How To Shuffle in Public

Ben Adida[1⋆] and Douglas Wikström[2]

[1] MIT, Computer Science and Artificial Intelligence Laboratory, ben@mit.edu
[2] ETH Zürich, Department of Computer Science, douglas@inf.ethz.ch

Abstract. We show how to public-key obfuscate two commonly used shuffles: decryption shuffles which permute and decrypt ciphertexts, and re-encryption shuffles which permute and re-encrypt ciphertexts. Given a trusted party that samples and obfuscates a shuffle *before* any ciphertexts are received, this reduces the problem of constructing a mix-net to verifiable joint decryption.
We construct a decryption shuffle from any additively homomorphic cryptosystem and show how it can be public-key obfuscated. This construction does not allow efficient distributed verifiable decryption. Then we show how to public-key obfuscate: a decryption shuffle based on the Boneh-Goh-Nissim (BGN) cryptosystem, and a re-encryption shuffle based on the Paillier cryptosystem. Both constructions allow *efficient* distributed verifiable decryption. In the Paillier case we identify and exploit a previously overlooked "homomorphic" property of the cryptosystem.
Finally, we give a distributed protocol for sampling and obfuscating each of the above shuffles and show how it can be used in a trivial way to construct a universally composable mix-net. Our constructions are practical when the number of senders N is reasonably small, e.g. $N = 350$ in the BGN case and $N = 2000$ in the Paillier case.

1 Introduction

Suppose a set of senders $\mathcal{P}_1, \ldots, \mathcal{P}_N$, each with input m_i, want to compute the sorted list $(m_{\pi(1)}, \ldots, m_{\pi(N)})$ of messages while keeping the permutation π secret. A trusted party can provide this service. First, it collects all messages. Then, it sorts the inputs and outputs the result. A protocol, i.e., a list of machines $\mathcal{M}_1, \ldots, \mathcal{M}_k$, that emulates the service of this trusted party is called a *mix-net*, and the parties $\mathcal{M}_1, \ldots, \mathcal{M}_k$ are referred to as *mix servers*. The notion of a mix-net was introduced by Chaum [10] and the main application of mix-nets is to perform electronic elections.

Program obfuscation is the process of "muddling" a program's instructions to prevent reverse-engineering while preserving proper function. Barak et al. first formalized obfuscation as simulatability from black-box access [2]. Goldwasser and Tauman-Kalai extended this definition to consider auxiliary inputs [18]. Some simple programs have been successfully obfuscated [8, 28]. However, generalized program obfuscation, though it would be fantastically useful in practice, has been proven impossible in even the weakest of settings for both models (by their respective authors). Ostrovsky and Skeith [23] consider a weaker model, public-key obfuscation, where the obfuscated program's output is encrypted. In this model, they achieve the more complex application of private stream searching.

1.1 Our Contributions

We show how to public-key obfuscate the shuffle phase of a mix-net. We show how any homomorphic cryptosystem can provide obfuscated mixing, though the resulting mix-net inefficient. We show how special – and distinct – properties of the Boneh-Goh-Nissim [7] and Paillier [24] cryptosystems enable obfuscated mixing with sufficient efficiency to be practical in some settings.

⋆ The author wishes to acknowledge support from the Caltech/MIT Voting Technology Project and the Knight Foundation.

We formalize our constructions in the public-key obfuscation model of Ostrovsky and Skeith, whose indistinguishability property closely matches the security requirements of a mix-net. We also show how to prove in zero-knowledge the correct obfuscation of a shuffle. Finally, we describe a protocol that allows a set of parties to jointly and robustly generate an obfuscated randomly chosen shuffle. Our mix-nets require considerably more exponentiations, $O(\kappa N^2)$ instead of $O(\kappa N)$ where κ is the security parameter, than private mix-net techniques, yet they remain reasonably practical for precinct-based elections, where voters are anonymized in smaller batches and all correctness proofs can be carried out in advance.

Our constructions can also be presented in the obfuscation model of Barak et al., but we choose the public-key obfuscation model as it immediately provides the indistinguishability property desired from a shuffle functionality.

1.2 Previous Work

Most mix-nets are based on homomorphic cryptosystems and use the re-encryption-permutation paradigm introduced by Park et al. [25] and made universally verifiable by Sako and Kilian [26]. Each mix-server in turn re-encrypts and permutes the ciphertexts. The first efficient zero-knowledge shuffle proofs were given independently by Neff [22] and Furukawa and Sako [17]. Groth [20] generalized Neff's approach to any homomorphic cryptosystem and improved its efficiency. A third, different, approach was given recently by Wikström [30]. The first definition of security of a mix-net was given by Abe and Imai [1] and the first proof of security of a mix-net as a whole was given by Wikström [29, 30]. Wikström and Groth [32] give the first adaptively secure mix-net.

Multi-candidate election schemes where the set of candidates is predetermined have been proposed using homomorphic encryption schemes, initially by Benaloh [6, 5] and subsequently by others to handle multiple races and multiple candidates per race [13, 27, 14, 16, 3, 20]. Homomorphic tallying is similar to obfuscated shuffles in that, on and after election day, only public computation is required for the anonymization process. However, homomorphic tallying cannot recover the individual input plaintexts, which is required by the election laws in some countries and in the case of write-in votes.

Ostrovsky and Skeith define the notion of public-key obfuscation to describe and analyze their work on streaming-data search using homomorphic encryption [23]. In their definition, an obfuscated program is run on plaintext inputs and provides the outputs of the original program in encrypted form. We use a variation of this definition, where the inputs are encrypted and the unobfuscated program may depend on the public key of the cryptosystem.

1.3 Overview of Techniques

The protocols presented in this work use homomorphic multiplication with a permutation matrix, followed by a provable threshold decryption step typical of mix-nets. Roughly, the semantic security of the encryption scheme hides the permutation.

Generic Construction. Consider two semantically-secure cryptosystems, $\mathcal{CS} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ and $\mathcal{CS}' = (\mathcal{G}', \mathcal{E}', \mathcal{D}')$, where $\mathcal{CS}'$ is additively homomorphic and the plaintext space of $\mathcal{CS}'$ can accommodate any ciphertext from $\mathcal{CS}$. Note the interesting properties:

$$\mathcal{E}'_{pk'}(1)^{\mathcal{E}_{pk}(m)} = \mathcal{E}'_{pk'}(\mathcal{E}_{pk}(m)) \ , \quad \mathcal{E}'_{pk'}(0)^{\mathcal{E}_{pk}(m)} = \mathcal{E}'_{pk'}(0) \ , \text{ and}$$
$$\mathcal{E}'_{pk'}(0)\mathcal{E}'_{pk'}(\mathcal{E}_{pk}(m)) = \mathcal{E}'_{pk'}(\mathcal{E}_{pk}(m)) \ .$$

Consider the element-wise encryption of a permutation matrix under $\mathcal{E}'$, and consider inputs to the shuffle as ciphertexts under $\mathcal{E}$. Homomorphic matrix multiplication can be performed using the

properties above for multiplication and addition. The result is a list of *double-encrypted* messages, $\mathcal{E}'_{pk'}(\mathcal{E}_{pk}(m_i))$, that must then be provably decrypted. Unfortunately, a proof of double-decryption is particularly inefficient because revealing any intermediate ciphertext $\mathcal{E}_{pk}(m_i)$ is not an option, as it would immediately leak the permutation.

BGN Construction. The BGN cryptosystem is additively homomorphic and has two encryption algorithms and two decryption algorithms that can be used with the same keys and which provides both additive and multiplicative homomorphisms in the following sense.

$$\mathcal{E}_{pk}(m_1) \otimes \mathcal{E}_{pk}(m_1) = \mathcal{E}'_{pk}(m_1 m_2) \ , \quad \mathcal{E}_{pk}(m_1) \oplus \mathcal{E}_{pk}(m_2) = \mathcal{E}_{pk}(m_1 + m_2) \ , \text{ and}$$
$$\mathcal{E}'_{pk}(m_1) \oplus \mathcal{E}'_{pk}(m_2) = \mathcal{E}'_{pk}(m_1 + m_2) \ .$$

Thus, both the matrix and the inputs can be encrypted using the *same* encryption algorithm $\mathcal{E}$ and public key, and the matrix multiplication uses both homomorphisms. The result is a list of singly encrypted ciphertexts under $\mathcal{E}'$, which lends itself to efficient, provable decryption.

Paillier Construction. The Paillier cryptosystem is additively homomorphic and supports layered encryption, where a ciphertext can be encrypted again using the same public key. The homomorphic properties are preserved in the inner layer; in addition to the generic layered homomorphic properties we have the special relation

$$\mathcal{E}'_{pk}(\mathcal{E}_{pk}(0,r))^{\mathcal{E}_{pk}(m,s)} = \mathcal{E}'_{pk}(\mathcal{E}_{pk}(0,r)\mathcal{E}_{pk}(m,s)) = \mathcal{E}'_{pk}(\mathcal{E}_{pk}(m,r+s)) \ .$$

Thus, we can use $\mathcal{E}'$ encryption for the permutation matrix, and $\mathcal{E}$ encryption for the inputs. When representing the permutation matrix under $\mathcal{E}'$, instead of $\mathcal{E}'_{pk}(1)$ to represent a one we use $\mathcal{E}'_{pk}(\mathcal{E}_{pk}(0,r))$ with a random r. During the matrix multiplication, the "inner" $\mathcal{E}_{pk}(0,r)$ performs re-encryption on the inputs, which allows the decryption process to reveal the intermediate ciphertext without leaking the permutation, making the decryption proof much more efficient.

