}'}(m, r')$.

1. *The prover chooses* $r'' \in [0, 2^{\kappa+\kappa_r} - 1]$, $s_0 \in \mathbb{Z}_{\mathsf{n}^2}^*$ *and* $s_1 \in \mathbb{Z}_{(\mathsf{n}')^2}^*$, *and* $t \in [0, 2^{\kappa_m+\kappa_c+\kappa_r} - 1]$ *and* $s_2, \in [0, 2^{2\kappa+\kappa_c+2\kappa_r} - 1]$ *randomly. Then it computes* $C = \mathbf{g}^m \mathbf{h}^{r''} \mod \mathbf{N}$ *and* $(\alpha_0, \alpha_1, \alpha_2) = (K^t s_0^{2\mathsf{n}} \mod \mathsf{n}^2, (K')^t s_1^{2\mathsf{n}'} \mod (\mathsf{n}')^2, \mathbf{g}^t \mathbf{h}^{s_2} \mod \mathbf{N})$, *and hands* $(C, \alpha_0, \alpha_1, \alpha_2)$ *to the verifier.*
2. *The verifier chooses* $c \in [2^{\kappa_c - 1}, 2^{\kappa_c} - 1]$ *and hands* c *to the prover.*
3. *The prover computes* $(e_0, e_1) = (r^c s_0 \mod \mathsf{n}, (r')^c s_1 \mod \mathsf{n}')$, $(e_2, e_3) = (cr'' + s_2 \mod 2^{\kappa+\kappa_c+2\kappa_r}, cm + t \mod 2^{\kappa_m+\kappa_c+\kappa_r})$ *and hands* (e_0, e_1, e_2, e_3) *to the verifier.*
4. *The verifier checks* $(u^c \alpha_0, (u')^c \alpha_1) = (K^{e_3} e_0^{2\mathsf{n}} \mod \mathsf{n}, (K')^{e_3} e_1^{2\mathsf{n}'} \mod \mathsf{n}')$ *and* $C^c \alpha_2 = \mathbf{g}^{e_3} \mathbf{h}^{e_2} \mod \mathbf{N}$.

The protocol is statistical zero-knowledge, but this is not enough since we must construct plausible history tapes for simulated senders.

Proposition 2 ("Zero-Knowledge"). *Let* $K = R^{2\mathsf{n}} \mod \mathsf{n}^2$ *and* $K' = {R'}^{2\mathsf{n}'} \mod (\mathsf{n}')^2$ *for some* $R \in \mathbb{Z}_{\mathsf{n}}^*$ *and* $R' \in \mathbb{Z}_{\mathsf{n}'}^*$. *Let* $\mathbf{h}$ *be a generator of* $\mathrm{QR}_{\mathbf{N}}$ *and* $\mathbf{g} = \mathbf{h}^{\mathbf{x}}$. *Let* r, r', *and* (r'', s_0, s_1, t, s_2) *be randomly distributed in the domains described in the protocol, and denote by* $I(m) = (\mathsf{n}, K, u, \mathsf{n}', K', u', \mathbf{N}, \mathbf{g}, \mathbf{h})$ *the common input corresponding to the private input* (m, r, r'). *Denote by* c *the random challenge from the verifier and let* $T(m) = (\alpha, c, e)$ *be the proof transcript induced by* (m, r, r'), c, *and* (r'', s_0, s_1, t, s_2).

There is a deterministic polynomial-time algorithm His *such that for every* $m \in \{0,1\}^{\kappa_m}$ *with* $(\bar{r}, \bar{r}', \bar{r}'', \bar{s}_0, \bar{s}_1, \bar{t}, \bar{s}_2) = \mathsf{His}(R, R', \mathbf{x}, m, r, r', r'', s_0, s_1, t, s_2, c)$ *the distributions of* $[I(m), T(m), (m, r, r'), (r'', s_0, s_1, t, s_2)]$ *and* $[I(0), T(0), (m, \bar{r}, \bar{r}'), (\bar{r}'', \bar{s}_0, \bar{s}_1, \bar{t}, \bar{s}_2))]$ *are statistically close.*

The proposition in itself does not imply statistical zero-knowledge, since it only applies to inputs where K and K' are both encryptions of zero.