2 Preliminaries

2.1 Notation

We denote by κ the main security parameter and say that a function $\epsilon(\cdot)$ is negligible if for every constant c there exists a constant κ_0 such that $\epsilon(\kappa) < \kappa^{-c}$ for $\kappa > \kappa_0$. We denote by κ_c and κ_r additional security parameters such that $2^{-\kappa_c}$ and $2^{-\kappa_r}$ are negligible, which determines the bit-size of challenges and random paddings in our protocols. We denote by PT, PPT, and PT*, the set of uniform polynomial time, probabilistic uniform polynomial time, and non-uniform polynomial time Turing machines respectively. In interactive protocols we denote by $\mathcal{P}$ the prover and $\mathcal{V}$ the verifier. We understand a proof of knowledge to mean a complete proof of knowledge with overwhelming soundness and negligible knowledge error. We denote by Σ_N the set of permutations of N elements. And we write $\Lambda^\pi = (\lambda_{ij}^\pi)$ for the corresponding permutation matrix. We denote by M_{pk}, R_{pk}, and C_{pk}, the plaintext space, the randomizer space, and the ciphertext space induced by the public key pk of some cryptosystem.

2.2 Homomorphic Cryptosystems

In the following definition we mean by abelian group a specific representation of an abelian group for which there exists a polynomial time algorithm for computing the binary operator.

Definition 1 (Homomorphic). *A cryptosystem $\mathcal{CS} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ is homomorphic if for every key pair $(pk, sk) \in \mathcal{G}(1^\kappa)$*

1. *The message space M_{pk} is a subset of an abelian group $G(\mathsf{M}_{pk})$ written additively.*
2. *The randomizer space R_{pk} is an abelian group written additively.*
3. *The ciphertext space C_{pk} is a abelian group written multiplicatively.*
4. *For every $m, m' \in \mathsf{M}_{pk}$ and $r, r' \in \mathsf{R}_{pk}$ we have $\mathcal{E}_{pk}(m, r)\mathcal{E}_{pk}(m', r') = \mathcal{E}_{pk}(m + m', r + r')$.*

Furthermore, if $\mathsf{M}_{pk} = G(\mathsf{M}_{pk})$ it is called fully homomorphic, and if $G(\mathsf{M}_{pk}) = \mathbb{Z}_n$ for some integer $n > 0$ it is called additive.

For an additively homomorphic cryptosystem, $\mathcal{RE}_{pk}(c, r) = c\mathcal{E}_{pk}(0, r)$ is called a re-encryption algorithm.

2.3 Functionalities

Definition 2 (Functionality). *A functionality is a family $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa \in \mathbb{N}}$ of sets of circuits such that there exists a polynomial $s(\cdot)$ such that $|F| \leq s(\kappa)$ for every $F \in \mathcal{F}_\kappa$.*

We use a variation of the definition of public-key obfuscation of Ostrovsky and Skeith [23]. Our definition differs in that the functionality takes the public and secret keys as input.

Definition 3 (Public-Key Obfuscator). *An algorithm $\mathcal{O} \in \mathrm{PPT}$ is a public-key obfuscator for a functionality $\mathcal{F}$ with respect to a cryptosystem $\mathcal{CS} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ if there exists a decryption algorithm $\mathcal{D}' \in \mathrm{PT}$ and a polynomial $s(\cdot)$ such that for every $\kappa \in \mathbb{N}$, $F \in \mathcal{F}_\kappa$, $(pk, sk) \in \mathcal{G}(1^\kappa)$, and $x \in \{0,1\}^*$,*

1. Correctness. $\mathcal{D}'_{sk}(\mathcal{O}(1^\kappa, pk, sk, F)(x)) = F(pk, sk, x)$.
2. Polynomial blow-up. $|\mathcal{O}(1^\kappa, pk, sk, F)| \leq s(|F|)$.

Example 1. Suppose $\mathcal{CS}$ is additively homomorphic, $a \in \mathsf{M}_{pk}$, $(pk, sk) \in \mathcal{G}(1^\kappa)$, and define $F_a(pk, sk, x) = ax$, where $x \in \mathsf{M}_{pk}$. An obfuscator for the functionality $\mathcal{F}$ of such circuits can be defined as a circuit with $\mathcal{E}_{pk}(a)$ hardcoded which, on input $x \in \mathsf{M}_{pk}$, outputs $\mathcal{E}_{pk}(a)^x = \mathcal{E}_{pk}(ax)$.

We extend the definition of polynomial indistinguishability of a public-key obfuscator.

Experiment 1 (Indistinguishability, $\mathsf{Exp}^{\mathsf{oind}-b}_{\mathcal{F},\mathcal{CS},\mathcal{O},\mathcal{A}}(\kappa)$).

$$\begin{aligned} (pk, sk) &\leftarrow \mathcal{G}(1^\kappa) \\ (F_0, F_1, state) &\leftarrow \mathcal{A}(\mathsf{choose}, pk), \\ d &\leftarrow \mathcal{A}(\mathcal{O}(1^\kappa, pk, sk, F_b), state) \end{aligned}$$

If $F_0, F_1 \in \mathcal{F}_\kappa$ return d, otherwise 0.

Definition 4 (Indistinguishability). *A public-key obfuscator $\mathcal{O}$ for a functionality $\mathcal{F}$ with respect to a cryptosystem $\mathcal{CS} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ is polynomially indistinguishable if $|\Pr[\mathsf{Exp}^{\mathsf{oind}-0}_{\mathcal{F},\mathcal{CS},\mathcal{O},\mathcal{A}}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{oind}-1}_{\mathcal{F},\mathcal{CS},\mathcal{O},\mathcal{A}}(\kappa) = 1]|$ is negligible.*

Note that the obfuscator in Example 1 is polynomially indistinguishable if $\mathcal{CS}$ is polynomially indistinguishable.

2.4 Shuffles

The most basic form of a shuffle is the decryption shuffle. It simply takes a list of ciphertexts, decrypts them and outputs them in permuted order. In some sense this is equivalent to a mix-net.

Definition 5 (Decryption Shuffle). *A* $\mathcal{CS}$-decryption shuffle*, for a cryptosystem* $\mathcal{CS} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ *is a functionality* $\mathcal{DS}_N = \{\mathcal{DS}_{N(\kappa),\kappa}\}_{\kappa\in\mathbb{N}}$*, where* $N(\kappa)$ *is a polynomially bounded and polynomially computable function, such that for every* $\kappa \in \mathbb{N}$*,* $\mathcal{DS}_{N(\kappa),\kappa} = \{DS_\pi\}_{\pi\in\Sigma_{N(\kappa)}}$*, and for every* $(pk, sk) \in \mathcal{G}(1^\kappa)$*, and* $c_1, \ldots, c_{N(\kappa)} \in \mathsf{C}_{pk}$ *the circuit* DS_π *is defined by*

$$DS_\pi(pk, sk, (c_1, \ldots, c_{N(\kappa)})) = (\mathcal{D}_{sk}(c_{\pi(1)}), \ldots, \mathcal{D}_{sk}(c_{\pi(N(\kappa))})) \ .$$

A common way to implement a mix-net is to use the re-encryption-permutation paradigm of Park et al. [25]. Using this approach the ciphertexts are first re-encrypted and permuted in a joint way and then decrypted. The re-encryption shuffle below captures the joint re-encryption and permutation phase of a mix-net.

Definition 6 (Re-encryption Shuffle). *A* $\mathcal{CS}$-re-encryption shuffle*, for a homomorphic cryptosystem* $\mathcal{CS} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ *is a functionality* $\mathcal{RS}_N = \{\mathcal{RS}_{N(\kappa),\kappa}\}_{\kappa\in\mathbb{N}}$*, where* $N(\kappa)$ *is a polynomially bounded and polynomially computable function, such that for every* $\kappa \in \mathbb{N}$*,* $\mathcal{RS}_{N(\kappa),\kappa} = \{RS_{\pi,r}\}_{\pi\in\Sigma_{N(\kappa)},r\in(\{0,1\}^*)^{N(\kappa)}}$*, and for every* $(pk, sk) \in \mathcal{G}(1^\kappa)$*, and* $c_1, \ldots, c_{N(\kappa)} \in \mathsf{C}_{pk}$ *the circuit* $RS_{\pi,r}$ *is defined by*

$$RS_\pi(pk, sk, (c_1, \ldots, c_{N(\kappa)})) = (\mathcal{RE}_{pk}(c_{\pi(1)}, r_1), \ldots, \mathcal{RE}_{pk}(c_{\pi(N(\kappa))}, r_{N(\kappa)})) \ .$$

3 Constructing and Obfuscating a Generic Decryption Shuffle

We show that, in principle, all that is needed is an additively homomorphic cryptosystem. Consider two semantically-secure cryptosystems, $\mathcal{CS} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ and $\mathcal{CS}' = (\mathcal{G}', \mathcal{E}', \mathcal{D}')$, with $\mathcal{CS}'$ being additively homomorphic with binary operator $\oplus$ on ciphertexts. Suppose that ciphertexts from $\mathcal{CS}$ can be encrypted under $\mathcal{CS}'$ for all $(pk, sk) \in \mathcal{G}(1^\kappa)$ and $(pk', sk') \in \mathcal{G}'(1^\kappa)$, i.e., $\mathsf{C}_{pk} \subseteq \mathsf{M}'_{pk'}$. The following operations are then possible and, more interestingly, indistinguishable thanks to the semantic security of both cryptosystems:

$$\mathcal{E}'_{pk'}(1)^{\mathcal{E}_{pk}(m)} = \mathcal{E}'_{pk'}(\mathcal{E}_{pk}(m)) \quad \text{and} \quad \mathcal{E}'_{pk'}(0)^{\mathcal{E}_{pk}(m)} = \mathcal{E}'_{pk'}(0) \ .$$

3.1 The Obfuscator

Consider a permutation matrix $\Lambda^\pi = (\lambda^\pi_{ij})$ corresponding to a permutation π. Consider its element-wise encryption under $\mathcal{CS}'$ with public key pk' and a corresponding matrix of random factors $(r_{ij}) \in \mathsf{R}'^{N^2}_{pk'}$, i.e., $C^\pi = (\mathcal{E}'_{pk'}(\lambda^\pi_{ij}, r_{ij}))$. Then given $d = (d_1, d_2, \ldots, d_N) \in \mathsf{C}^N_{pk}$ it is possible to perform homomorphic matrix multiplication as

$$d \star C^\pi = \left(\bigoplus_{i=1}^{N} (c^\pi_{ij})^{d_i} \right) \quad \text{such that} \quad \mathcal{D}_{sk}(\mathcal{D}'_{sk'}(d \star C^\pi)) = (m_{\pi(1)}, m_{\pi(2)}, \ldots, m_{\pi(N)}) \ .$$

Definition 7 (Obfuscator). *The obfuscator* $\mathcal{O}$ *for the decryption shuffle* $\mathcal{DS}_N$ *takes input* $(1^\kappa, (pk, pk'), (sk, sk'), DS_\pi)$*, where* $(pk, sk) \in \mathcal{G}(1^\kappa)$*,* $(pk', sk') \in \mathcal{G}'(1^\kappa)$ *and* $DS_\pi \in \mathcal{DS}_{N(\kappa),\kappa}$*, computes* $C^\pi = \mathcal{E}'_{pk'}(\Lambda^\pi)$*, and outputs a circuit that hardcodes* C^π*, and on input* $d = (d_1, \ldots, d_{N(\kappa)})$ *computes* $d' = d \star C^\pi$ *as outlined above and outputs* d'*.*

Technically, this is a decryption shuffle of a new cryptosystem $\mathcal{CS}'' = (\mathcal{G}'', \mathcal{E}, \mathcal{D})$, where $\mathcal{CS}''$ executes the original key generators and outputs $((pk, pk'), (sk, sk'))$ and the original algorithms $\mathcal{E}$ and $\mathcal{D}$ simply ignore (pk', sk'). We give a reduction without any loss in security for the following straight-forward proposition. We also note that $\mathcal{O}$ does *not* use (sk, sk'): obfuscation only requires the public key.

Proposition 1. *If $\mathcal{CS}'$ is polynomially indistinguishable then $\mathcal{O}$ is polynomially indistinguishable.*

The construction can be generalized to the case where the plaintext space of $\mathcal{CS}'$ does not contain the ciphertext space of $\mathcal{CS}$. Each inner ciphertext d_i is split into pieces $(d_{i1}, \ldots, d_{it})$ each fitting in the plaintext space of $\mathcal{CS}'$ and then apply each list $(d_{1,l}, \ldots, d_{N,l})$ to the encrypted permutation matrix as before. This gives lists $(d'_{1,l}, \ldots, d'_{N,l})$ from which the output list $((d'_{1,l})_l, \ldots, (d'_{N,l})_l)$ is constructed.

3.2 Limitations of the Generic Construction

The matrix C^π requires a proof that it is the encryption of a proper permutation matrix. This can be accomplished using more or less general techniques depending on the cryptosystem, but this is prohibitively expensive in general.

Even if we prove that C^π is correctly formed, the post-shuffle verifiable decryption of $\mathcal{E}'_{pk'}(\mathcal{E}_{pk}(m_i))$ to m_i is prohibitively expensive: revealing the inner ciphertext is out of the question, as it would leak the shuffle permutation, which again leaves us only with generic proof techniques. Instead, we turn to more efficient constructions.

4 Obfuscating a Boneh-Goh-Nissim Decryption Shuffle

We show how to obfuscate a decryption shuffle for the Boneh-Goh-Nissim (BGN) cryptosystem [7] by exploiting both its additive homomorphism and its one-time multiplicative homomorphism.

4.1 The BGN Cryptosystem

We denote the BGN cryptosystem by $\mathcal{CS}^{\mathsf{bgn}} = (\mathcal{G}^{\mathsf{bgn}}, \mathcal{E}^{\mathsf{bgn}}, \mathcal{D}^{\mathsf{bgn}})$. It operates in two groups $\mathbb{G}_1$ and $\mathbb{G}_2$, both of order $n = q_1 q_2$, where q_1 and q_2 are distinct prime integers. We use multiplicative notation in both $\mathbb{G}_1$ and $\mathbb{G}_2$, and denote by g a generator in $\mathbb{G}_1$. The groups $\mathbb{G}_1$ and $\mathbb{G}_2$ exhibit a polynomial-time computable bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ such that $G = e(g, g)$ generates $\mathbb{G}_2$. Bilinearity implies that $\forall u, v \in \mathbb{G}_1$ and $\forall a, b \in \mathbb{Z}, e(u^a, v^b) = e(u, v)^{ab}$. We refer the reader to [7] for details on how such groups can be generated and on the cryptosystem's properties, which we briefly summarize here.

Key generation. On input 1^κ, $\mathcal{G}^{\mathsf{bgn}}$ generates parameters $(q_1, q_2, \mathbb{G}_1, g, \mathbb{G}_2, e(\cdot, \cdot))$ as above such that $n = q_1 q_2$ is a κ-bit integer. It chooses $u \in \mathbb{G}_1$ randomly, defines $h = u^{q_2}$, and outputs a public key $pk = (n, \mathbb{G}_1, \mathbb{G}_2, e(\cdot, \cdot), g, h)$ and secret key $sk = q_1$.
Encryption in $\mathbb{G}_1$. On input pk and m, $\mathcal{E}^{\mathsf{bgn}}$ selects $r \in \mathbb{Z}_n$ randomly and outputs $c = g^m h^r$.
Decryption in $\mathbb{G}_1$. On input $sk = q_1$ and $c \in \mathbb{G}_1$, $\mathcal{D}^{\mathsf{bgn}}$ outputs $m' = \log_{g^{q_1}}(c^{q_1})$.

Because decryption computes a discrete logarithm, the plaintext space must be restricted considerably. Corresponding algorithms $\mathcal{E}'^{\mathsf{bgn}}$ and $\mathcal{D}'^{\mathsf{bgn}}$ perform encryption and decryption in $\mathbb{G}_2$ using the generators $G = e(g, g)$ and $H = e(g, h)$. The BGN cryptosystem is semantically secure under the *Subgroup Decision Assumption*, which states that no $\mathcal{A} \in \mathrm{PT}^*$ can distinguish between a uniform distribution on $\mathbb{G}_1$ and a uniform distribution on the unique order q_1 subgroup in $\mathbb{G}_1$.

Homomorphisms. The BGN cryptosystem is additively homomorphic. We need this property, but we also exploit its one-time multiplicative homomorphism implemented by the bilinear map (we denote $\alpha = \log_g u$ in the BGN key generation):

$$e(\mathcal{E}_{pk}^{\mathsf{bgn}}(m_0, r_0), \mathcal{E}_{pk}^{\mathsf{bgn}}(m_1, r_1)) = \mathcal{E}_{pk}^{'\mathsf{bgn}}(m_0 m_1, m_0 r_1 + m_1 r_0 + \alpha q_2 r_0 r_1)$$

The result is a ciphertext in $\mathbb{G}_2$ which cannot be efficiently converted back to an equivalent ciphertext in $\mathbb{G}_1$. Thus, the multiplicative homomorphism can be evaluated only once, after which only homomorphic additions are possible. For clarity, we write $c_1 \oplus c_2 \stackrel{\text{def}}{=} c_1 c_2$ for ciphertexts in $\mathbb{G}_1$ or $\mathbb{G}_2$ and $c_1 \otimes c_2 \stackrel{\text{def}}{=} e(c_1, c_2)$ for ciphertexts in $\mathbb{G}_1$.

4.2 The Obfuscator

Our obfuscator is based on the observation that matrix multiplication only requires an arithmetic circuit with multiplication depth 1. Thus, the BGN cryptosystem can be used for homomorphic matrix multiplication. Consider a $N_1 \times N_2$-matrix $C = (c_{ij}) = (\mathcal{E}_{pk}^{\mathsf{bgn}}(a_{ij}))$ and a $N_2 \times N_3$-matrix $C' = (d_{jk}) = (\mathcal{E}_{pk}^{\mathsf{bgn}}(b_{jk}))$, and let $A = (a_{ij})$ and $B = (b_{jk})$. We define homomorphic matrix multiplication by

$$C \star C' \stackrel{\text{def}}{=} \left(\bigoplus_{j=1}^{N_2} c_{ij} \otimes d_{jk} \right) \quad \text{and conclude that} \quad \mathcal{D}_{sk}^{'\mathsf{bgn}}(C \star C') = \left(\sum_{j=1}^{N_2} a_{ij} b_{jk} \right) = AB \ .$$

Definition 8 (Obfuscator). *The obfuscator $\mathcal{O}^{\mathsf{bgn}}$ for the decryption shuffle $\mathcal{DS}_N^{\mathsf{bgn}}$ takes input $(1^\kappa, pk, sk, DS_\pi^{\mathsf{bgn}})$, where $(pk, sk) \in \mathcal{G}^{\mathsf{bgn}}(1^\kappa)$ and $DS_\pi^{\mathsf{bgn}} \in \mathcal{DS}_{N(\kappa),\kappa}^{\mathsf{bgn}}$, computes $C^\pi = \mathcal{E}_{pk}^{\mathsf{bgn}}(\Lambda^\pi)$, and outputs a circuit with C^π hard-coded such that, on input $d = (d_1, \ldots, d_{N(\kappa)})$, it outputs $d' = d \star C^\pi$.*

Note that $\mathcal{O}^{\mathsf{bgn}}$ does not use sk. We have the following corollary from Proposition 3.