Proposition 3. *The protocol is a computationally convincing proof with respect to the distribution of* $(\mathbf{N}, \mathbf{g}, \mathbf{h})$*, and has overwhelming completeness.*

Multiple instances of the protocol can be run in parallel using the same RSA-parameters and same challenge. Thus, we use the protocol also for common inputs on the form $(\mathsf{n}, K, \{u_i\}_{i=1}^N, \mathsf{n}', K', \{u_i'\}_{i=1}^N, \mathbf{N}, \mathbf{g}, \mathbf{h})$ and with corresponding private input $(\{m_i\}_{i=1}^N, \{r_i\}_{i=1}^N, \{r_i'\}_{i=1}^N)$. We extend the notation in the next subsection similarly.

Proof of Equality of Exponents. During joint decryption of a ciphertext u each mix-server computes $u^{\mathsf{d}_j} \bmod \mathsf{n}^2$ using its part d_j of the private key, and proves correctness relative $\mathsf{v}_j = \mathsf{v}^{\mathsf{d}_j} \bmod \mathsf{n}^2$, i.e., that it uses the same exponent d_j for both elements. We denote by $\pi_{\exp} = (P_{\exp}, V_{\exp})$ the 3-move protocol proposed in [5]. It has the following properties.

Proposition 4. *The protocol* $\pi_{\exp}$ *is an honest verifier statistical zero-knowledge proof with overwhelming completeness.*

3.7 The Mix-Net

We are now ready to give a detailed description of the mix-net. Recall that $k' = \lceil (k+1)/2 \rceil$ denotes the number of mix-servers needed for majority. Each entry on the bulletin board is given a sequence number denoted by T below (with different subscripts). To ensure that the ciphertexts in the common inputs to the proofs of a shuffle belong to $\mathrm{QR}_{\mathsf{n}^2}$ the mix-servers square the ciphertexts between each mix-server. The effect of the squaring is eliminated at the end.

Protocol 2 (Mix-Net). The mix-net $\pi_{\mathrm{RMN}} = (S_1, \ldots, S_N, M_1, \ldots, M_k)$ consists of senders S_i, and mix-servers M_j.

SENDER S_i. Each sender S_i proceeds as follows.

1. Wait until $(M_l, \mathsf{n}, K_1, \mathsf{n}', \{K_i'\}_{i=1}^N, \mathbf{N}, \mathbf{g}, \mathbf{h})$ appears on $\mathcal{F}_{\mathrm{BB}}$ for k' distinct indices l.
2. Wait for an input $(\texttt{Send}, m_i)$, such that $m_i \in [-2^{\kappa_m}+1, 2^{\kappa_m}-1]$. Choose $r_i \in \mathbb{Z}_{\mathsf{n}}^*$, $b_i \in \mathbb{Z}_{\mathsf{n}'}$ and $r_i' \in \mathbb{Z}_{\mathsf{n}'}^*$ randomly and compute $u_i = E_{K_1,\mathsf{n}}(m_i, r_i)$, $u_i' = E_{(\mathsf{g}')^{b_i} K_i', \mathsf{n}'}(m_i, r_i')$, and $(\alpha_i, \mathrm{state}_i) = P_{\mathrm{eq}}((\mathsf{n}, K_1, u_i, \mathsf{n}', (\mathsf{g}')^{b_i} K_i', u_i', \mathbf{N}, \mathbf{g}, \mathbf{h}), (m_i, r_i, r_i'))$. Then hand $(\texttt{Write}, \texttt{Submit}, (b_i, u_i, u_i'), \texttt{Commit}, \alpha_i)$ to $\mathcal{F}_{\mathrm{BB}}$.
3. Wait until $(M_j, \texttt{Challenge}, S_i, c_i)$ appears on $\mathcal{F}_{\mathrm{BB}}$ for k' distinct j with identical c_i. Then compute $e_i = P_{\mathrm{eq}}(\mathrm{state}_i, c_i)$ and hand $(\texttt{Write}, \texttt{Reply}, e_i)$ to $\mathcal{F}_{\mathrm{BB}}$.

MIX-SERVER M_j. Each mix-server M_j proceeds as follows.

Preliminaries

1. Wait for a message on the form $(\texttt{Keys}, (\mathbf{N}, g, h, \mathsf{n}', \mathsf{n}, K_0, K_1, \mathsf{v}, (\mathsf{v}_l)_{l=1}^k, (F_{l,l'})_{l,l' \in \{1,\ldots,k\}}), (\mathsf{d}_j, (\mathsf{d}_{l,j}, \mathsf{t}_{l,j})_{l=1}^k))$ from $\mathcal{F}_{\mathrm{PKG}}$.
2. Hand $(\texttt{GenerateCoins}, (N+k)(\kappa+\kappa_r)+(\kappa+\kappa_r)+2(\kappa+\kappa_r)+N(\kappa+\kappa_r))$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, \{K_i'\}_{i=1}^N, \{\bar{K}_j'\}_{j=1}^k, \mathsf{g_f}, \mathbf{g}, \mathbf{h}, g_1, \ldots, g_N)$. Then hand $(\texttt{Write}, \mathsf{n}, K_1, \mathsf{n}', \{K_i'\}_{i=1}^N, \mathbf{N}, \mathbf{g}, \mathbf{h})$ to $\mathcal{F}_{\mathrm{BB}}$.

Reception of Inputs

3. Initialize $L_0 = \emptyset$, $J_S = \emptyset$ and $J_M = \emptyset$.
4. Repeat
   (a) When given input $(\texttt{Run})$ hand $(\texttt{Write}, \texttt{Run})$ to $\mathcal{F}_{\mathrm{BB}}$.
   (b) When a new entry $(T, M_l, \texttt{Run})$ appears on $\mathcal{F}_{\mathrm{BB}}$ set $J_M \leftarrow J_M \cup \{l\}$ and if $|J_M| \geq k'$ set $T_{\mathrm{run}} = T$ and go to Step 5.
   (c) When a new entry $(S_i, \texttt{Submit}, (b_i, u_i, u_i'), \texttt{Commit}, \alpha_i)$ appears on $\mathcal{F}_{\mathrm{BB}}$ such that $i \notin J_S$, set $J_S \leftarrow J_S \cup \{i\}$ and hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_i)$. Hand $(\texttt{Write}, \texttt{Challenge}, S_i, c_i)$ to $\mathcal{F}_{\mathrm{BB}}$.
5. Request the contents on $\mathcal{F}_{\mathrm{BB}}$ with index less than T_{run}. Find for each i the first occurrences of entries on the forms $(T_i, \texttt{Submit}, (b_i, u_i, u_i'), \texttt{Commit}, \alpha_i)$, $(T_{j,i}', M_j, \texttt{Challenge}, S_i, c_i)$, and $(T_i'', S_i, \texttt{Reply}, e_i)$. Then form a list L_0 of all ciphertexts $u_i^2 \bmod \mathsf{n}^2$ such that $T_i < T_{j,i}' < T_i'' < T_{\mathrm{run}}$ for at least k' distinct indices j and $V_{\mathrm{eq}}(\mathsf{n}, K_1, u_i, \mathsf{n}', (\mathsf{g}')^{b_i} K_i', u_i', \mathbf{N}, \mathbf{g}, \mathbf{h}, \alpha_i, c_i, e_i) = 1$.