Corollary 1. *The obfuscator $\mathcal{O}^{\mathsf{bgn}}$ for $\mathcal{DS}_N^{\mathsf{bgn}}$ is polynomially indistinguishable if the BGN cryptosystem is polynomially indistinguishable.*

5 Obfuscating a Paillier Re-encryption Shuffle

We show how to obfuscate a re-encryption shuffle for the Paillier cryptosystem [24] by exploiting its additive homomorphism and its generalization introduced by Damgård et al. [15]. We expose a previously unnoticed homomorphic property of this generalized Paillier construction.

5.1 The Paillier Cryptosystem

We denote the Paillier cryptosystem $\mathcal{CS}^{\mathsf{pai}} = (\mathcal{G}^{\mathsf{pai}}, \mathcal{E}^{\mathsf{pai}}, \mathcal{D}^{\mathsf{pai}})$. It is defined as follows:

Key Generation. On input 1^κ, $\mathcal{G}^{\mathsf{pai}}$ chooses safe primes $p = 2p' + 1$ and $q = 2q' + 1$ randomly such that $n = pq$ is a κ-bit integer, defines global parameter $v = n + 1$ and outputs a public key $pk = n$ and a secret key $sk = p$.

Encryption. On input pk and $m \in \mathbb{Z}_n$, $\mathcal{E}^{\mathsf{pai}}$ selects $r \in \mathbb{Z}_n^*$ randomly and outputs $v^m r^n \bmod n^2$.

Decryption. On input sk and c, given e such that $e = 1 \bmod n$ and $e = 0 \bmod \phi(n)$, $\mathcal{D}^{\mathsf{pai}}$ outputs $(c^e - 1)/n$.

The Paillier cryptosystem is polynomially indistinguishable under the *Decision Composite Residuosity Assumption*, which states that no $\mathcal{A} \in \mathrm{PT}^*$ can distinguish the uniform distribution on $\mathbb{Z}_{n^2}^*$ from the uniform distribution on the subgroup of nth residues in $\mathbb{Z}_{n^2}^*$.

Generalized Paillier. Damgård et al. [15] generalize this scheme, replacing computations modulo n^2 with computations modulo n^{s+1} and plaintext space $\mathbb{Z}_n$ with $\mathbb{Z}_{n^s}$. Damgård et al. prove that the security of the generalized scheme follows from the security of the original scheme for $s > 0$ polynomial in the security parameter, though we only exploit the cases $s = 1, 2$. We write $\mathcal{E}^{\mathsf{pai}}_{n^{s+1}}(m) = v^m r^{n^s} \bmod n^{s+1}$ for generalized encryption to make explicit the value of s used in a particular encryption. Similarly we write $\mathcal{D}^{\mathsf{pai}}_{p,s+1}(c)$ for the decryption algorithm (see [15] for details) and we use $\mathsf{M}_{n^{s+1}}$ and $\mathsf{C}_{n^{s+1}}$ to denote the corresponding message and ciphertext spaces.

Alternative Encryption. There are natural and well known alternative encryption algorithms. It is easy to see that one can pick the random element $r \in \mathbb{Z}^*_{n^s}$ instead of in $\mathbb{Z}^*_n$. If h_{s+1} is a generator of the group of n^sth residues, then we may define encryption of a message $m \in \mathbb{Z}_{n^s}$ as $v^m h^r_{s+1} \bmod n^s$ where r is chosen randomly in $[0, n2^{\kappa_r}]$.

Homomorphisms. The Paillier cryptosystem is additively homomorphic. Furthermore, the recursive structure of the Paillier cryptosystem allows a ciphertext $\mathcal{E}^{\mathsf{pai}}_{n^2}(m) \in \mathsf{C}_{n^2} = \mathbb{Z}^*_{n^2}$ to be viewed as a plaintext in the group $\mathsf{M}_{n^3} = \mathbb{Z}_{n^2}$ that can be encrypted using a generalized version of the cryptosystem, i.e., we can compute $\mathcal{E}^{\mathsf{pai}}_{n^3}\big(\mathcal{E}^{\mathsf{pai}}_{n^2}(m)\big)$. Interestingly, *the nested cryptosystems preserve the group structures over which they are defined*. In other words we have

$$\mathcal{E}^{\mathsf{pai}}_{n^3}(\mathcal{E}^{\mathsf{pai}}_{n^2}(0,r))^{\mathcal{E}^{\mathsf{pai}}_{n^2}(m,s)} = \mathcal{E}^{\mathsf{pai}}_{n^3}(\mathcal{E}^{\mathsf{pai}}_{n^2}(0,r)\mathcal{E}^{\mathsf{pai}}_{n^2}(m,s)) = \mathcal{E}^{\mathsf{pai}}_{n^3}(\mathcal{E}^{\mathsf{pai}}_{n^2}(m,r+s)) \ .$$

Note how this homomorphic operation is similar to the generic additive operation from Section 3, except the inner "1" has been replaced with an encryption of 0. As a result, though the output is also a double-encrypted m_i, a re-encryption has occurred on the inner ciphertext.

5.2 The Obfuscator

We use the additive homomorphism and the special homomorphic property exhibited above to define a form of homomorphic matrix multiplication of matrices of ciphertexts.

Given an N-permutation matrix $\Lambda^\pi = (\lambda^\pi_{ij})$ and randomness $r, s \in (\mathbb{Z}^*_n)^{N\times N}$, define

$$C^\pi = (c^\pi_{ij}) = \left(\mathcal{E}^{\mathsf{pai}}_{n^3}\big(\lambda^\pi_{ij}\mathcal{E}^{\mathsf{pai}}_{n^2}(0, r_{ij}), s_{ij}\big)\right) \ .$$

We define a kind of matrix multiplication of $d = (d_1, \ldots, d_N) \in \mathsf{C}_{n^2}$ and C^π as above by

$$d \star C^\pi = \left(\prod_{i=1}^{N} (c^\pi_{ij})^{d_i}\right) \text{ and conclude that } \mathcal{D}^{\mathsf{pai}}_{p,2}(\mathcal{D}^{\mathsf{pai}}_{p,3}(d \star C^\pi)) = (m_{\pi(1)}, \ldots, m_{\pi(N)}) \ .$$

In other words, we can do homomorphic matrix multiplication with a permutation matrix using layered Paillier, but we stress that the above matrix multiplication does *not* work for all matrices. We are now ready to define the obfuscator for the Paillier-based shuffle.

Definition 9 (Obfuscator). *The obfuscator $\mathcal{O}^{\mathsf{pai}}$ for the re-encryption shuffle $\mathcal{RS}^{\mathsf{pai}}_N$ takes input a tuple $(1^\kappa, n, sk, RS^{\mathsf{pai}})$, where $(n, p) \in \mathcal{G}^{\mathsf{pai}}(1^\kappa)$ and $RS^{\mathsf{pai}} \in \mathcal{RS}^{\mathsf{pai}}_{N(\kappa),\kappa}$, computes $C^\pi = (\mathcal{E}^{\mathsf{pai}}_{n^3}(\lambda^\pi_{ij}\mathcal{E}^{\mathsf{pai}}_{n^2}(0, r_{ij}), s_{ij}))$, and outputs a circuit with hardcoded C^π that, on input $d = (d_1, \ldots, d_{N(\kappa)})$, outputs $d' = d \star C^\pi$.*

Note, again, that $\mathcal{O}^{\mathsf{pai}}$ does not use sk.

Proposition 2. *The obfuscator $\mathcal{O}^{\mathsf{pai}}$ for $\mathcal{RS}^{\mathsf{pai}}_N$ is polynomially indistinguishable if the Paillier cryptosystem is polynomially indistinguishable.*

6 Proving Correctness of Obfuscation

We show how to prove the correctness of a BGN or Paillier obfuscation. We assume, for now, that a single party generates the encrypted matrix, though the techniques described here are immediately applicable to the distributed generation and proofs in Section 7. For either cryptosystem, we start with a trivially encrypted identity matrix, and we let the prover demonstrate that he correctly shuffled the columns of this matrix.

Definition 10. *Denote by $\mathcal{R}_{mrp}$ the relation consisting of pairs $((1^\kappa, pk, C, C'), r)$ such that $C \in \mathsf{C}_{pk}^{N\times N}$, $C' = (\mathcal{RE}_{pk}(c_{i,\pi(j)}, r_{ij}))$, $r \in \mathsf{R}_{pk}^{N\times N}$, and $\pi \in \Sigma_N$.*

In the BGN case, the starting identity matrix can be simply $C = \mathcal{E}_{pk}(\Lambda^{\mathrm{id}}, 0^*)$. Recall that, where the BGN matrix contains encryptions of 1, the Paillier matrix contains outer encryptions of *different* inner encryptions of zero, which need to remain secret.