Re-encryption and Permutation

6. Write $L_0 = \{u_{0,i}\}_{i=1}^{N'}$ for some N'. Then for $l = 1, \ldots, k$ do
   (a) If $l = j$, then do
      i. Choose $r_{j,i} \in [0, 2^{\kappa+\kappa_r} - 1]$ randomly, compute
      $$L_j = \{u_{j,i}\}_{i=1}^{N'} = \mathsf{Sort}(\{\mathsf{g_f}^{r_{j,i}} u_{j-1,i}^2 \bmod \mathsf{n}^2\}_{i=1}^{N'}) \ , \ \text{ and}$$
      $$(\alpha_j, \mathrm{state}_j) = P_{\mathrm{prp}}(\mathsf{n}, \mathsf{g_f}, L_{l-1}^4, L_l^2, \mathbf{N}, \mathbf{g}, \mathbf{h}, g, g_1, \ldots, g_{N'}, \{2r_{j,i}\}_{i=1}^{N'}) \ ,$$
      and hand $(\texttt{Write}, \texttt{List}, L_j, \texttt{Commit1}, \alpha_j)$ to $\mathcal{F}_{\mathrm{BB}}$. The exponentiations L_{l-1}^4 and L_l^2 should be interpreted term-wise.
      ii. Hand $(\texttt{GenerateCoins}, \kappa)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_j)$. Then compute $(\alpha_j', \mathrm{state}_j') = P_{\mathrm{prp}}(\mathrm{state}_j, c_j)$ and hand $(\texttt{Write}, \texttt{Commit2}, \alpha_j')$ to $\mathcal{F}_{\mathrm{BB}}$.
      iii. Hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_j')$. Then compute $e_j = P_{\mathrm{prp}}(\mathrm{state}_j', c_j')$ and hand $(\texttt{Write}, \texttt{Reply}, e_j)$ to $\mathcal{F}_{\mathrm{BB}}$.
   (b) If $l \neq j$, then do
      i. Wait until an entry $(M_l, \texttt{List}, L_l, \texttt{Commit1}, \alpha_l)$ appears on $\mathcal{F}_{\mathrm{BB}}$.
      ii. Hand $(\texttt{GenerateCoins}, \kappa)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_l)$.

iii. Wait for a new entry $(M_l, \texttt{Commit2}, \alpha'_l)$ on $\mathcal{F}_{\mathrm{BB}}$. Hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c'_l)$.
iv. Wait for a new entry $(M_l, \texttt{Reply}, e_l)$ on $\mathcal{F}_{\mathrm{BB}}$ and compute $b_l = V_{\mathrm{prp}}(\mathsf{n}, \mathsf{g_f}, L^4_{l-1}, L^2_l, \mathbf{N}, \mathbf{g}, \mathbf{h}, g, g_1, \ldots, g_{N'}, \alpha_l, c_l, \alpha'_l, c'_l, e_l)$.
v. If $b_l = 0$, then set $L_l = L^2_{l-1}$.

Joint Re-encryption

7. Choose $\bar{b}_j \in \mathbb{Z}_{\mathsf{n}'}$, $m_{j,i} \in \mathbb{Z}_{\mathsf{n}'}$ and $s'_{j,i} \in \mathbb{Z}^*_{\mathsf{n}'}$ randomly and compute $W'_j = \{w'_{j,i}\}_{i=1}^{N'} = \{E_{\mathsf{g}'^{\bar{b}_j} \bar{K}'_j, \mathsf{n}'}(m_{j,i}, s'_{j,i})\}_{i=1}^{N'}$. Hand $(\texttt{Write}, \texttt{RandExp}, \bar{b}_j, W'_j)$ to $\mathcal{F}_{\mathrm{BB}}$.
8. Wait until $(\texttt{RandExp}, \bar{b}_l, W'_l)$ appears on $\mathcal{F}_{\mathrm{BB}}$ for $l = 1, \ldots, k$. Then choose $s_{j,i} \in \mathbb{Z}^*_{\mathsf{n}}$ randomly, compute $W_j = \{w_{j,i}\}_{i=1}^{N'} = \{E_{K_0,\mathsf{n}}(m_{j,i}, s_{j,i})\}_{i=1}^{N'}$, and
$$(\alpha_j, \mathrm{state}_j) = P_{\mathrm{eq}}((\mathsf{n}, K_0, W_j, \mathsf{n}', K', W'_j, \mathbf{N}, \mathbf{g}, \mathbf{h}), (\{m_{j,i}\}_{i=1}^{N'}, \{s_{j,i}\}_{i=1}^{N'}, \{s'_{j,i}\}_{i=1}^{N'})) \ ,$$
and hand $(\texttt{Write}, \texttt{RandExp}, W_j, \texttt{Commit}, \alpha_j)$ to $\mathcal{F}_{\mathrm{BB}}$.
9. Wait until $(\texttt{RandExp}, W_l, \texttt{Commit}, \alpha_l)$ appears on $\mathcal{F}_{\mathrm{BB}}$ for $l = 1, \ldots, k$. Hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c)$. Compute $e_j = P_{\mathrm{eq}}(\mathrm{state}_j, c)$ and hand $(\texttt{Write}, \texttt{Reply}, e_j)$ to $\mathcal{F}_{\mathrm{BB}}$.
10. Wait until $(\texttt{Reply}, e_l)$ appears on $\mathcal{F}_{\mathrm{BB}}$ for $l = 1, \ldots, k$. Let I be the first set of k' indices with $V_{\mathrm{eq}}(\mathsf{n}, K_0, W_l, \mathsf{n}', K', W'_l, \mathbf{N}, \mathbf{g}, \mathbf{h}, \alpha_l, c, e_l) = 1$.
11. Compute $L_{k+1} = \{u_{k+1,i}\}_{i=1}^{N'} = \left\{ u_{k,i} \prod_{l \in I} w_{l,i}^{\prod_{l' \neq l} \frac{l'}{l'-l}} \right\}_{i=1}^{N'}$.

Joint Decryption

12. Compute $\Gamma_j = \{v_{j,i}\}_{i=1}^{N'} = \{u_{k+1,i}^{2\mathsf{d}_j}\}_{i=1}^{N}$ using d_j and a proof $(\alpha_j, \mathrm{state}_j) = P_{\mathrm{exp}}((\mathsf{n}, \mathsf{v}, \mathsf{v}_j, L^2_{k+1}, \Gamma_j), \mathsf{d}_j)$. Then hand $(\texttt{Write}, \texttt{Decrypt}, \Gamma_j, \texttt{Commit}, \alpha_j)$ to $\mathcal{F}_{\mathrm{BB}}$, where exponentiation is interpreted element-wise.
13. Wait until $(M_l, \texttt{Decrypt}, \Gamma_l, \texttt{Commit}, \alpha_l)$ appears on $\mathcal{F}_{\mathrm{BB}}$ for $l = 1, \ldots, k$. Then hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c)$.
14. Compute $e_j = P_{\mathrm{exp}}(\mathrm{state}_j, c)$ and hand $(\texttt{Write}, \texttt{Reply}, e_j)$ to $\mathcal{F}_{\mathrm{BB}}$.
15. Wait until $(\texttt{Reply}, e_l)$ appears on $\mathcal{F}_{\mathrm{BB}}$ for $l = 1, \ldots, k$. For $l = 1, \ldots, k$ do the following. If $V_{\mathrm{exp}}(\mathsf{n}, \mathsf{v}, v_l, L^2_{k+1}, \Gamma_l, \alpha_l, c, e_l) = 0$ do
    (a) Hand $(\texttt{Write}, \texttt{Recover}, M_l, \mathsf{d}_{l,j}, \mathsf{t}_{l,j})$ to $\mathcal{F}_{\mathrm{BB}}$.
    (b) Wait until $(M_{l'}, \texttt{Recover}, M_l, \mathsf{d}_{l,l'}, \mathsf{t}_{l,l'})$ appears on $\mathcal{F}_{\mathrm{BB}}$ for $l' = 1, \ldots, k$. Then find a subset I of k' indices l' such that $F_{l,l'} = g^{\mathsf{d}_{l,l'}} h^{\mathsf{t}_{l,l'}}$ and Lagrange interpolate $\mathsf{d}_l = \sum_{l' \in I} \mathsf{d}_{l,l'} \prod_{l'' \neq l'} \frac{l''}{l''-l'} \bmod q$.
    (c) Compute $\Gamma_l = \{v_{l,i}\}_{i=1}^{N'} = \{u_{k,i}^{2\mathsf{d}_l}\}_{i=1}^{N}$.
16. Interpret each element in $\{L(\prod_{l=1}^{k} v_{l,i})/2^{k+2}\}_{i=1}^{N'}$ as an integer in $[-2^{\kappa_m+\kappa_r} + 1, 2^{\kappa_m+\kappa_r} - 1]$ (this can be done uniquely, since $\kappa_m + \kappa_r < \kappa - 1$), truncate to κ_m bits, and let L_{out} be the result. Output $(\texttt{Output}, \mathsf{Sort}(L_{\mathrm{out}}))$.