Thus, in the Paillier case, we begin by generating and proving correct a list of N double-encryptions of zero. We construct a proof of double-discrete log with 1/2-soundness that must be repeated a number of times. This repetition remains "efficient enough" because we only need to perform a linear number of sets of repeated proofs. We then use these N double-encrypted zeros as the diagonal of our identity matrix, completing it with trivial outer encryptions of zero.

In both cases, we then take this identity matrix, shuffle and re-encrypt its columns, and provide a zero-knowledge proof of knowledge of the permutation and re-encryption factors. A verifier is then certain that the resulting matrix is a permutation matrix.

6.1 Proving a Shuffle of the Columns of a Ciphertext Matrix

Consider the simpler and extensively studied problem of proving that a list of ciphertexts have been correctly re-encrypted and permuted, sometimes called a "proof of shuffle".

Definition 11. *Denote by $\mathcal{R}_{rp}$ the relation consisting of pairs $((1^\kappa, pk, d, d'), r)$ such that $d = (d_j) \in \mathsf{C}_{pk}^N$ and $d' = \mathcal{RE}_{pk}((d_{\pi(j)}), r)$ for some $r \in \mathsf{R}_{pk}^N$ and $\pi \in \Sigma_N$.*

There are several known efficient methods [22, 17, 20, 30] for constructing a protocol for this relation. Although these protocols differ slightly in their properties, they all essentially give "honest verifier zero-knowledge proofs of knowledge." As our protocol can be adapted to the concrete details of these techniques, we assume, for clarity, that there exists an honest verifier zero-knowledge proof of knowledge π_{rp} for the above relation. These protocols can be extended to prove a shuffle of lists of ciphertexts (which is what we need), but a detailed proof of this fact have not appeared. We present a simple batch proof (see [4]) of a shuffle to allow us to argue more concretely about the complexity of our scheme.

Protocol 1 (Matrix Re-encryption-Permutation).
Common Input. A public key pk and $C, C' \in \mathsf{C}_{pk}^{N\times N}$
Private Input. $\pi \in \Sigma_N$ and $r \in \mathsf{R}_{pk}^{N\times N}$ such that $C' = \mathcal{RE}_{pk}((c_{i,\pi(j)}), r)$.

1. $\mathcal{V}$ chooses $u \in [0, 2^{\kappa_c} - 1]^N$ randomly and hands it to $\mathcal{P}$.
2. They both compute $d = (\bigoplus_{i=1}^N c_{ij}^{u_i})$ and $d' = (\bigoplus_{i=1}^N (c'_{ij})^{u_i})$.
3. They run π_{rp} on common input (pk, d, d') and private input $\pi, r' = (\sum_{i=1}^N r_{ij} u_i)$.

Proposition 3. *Protocol 1 is public-coin and honest verifier zero-knowledge. For inputs with $C = \mathcal{E}_{pk}(\Lambda^\pi)$ for $\pi \in \Sigma_N$ the error probability is negligible and there exists a knowledge extractor.*

Remark 1. When the plaintexts are known, and this is the case when C is an encryption of the identity matrix, slightly more efficient techniques can be used. This is sometimes called a "shuffle of known plaintexts" (see [22, 20, 30]).

6.2 Proving Double Re-encryption

The following relation captures the problem of proving correctness of a double-re-encryption.

Definition 12. *Denote by* $\mathcal{R}_{dr}^{\mathsf{pai}}$ *the relation consisting of pairs* $((1^\kappa, n, c, c'), (r, s))$*, such that* $c' = c^{h_1^r \bmod n^2} h_2^s \bmod n^3$ *with* $r, s \in [0, N2^{\kappa_r}]$.

Protocol 2 (Double Re-encryption).
Common Input. A modulus n and $c, c' \in \mathsf{C}_{n^3}$
Private Input. $r, s \in [0, n2^{\kappa_r}]$ such that $c' = c^{h_2^r \bmod n^2} h_3^s \bmod n^3$.

1. $\mathcal{P}$ chooses $r' \in [0, n2^{2\kappa_r}]$ and $s' \in [0, n^3 2^{2\kappa_r}]$ randomly, computes $\alpha = c^{h_2^{r'} \bmod n^2} h_3^{s'} \bmod n^3$, and hands α to $\mathcal{V}$.
2. $\mathcal{V}$ chooses $b \in \{0, 1\}$ randomly and hands b to $\mathcal{P}$.
3. $\mathcal{P}$ defines $(e, f) = (r' - br, s' - b(h_2^e \bmod n^2)s)$. Then it hands (e, f) to $\mathcal{V}$.
4. $\mathcal{V}$ checks that $\alpha = ((c')^b c^{1-b})^{h_2^e \bmod n^2} h_3^f \bmod n^3$.

The protocol is iterated in parallel κ_c times to make the error probability negligible. For proving a lists of ciphertexts, we use independent copies of the protocol for each element.

Proposition 4. *Protocol 2 is a public-coin honest verifier zero-knowledge proof of knowledge for* $\mathcal{R}_{dr}^{\mathsf{pai}}$.

7 Distributed Generation and Obfuscation of a Shuffle

We now explain how to sample and obfuscate the BGN and Paillier shuffles in a distributed way. In order to focus the discussion on our contribution, we study the properties of our protocol in a hybrid model as defined in the universally composable framework of Canetti [9]. The hybrid world contains an ideal authenticated bulletin board $\mathcal{F}_{\mathrm{BB}}$, an ideal coin-flipping functionality $\mathcal{F}_{\mathrm{CF}}$, an ideal zero-knowledge proof of knowledge of a plaintext $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}_{kp}}$, and an ideal key generator $\mathcal{F}_{\mathrm{KG}}$ that also allows decryption of a list of ciphertexts if requested by a majority of the mix-servers.

The only natural use of our construction is as a subprotocol, as it does not realize any natural, universally composable functionality. We could, of course, use game-based definitions instead, but this would not capture all the relevant security properties when used in a complete mix-net setting. Instead, we show how our protocol can be used in a trivial way to securely realize an ideal mix-net functionality $\mathcal{F}_{\mathrm{MN}}$. To simplify the exposition, we say that a public-coin protocol is used "in a distributed way" when the challenge is taken from the ideal coin-flipping functionality.

Our protocol varies slightly depending on the cryptosystem used: the BGN construction is a *decryption* shuffle, while the Paillier construction is a *re-encryption* shuffle. We only indicate which cryptosystem is used when necessary and keep our notation generic otherwise, e.g. we write $\mathcal{CS}$ instead of $\mathcal{CS}^{\mathsf{bgn}}$ or $\mathcal{CS}^{\mathsf{pai}}$. Recall that, in the Paillier case, the obfuscated shuffle can be viewed as an outer-layer encrypted permutation matrix where the ones are replaced by inner-layer ciphertexts of zero. Thus, at the start of the Paillier version, we need an additional subprotocol.

Protocol 3 (Generation of Double-Paillier Encrypted Zeros).
Common Input. A Paillier public key n and integer N.
Mix-server $\mathcal{M}_j$ proceeds as follows.

1. Define $d_0 = (\mathcal{E}_{n^3}^{\mathsf{pai}}(\mathcal{E}_{n^2}^{\mathsf{pai}}(0, 0^*)), \ldots, \mathcal{E}_{n^3}^{\mathsf{pai}}(\mathcal{E}_{n^2}^{\mathsf{pai}}(0, 0^*)))$ of length N.
2. For $l = 1, \ldots, k$ do:

(a) If $l \neq j$, then wait until $(\mathcal{M}_l, \texttt{List}, d_l)$, $d_l \in \mathsf{C}_{n^3}$, appears on $\mathcal{F}_{\text{BB}}$. Execute the verifier of Protocol 2 in a distributed way. If it fails, then set $d_l = d_{l-1}$.

(b) If $l = j$, then do:

i. Choose $r_j, s_j \in [0, n2^{\kappa_r}]^N$ randomly, compute $d_j = (d_{j-1,i}^{h_2^{r_{j,i}} \bmod n^2} h_3^{s_{j,i}} \bmod n^3)$, and publish $(\texttt{List}, d_j)$ on $\mathcal{F}_{\text{BB}}$.

ii. Prove using Protocol 2 in a distributed way knowledge of $r_j, s_j \in [0, n2^{\kappa_r}]^N$ such that $(n, d_{j-1}, d_j) \in \mathcal{R}_{dr}^{\mathsf{pai}}$.

3. Output d_k.

Protocol 4 (Generation and Obfuscation of a Shuffle).
Common Input. A public key pk and integer N.
Mix-server $\mathcal{M}_j$ proceeds as follows.

1. In BGN Case. Define $C_0 = \mathcal{E}_{pk}(\Lambda^{\text{id}}, 0^*)$, with Λ^{id} an $N \times N$ identity matrix .
   In Paillier Case. Execute Protocol 3 and denote its output by $(c_{1,1}, \ldots, c_{N,N})$. Then form a matrix $C_0 = (c_{it})$ by setting the remaining elements $c_{i,t} = \mathcal{E}_{n^3}^{\mathsf{pai}}(0, 0^*)$ for $i \neq t$.
2. For $l = 1, \ldots, k$ do:
   (a) If $l \neq j$, then wait until $(\mathcal{M}_l, \texttt{Matrix}, C_l)$, $C_l = (c_{l,i,t}) \in \mathsf{C}_{pk}^{N \times N}$, appears on $\mathcal{F}_{\text{BB}}$. Execute the verifier of Protocol 1 in a distributed way. If it fails, then set $C_l = C_{l-1}$.
   (b) If $l = j$, then do:
      i. Choose $r_j \in \mathsf{R}_{pk}^{N \times N}$ randomly, choose $\pi_j \in \Sigma_N$ randomly, compute $C_j = \mathcal{RE}_{pk}((c_{j-1,i,\pi_j(t)}), r_j)$, and publish $(\texttt{Matrix}, C_j)$ on $\mathcal{F}_{\text{BB}}$.
      ii. Prove, using Protocol 1 in a distributed way, knowledge of $r_j \in \mathsf{R}_{pk}^{N \times N}$ such that $((pk, C_{j-1}, C_j), r_j) \in \mathcal{R}_{mrp}$.
3. Output C_k.

8 Trivial Mix-Net From Obfuscated Shuffle

We now give the promised trivial realization of an ideal mix-net. The ideal mix-net functionality $\mathcal{F}_{\text{MN}}$ we consider is essentially the same as that used in [29, 30, 32]. It simply accepts inputs and waits until a majority of the mix-servers requests that the mixing process starts. At this point it sorts the inputs and outputs the result.

Functionality 1 (Mix-Net). The ideal functionality for a mix-net, $\mathcal{F}_{\text{MN}}$, running with mix-servers $\mathcal{M}_1, \ldots, \mathcal{M}_k$, senders $\mathcal{P}_1, \ldots, \mathcal{P}_N$, and ideal adversary $\mathcal{S}$ proceeds as follows

1. Initialize a list $L = \emptyset$, a database D, a counter $c = 0$, and set $J_S = \emptyset$ and $J_M = \emptyset$.
2. Repeatedly wait for inputs
   - Upon receipt of $(\mathcal{P}_i, \texttt{Send}, m_i)$ with $m_i \in \{0,1\}^{\kappa_m}$ and $i \notin J_S$ from $\mathcal{C}_{\mathcal{I}}$, store this tuple in D under the index c, set $c \leftarrow c + 1$, and hand $(\mathcal{S}, \mathcal{P}_i, \texttt{Input}, c)$ to $\mathcal{C}_{\mathcal{I}}$.
   - Upon receipt of $(\mathcal{M}_j, \texttt{Run})$ from $\mathcal{C}_{\mathcal{I}}$, store $(\mathcal{M}_j, \texttt{Run})$ in D under the index c, set $c \leftarrow c+1$, and hand $(\mathcal{S}, \mathcal{M}_j, \texttt{Input}, c)$ to $\mathcal{C}_{\mathcal{I}}$.
   - Upon receipt of $(\mathcal{S}, \texttt{AcceptInput}, c)$ such that something is stored under the index c in D do
     (a) If $(\mathcal{P}_i, \texttt{Send}, m_i)$ with $i \notin J_S$ is stored under c, then append m_i to L, set $J_S \leftarrow J_S \cup \{i\}$, and hand $(\mathcal{S}, \mathcal{P}_i, \texttt{Send})$ to $\mathcal{C}_{\mathcal{I}}$.
     (b) If $(\mathcal{M}_j, \texttt{Run})$ is stored under c, then set $J_M \leftarrow J_M \cup \{j\}$. If $|J_M| > k/2$, then sort the list L lexicographically to form a list L', hand $((\mathcal{S}, \mathcal{M}_j, \texttt{Output}, L'), \{(\mathcal{M}_l, \texttt{Output}, L')\}_{l=1}^k)$ to $\mathcal{C}_{\mathcal{I}}$ and ignore further messages. Otherwise, hand $\mathcal{C}_{\mathcal{I}}$ the list $(\mathcal{S}, \mathcal{M}_j, \texttt{Run})$.

The functionalities of the hybrid world are fairly natural. The bulletin board $\mathcal{F}_{\text{BB}}$ is authenticated; everybody can write to it, and nobody can delete anything. It also stores the order in which messages appear. The coin-flipping functionality $\mathcal{F}_{\text{CF}}$ waits for a coin-request and then simply outputs the requested number of random coins. The zero-knowledge proof of knowledge of a plaintext $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{kp}}$ allows a sender to submit a public key pk, a ciphertext c, a message m, and a random string $r \in \{0,1\}^*$, and simply tells the mix-servers if $m \in \{0,1\}^{\kappa_m}$ and $c = \mathcal{E}_{pk}(m,r)$ or not. The key generation functionality generates a key pair $(pk, sk) = \mathcal{G}(1^\kappa)$ and outputs the public key pk. Then if it receives more than $k/2$ requests to decrypt a certain list of ciphertexts, it decrypts it using sk and outputs the result. The definitions of these functionalities are given in Appendix B.

Protocol 5 (Trivial Mix-Net).
Sender $\mathcal{P}_i$ proceeds as follows.

1. Wait for $(\texttt{PublicKey}, pk)$ from $\mathcal{F}_{\text{KG}}$.
2. Wait for an input $m \in \{0,1\}^{\kappa_m}$, choose $r \in \{0,1\}^*$ randomly, and compute $c = \mathcal{E}_{pk}(m,r)$. Then publish $(\texttt{Send}, c)$ on $\mathcal{F}_{\text{BB}}$ and hand $(\texttt{Prover}, (pk, c), (m, r))$ to $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{kp}}$.

Mix-server $\mathcal{M}_j$ proceeds as follows.
Offline Phase

1. Wait for $(\texttt{PublicKey}, pk)$ from $\mathcal{F}_{\text{KG}}$.
2. Execute Protocol 4 (with either BGN or Paillier as appropriate) and denote the output obfuscated (decryption/re-encryption) shuffle by C^π.

Online Phase

3. Initialize $J_{\mathcal{M}} = \emptyset$, $J_{\mathcal{P}} = \emptyset$, and repeatedly wait for new inputs or a new message on $\mathcal{F}_{\text{BB}}$.
   - On input $(\texttt{Run})$ hand $(\texttt{Write}, \texttt{Run})$ to $\mathcal{F}_{\text{BB}}$.
   - If $(\mathcal{M}_j, \texttt{Run})$ appears on $\mathcal{F}_{\text{BB}}$, then set $J_{\mathcal{M}} \leftarrow J_{\mathcal{M}} \cup \{j\}$. If $|J_{\mathcal{M}}| > k/2$, go to Step 4.
   - If $(\mathcal{P}_\gamma, \texttt{Send}, c_\gamma)$ appears on $\mathcal{F}_{\text{BB}}$ for $\gamma \notin J_{\mathcal{P}}$ then do:
     (a) Set $J_{\mathcal{P}} \leftarrow J_{\mathcal{P}} \cup \{\gamma\}$.
     (b) Hand $(\texttt{Question}, \mathcal{P}_\gamma, (pk, c_\gamma))$ to $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{kp}}$ and wait for a reply $(\texttt{Verifier}, \mathcal{P}_\gamma, (pk, c_\gamma), b_\gamma)$ from $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{kp}}$.
4. Let $J'_{\mathcal{P}} \subset J_{\mathcal{P}}$ be the set of γ such that $b_\gamma = 1$. Form a list of trivial encryptions $d_{pad} = (\mathcal{E}_{pk}(0,0^*), \ldots, \mathcal{E}_{pk}(0,0^*))$ of length $N - |J'_{\mathcal{P}}|$. Then form the concatenated list $d = (c_\gamma)_{\gamma \in J_{\mathcal{P}}} \| d_{pad}$, and compute $d' = d \star C^\pi$.
5. In BGN Case. Hand $(\texttt{Decrypt}, d')$ to $\mathcal{F}_{\text{KG}}$ and wait until it returns $(\texttt{Decrypted}, m)$. Form a new list m' by sorting m lexicographically and removing $N - |J'_{\mathcal{P}}|$ copies of 0. Then output $(\texttt{Output}, m')$.
   In Paillier Case. Hand $(\texttt{Decrypt}, d')$ to $\mathcal{F}_{\text{KG}}$ and wait until it returns $(\texttt{Decrypted}, d'')$. Hand $(\texttt{Decrypt}, d'')$ to $\mathcal{F}_{\text{KG}}$ and wait until it returns $(\texttt{Decrypted}, m)$. Form a new list m' by sorting m lexicographically and removing $N - |J'_{\mathcal{P}}|$ copies of 0. Then output $(\texttt{Output}, m')$.

Remark 2. The decryption step at the end of the protocol can be implemented efficiently in a distributed and verifiable way using known methods (e.g. [15, 32]).

Proposition 5. *Protocol 5 securely realizes $\mathcal{F}_{\text{MN}}$ in the $(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{CF}}, \mathcal{F}_{\text{ZK}}^{\mathcal{R}_{kp}}, \mathcal{F}_{\text{KG}})$-hybrid model with respect to static adversaries corrupting any minority of the mix-servers and any set of senders under the polynomial indistinguishability of the BGN or Paillier cryptosystem respectively.*

9 Complexity Estimates

Our constructions clearly require $O(N^2)$ exponentiations, but we give estimates that show that the constant hidden in the ordo-notation is reasonably small in some practical settings. For simplicity we assume that the cost of squaring a group element equals the cost of multiplying two group elements and that computing an exponentiation using a κ_e-bit integer modulo a κ-bit integer corresponds to κ_e/κ full exponentiations modulo a κ-bit integer. We optimize using fixed-base exponentiation and simultaneous exponentiation (see [21]). We assume that evaluating the bilinear map corresponds to computing 6 exponentiations in the group $\mathbb{G}_1$ and we assume that such one such exponentiation corresponds to 8 modular exponentiations. This seems reasonable, although we are not aware of any experimental evidence. In the Paillier case we assume that multiplication modulo n^s is s^2 times as costly as multiplication modulo n. We assume that the proof of a shuffle requires $8N$ exponentiations (this is conservative).