Theorem 1. *The protocol π_{RMN} above securely realizes $\mathcal{F}_{\mathrm{RMN}}$ in the $(\mathcal{F}_{\mathrm{BB}}, \mathcal{F}_{\mathrm{PKG}}, \mathcal{F}_{\mathrm{CF}})$-hybrid model for $\mathcal{M}^*_{k/2}$-adversaries under the DCR- assumption, the strong RSA-assumption, and the DL-assumption.*

References

1. M. Abe and H. Imai. Flaws in some robust optimistic mix-nets. In *Australasian Conference on Information Security and Privacy – ACISP 2003*, volume 2727 of *LNCS*, pages 39–50. Springer Verlag, 2003.
2. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE Computer Society Press, 2001. (Full version at Cryptology ePrint Archive, Report 2000/067, `http://eprint.iacr.org`, October, 2001.).
3. D. Chaum. Untraceable electronic mail, return addresses and digital pseudo-nyms. *Communications of the ACM*, 24(2):84–88, 1981.
4. I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology – Asiacrypt 2002*, volume 2501 of *LNCS*, pages 125–142. Springer Verlag, 2002.
5. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography – PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer Verlag, 2001.
6. I. Damgård and J. B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology – Crypto 2003*, volume 2729 of *LNCS*, pages 247–267. Springer Verlag, 2003.
7. J. Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE Transactions*, 88-A(1):172–188, 2005.
8. J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology – Crypto 2001*, volume 2139 of *LNCS*, pages 368–387. Springer Verlag, 2001.
9. J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *Public Key Cryptography – PKC 2003*, volume 2567 of *LNCS*, pages 145–160. Springer Verlag, 2003. Full version at Cryptology ePrint Archive, Report 2005/246, 2005, `http://eprint.iacr.org/`.
10. A. Lysyanskaya and C. Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In *Advances in Cryptology – Asiacrypt 2001*, volume 2248 of *LNCS*, pages 331–350. Springer Verlag, 2001.
11. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attack. In *22th ACM Symposium on the Theory of Computing (STOC)*, pages 427–437. ACM Press, 1990.
12. A. Neff. A verifiable secret shuffle and its application to e-voting. In *8th ACM Conference on Computer and Communications Security (CCS)*, pages 116–125. ACM Press, 2001.
13. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *LNCS*, pages 223–238. Springer Verlag, 1999.
14. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – Crypto '91*, volume 576 of *LNCS*, pages 129–140. Springer Verlag, 1992.
15. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
16. D. Wikström. A universally composable mix-net. In *1st Theory of Cryptography Conference (TCC)*, volume 2951 of *LNCS*, pages 315–335. Springer Verlag, 2004.
17. D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *Advances in Cryptology – Asiacrypt 2005*, volume 3788 of *LNCS*, pages 273–292. Springer Verlag, 2005.