Most exponentiations when sampling and obfuscating a shuffle are fixed-base exponentiations. The only exception is a single exponentiation each time an element is double-re-encrypted, but there are only N such elements. In the proof of correct obfuscation the bit-size κ_c of the elements in the random vector u used in Protocol 1 is much smaller than the security parameter, and simultaneous exponentiation is applicable. In the Paillier case, simultaneous exponentiation is applicable during evaluation, and precomputation lowers the on-line complexity. Unfortunately, this does not work in the BGN case due to the bilinear map. For a detailed description of how we computed our estimates we refer the reader to the Scheme-program in Appendix C. For practical parameters we get the estimates in Fig. 1.

Construction	Sample & Obfuscate	Prove	Precompute	Evaluate
BGN with $N = 350$	14 (0.5h)	3 (0.1h)	NA	588 (19.6h)
Paillier with $N = 2000$	556 (18.5h)	290 (9.7h)	3800 (127h)	533 (17.8h)

Fig. 1. The table gives the complexity of the operations in terms of 10^4 modular κ-bit exponentiations and in parenthesis the estimated running time in hours assuming that $\kappa = 1024$, $\kappa_c = \kappa_r = 50$, and that one exponentiation takes 12 msec to compute (a 1024-bit exponentiation using GMP [19] takes 12 msec on our 3 GHz PC).

Given a single computer, the BGN construction is only practical when $N \approx 350$ and the maximal number of bits in any submitted ciphertext is small. On the other hand, the Paillier construction is practical for normal sized voting precincts in the USA: $N \approx 2000$ full length messages can be accommodated, and, given one week of pre-computing, the obfuscated shuffle can be evaluated overnight. All constructions are easily parallelized, i.e., larger values of N can be accommodated, or the running time can be reduced by using more computers.

10 Conclusion

It is surprising that a functionality as powerful as a shuffle can be public-key obfuscated in any useful way. It is even more surprising that this can be achieved using the Paillier cryptosystem which, in contrast to the BGN cryptosystem, was not specifically designed to have the kind of "homomorphic" properties we exploit. One intriguing question is whether other useful "homomorphic" properties have been overlooked in existing cryptosystems.

From a practical point of view we stress that, although the performance of our mix-net is much worse than that of known constructions, it exhibits a property which no previous construction has: a relatively small group of mix-servers can prepare obfuscated shuffles for voting precincts. The precincts can compute the shuffling without any private key and produce ciphertexts ready for decryption.

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 *Lecture Notes in Computer Science*, pages 39–50. Springer Verlag, 2003.
2. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology – Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer Verlag, 2001.
3. O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard. Practical multi-candidate election system. In *20th ACM Symposium on Principles of Distributed Computing – PODC*, pages 274–283. ACM Press, 2001.
4. M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology – Eurocrypt '98*, pages 236–250. Springer Verlag, 1998.
5. J. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *5th ACM Symposium on Principles of Distributed Computing – PODC*, pages 52–62. ACM Press, 1986.
6. J. Cohen (Benaloh) and M. Fischer. A robust and verifiable cryptographically secure election scheme. In *28th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–382. IEEE Computer Society Press, 1985.
7. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *2nd Theory of Cryptography Conference (TCC)*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–342. Springer Verlag, 2005.
8. R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Advances in Cryptology – Crypto 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469. Springer Verlag, 1997.
9. 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.).
10. D. Chaum. Untraceable electronic mail, return addresses and digital pseudo-nyms. *Communications of the ACM*, 24(2):84–88, 1981.
11. D. Chaum and T. Pedersen. Wallet Databases with Observers. In *Advances in Cryptology – Crypto 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer Verlag, 1992.
12. R. Cramer, I. Damgard, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology – Crypto 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer Verlag, 1994.
13. R. Cramer, M. Franklin, L. A.M. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. Technical report, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, 1995.
14. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology – Eurocrypt '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer Verlag, 1997.
15. 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 *Lecture Notes in Computer Science*, pages 119–136. Springer Verlag, 2001.
16. P.-A. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography 2000*, volume 2339 of *Lecture Notes in Computer Science*, pages 90–104, London, UK, 2001. Springer-Verlag.
17. J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology – Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer Verlag, 2001.
18. S. Goldwasser and Y. Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 553–562. IEEE Computer Society Press, 2005.
19. T. Granlund. Gnu multiple precision arithmetic library (GMP). Software available at `http://swox.com/gmp`, March 2005.
20. J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *Public Key Cryptography – PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer Verlag, 2003.
21. A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
22. 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.
23. R. Ostrovsky and W. E. Skeith III. Private searching on streaming data. Cryptology ePrint Archive, Report 2005/242, 2005. `http://eprint.iacr.org/`.
24. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer Verlag, 1999.

25. C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology – Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer Verlag, 1994.
26. K. Sako and J. Kilian. Reciept-free mix-type voting scheme. In *Advances in Cryptology – Eurocrypt '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer Verlag, 1995.
27. B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *Advances in Cryptology – Crypto '99*, volume 3027 of *Lecture Notes in Computer Science*, pages 148–164. Springer Verlag, 1999.
28. H. Wee. On obfuscating point functions. In *37th ACM Symposium on the Theory of Computing (STOC)*, pages 523–532. ACM Press, 2005.
29. D. Wikström. A universally composable mix-net. In *1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 315–335. Springer Verlag, 2004.
30. D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *Advances in Cryptology – Asiacrypt 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 273–292. Springer Verlag, 2005. (Full version [31]).
31. D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. Cryptology ePrint Archive, Report 2004/137, 2005. `http://eprint.iacr.org/`.
32. D. Wikström and J. Groth. An adaptively secure mix-net without erasures. In *33rd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4052 of *Lecture Notes in Computer Science*, pages 276–287. Springer Verlag, 2006.

A Proofs

Proof (Proposition 1). Denote by $\mathcal{A}$ an arbitrary adversary in the polynomial indistinguishability experiment run with the obfuscator $\mathcal{O}$. Denote by $\mathcal{A}'$ an adversary to the polynomial indistinguishability experiment $\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS}',\mathcal{A}'}(\kappa)$ with the cryptosystem $\mathcal{CS}'$ defined as follows. It accepts a public key pk as input and forwards it to $\mathcal{A}$. When $\mathcal{A}$ returns (DS_{π_0}, DS_{π_1}), $\mathcal{A}'$ outputs the two messages 0 and 1. Then it is given an encryption $c^{(b)} = \mathcal{E}'_{pk}(b)$. Denote by Λ^{π_0} and Λ^{π_1} the two permutation matrices corresponding to DS_{π_0} and DS_{π_1} respectively. The adversary $\mathcal{A}'$ defines a matrix $C^{\pi_b} = (c^{\pi}_{ij})$, by setting $c^{\pi}_{ij} = \mathcal{E}'_{pk}(\lambda^{\pi_0}_{ij})$ if $\lambda^{\pi_0}_{ij} = \lambda^{\pi_1}_{ij}$ and c^{π}_{ij} to a reencryption of $c^{(b)}$ if $\lambda^{\pi_0}_{ij} = 0$, or to a reencryption of $c^{(1-b)}$ if $\lambda^{\pi_0}_{ij} = 1$. This second ciphertext $c^{(1-b)}$ can be computed homomorphically from $c^{(b)}$. Then $\mathcal{A}'$ continues the simulation using C^{π_b} to compute the obfuscated circuit, and when $\mathcal{A}$ outputs a bit it gives it as its output. Then by construction $\Pr[\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS}',\mathcal{A}}(\kappa) = 1] = \Pr[\mathsf{Exp}^{\mathsf{oind}-b}_{\mathcal{DS}_N,\mathcal{CS}',\mathcal{O},\mathcal{A}}(\kappa) = 1]$ and the proposition follows.

Proof (Proposition 3). Completeness and the fact that the protocol is public-coin follow by inspection. We now concentrate on the more interesting properties.

Zero-Knowledge. The honest verifier zero-knowledge simulator simply picks u randomly as in the protocol and then invokes the honest verifier zero-knowledge simulator of the subprotocol π_{rp}. It follows that the simulated view is indistinguishable from the real view of the verifier.

Negligible Error Probability. Consider the following intuitively appealing lemma.

Lemma 1. *Let η be a product of $\kappa/2$-bit primes and let N be polynomially bounded in κ. Let $\Lambda = (\lambda_{ij})$ be an $N \times N$-matrix over $\mathbb{Z}_\eta$ and let $u \in [0, 2^{\kappa_c} - 1]^N$ be randomly chosen. Then if Λ is not a permutation matrix $\Pr_u[\exists \pi \in \Sigma_N \ : \ u\Lambda^\pi = u\Lambda]$ is negligible.*

Proof. Follows by elementary linear algebra (see [31]).

By assumption $C = \mathcal{E}_{pk}(\Lambda^\pi)$ for some $\pi \in \Sigma_N$. Write $\Lambda = \mathcal{D}_{pk}(C')$. Then the lemma and the soundness of the proof of a shuffle π_{rp} implies the soundness of the protocol.

Knowledge Extraction. For knowledge extraction we may now assume that C' can be formed from C by permuting and re-encrypting its columns. Before we start we state a useful lemma.

Lemma 2. *Let η be a product of $\kappa/2$-bit primes, let N be polynomially bounded in κ, and let $u_1, \ldots, u_{l-1} \in \mathbb{Z}^N$ such that $u_{jj} = 1 \bmod \eta$ and $u_{ji} = 0 \bmod \eta$ for $1 \leq i, j \leq l-1 < N$ and $i \neq j$. Let $u_l \in [0, 2^{\kappa_c} - 1]^N$ be randomly chosen, where $2^{-\kappa_c}$ is negligible. Then the probability that there exists $a_1, \ldots, a_l \in \mathbb{Z}$ such that if we define $u'_l = \sum_{j=1}^{l} a_j u_j \bmod \eta$, then $u'_{l,l} = 1 \bmod \eta$, and $u'_{l,i} = 0 \bmod \eta$ for $i < l$ is overwhelming in κ.*

Proof. Note that $b = u_{l,l} - \sum_{j=1}^{l-1} u_{l,j} u_{j,l}$ is invertible with overwhelming probability, and when it is we view its inverse b^{-1} as an integer and define $a_j = -b^{-1} u_{l,j}$ for $j < l$ and $a_l = b^{-1}$. For $i < l$ this gives $u_{l,i} = \sum_{j=1}^{l} a_j u_{j,i} = b^{-1}(1 - a_i u_{ii}) = 0 \bmod \eta$ and for $i = l$ this gives $u_{l,l} = \sum_{j=1}^{l} a_j u_{j,l} = b^{-1}(u_{l,l} - \sum_{j=1}^{l-1} u_{l,j} u_{j,l}) = 1 \bmod \eta$.

It remains to exhibit a knowledge extractor. By assumption there exists a polynomial $t(\kappa)$ and negligible knowledge error $\epsilon(\kappa)$ such that the extractor of the subprotocol π_{rp} executes in time $T_{\gamma'}(\kappa) = t(\kappa)/(\gamma' - \epsilon(\kappa))$ for every common input (d, d'), induced by a random vector u, to the subprotocol such that the success probability of the subprotocol is γ'. We invoke the extractor, but we must stop it if γ' turns out to be too low and find a new random u that induces a common input to the subprotocol with a larger value of γ'. For simplicity we assume that the same negligible function $\epsilon(\kappa)$ bounds the failure probability in Lemma 2. We assume that $\epsilon(\kappa) < \gamma/4$, i.e., the knowledge error will increase somewhat compared to the knowledge error of π_{rp}.

Consider a fixed common input (pk, C, C') and prover $\mathcal{P}$. Denote by γ the probability that $\mathcal{P}$ convinces $\mathcal{V}$. We denote by B the distribution over $\{0, 1\}$ given by $p_B(1) = \gamma/(8t(\kappa))$. Note that this distribution can be sampled for any common input even without knowledge of γ, since we can simply perform a simulation of the protocol, pick an element from the space $\{1, \ldots, 8t(\kappa)\}$ randomly, and define the sample to be one if the prover succeeds and the picked element equal one. We are going to use the random variable to implicitly be able to say if an induced common input to the subprotocol gives a too low success probability γ'. We now make this idea precise. The extractor proceeds as follows, where in the BGN case η denotes the modulus n and in the Paillier case η denotes the order of the plaintext space of the outer layer Paillier, i.e., n^2 where n is the modulus and encryption is defined modulo n^3.

1. For $l = 1, \ldots, N$ do:
   (a) Start the simulation of an execution between $\mathcal{V}$ and $\mathcal{P}$ and denote by u_l the random vector chosen by the simulator. Denote by (pk, d_l, d'_l) the common input to the subprotocol π_{rp} induced by u_l.
   (b) If $u_{l,j} = u_{l,j'}$ for some $j \neq j'$ or if there does not exists $a_{k,l} \in \mathbb{Z}$ such that $\sum_{l'=1}^{l} a_{k,l'} u_{l',j}$ equals one modulo η if $j = l$ and it equals zero modulo η for $j < l$, then go to Step 1a.
   (c) Invoke the knowledge extractor of the protocol π_{rp} on the common input (pk, d_l, d'_l). However, in between each step executed by the extractor, the distribution B is sampled. If a sample equals one before the extractor halts, then go to Step 1a. Otherwise, denote by π_l and s_l the permutation and extracted randomness such that $((pk, d_l, d'_l), (\pi_l, s_l)) \in \mathcal{R}_{rp}$.
2. Compute $a_{k,l} \in \mathbb{Z}$ such that $\sum_{l=1}^{N} a_{k,l} u_{l,j}$ equals one or zero modulo η depending on if $k = j$ or not. Define $(b_{kj}) = (a_{kl})(u_{lj}) - I$, where I is the identity $N \times N$-matrix and the matrix operations are taken over the integers.

   In BGN Case. Compute $r = (r_{k,j}) = (a_{k,l})(s_{l,j})$, and output (π, r).

   In Paillier Case. Compute $r = (r_{k,j}) = (\prod_{i=1}^{N} (c_{i,\pi(j)}/c'_{ij})^{b_{ki}/\eta} \prod_{l=1}^{N} s_{l,j}^{a_{k,l}})$, where the division b_{ki}/η is taken over the integers, and output (π, r).

We do the easy part of the analysis first. Consider the correctness of the output given that the extractor halts. Since $u_{l,j} = u_{l,j'}$ for all $j \neq j'$ and both $\mathcal{D}_{pk}(C)$ and $\mathcal{D}_{pk}(C')$ are permutation matrices by assumption, we conclude that $\pi_1 = \ldots = \pi_N = \pi$ for some permutation $\pi \in \Sigma_N$. We have

$$\prod_{i=1}^{N}(c'_{ij})^{u_{li}} = d'_{lj} = d_{l,\pi(j)}\mathcal{E}_{pk}(0, s_{l,j}) = \mathcal{E}_{pk}(0, s_{l,j})\prod_{i=1}^{N} c_{i,\pi(j)}^{u_{li}} \ . \tag{1}$$

Apply the $a_{k,l}$ as exponents on the left of Equation (1) and take the product over all l. This gives

$$\prod_{l=1}^{N}\left(\prod_{i=1}^{N}(c'_{ij})^{u_{li}}\right)^{a_{k,l}} = \prod_{i=1}^{N}\left(\prod_{l=1}^{N}(c'_{ij})^{a_{k,l}u_{li}}\right) = c'_{kj}\prod_{i=1}^{N}(c'_{ij})^{b_{ki}} \ .$$

Then apply the exponents $a_{k,l}$ on the right side of Equation (1) and take the product over all l. This gives

$$\prod_{l=1}^{N}\left(\mathcal{E}_{pk}(0, s_{l,j})\prod_{i=1}^{N} c_{i,\pi(j)}^{u_{li}}\right)^{a_{k,l}} = \left(\prod_{l=1}^{N}\mathcal{E}_{pk}(0, s_{l,j})^{a_{kl}}\right)\left(\prod_{i=1}^{N} c_{i,\pi(j)}^{b_{ki}}\right) c_{k,\pi(j)} \ .$$

To summarize, we have shown that

$$c'_{kj} = \left(\prod_{i=1}^{N}(c_{i,\pi(j)}/c'_{ij})^{b_{ki}}\right)\left(\prod_{l=1}^{N}\mathcal{E}_{pk}(0, s_{l,j})^{a_{kl}}\right) c_{k,\pi(j)} \ .$$

We conclude the argument differently depending on the cryptosystem used.

In BGN Case. Note that $b_{ki} = 0 \bmod \eta$ for all k and i, and the order of any ciphertext divides η. Thus, the first product equals one in the ciphertext group. Furthermore, the randomizer space is $\mathbb{Z}_\eta$ so we have

$$c'_{kj} = \mathcal{E}_{pk}\left(0, \sum_{l=1}^{N} a_{kl}s_{l,j}\right) c_{k,\pi(j)} \ .$$

In Paillier Case. Again $b_{ki} = 0 \bmod \eta$ for all k and i, but the order a ciphertext may be larger than η. However, we may define $b'_{ki} = b_{ki}/\eta$, where division is over the integers, define $s'_j = \prod_{i=1}^{N}(c_{i,\pi(j)}/c'_{ij})^{b'_{ki}}$, and write

$$c'_{kj} = \mathcal{E}_{pk}\left(0, s'_j\prod_{l=1}^{N} s_{l,j}^{a_{kl}}\right) c_{k,\pi(j)} \ .$$

We remark that s'_j is an element in $\mathbb{Z}^*_{n^3}$ and not in $\mathbb{Z}^*_n$ as expected. However, it is a witness of re-encryption using one of the alternative Paillier encryption algorithms.

It remains to prove that the extractor is efficient in terms of the inverse success probability of the prover. Fix an l. Denote by E the event that the prover succeeds to convince the adversary, i.e., $\Pr[E] = \gamma$. Denote by S the set of vectors u such that $\Pr[E \mid u \in S] \geq \gamma/2$. An averaging argument implies that $\Pr[u \in S] \geq \gamma/2$. Denote by E_{u_l} the event that the go to statement in Step 1b is executed. We show that if $u \in S$, then a witness is extracted efficiently with constant probability, and if $u \notin S$, then the extraction algorithm will be stopped relatively quickly.

If $u \notin S$, then we focus only on the distribution B. The expected number of samples from B needed before a sample is equal to one is clearly $1/p_B(1) = 8t(\kappa)/\gamma$. Thus, if we ignore the issue of finding the witness the simulation in Step 1c is efficient in terms of $1/\gamma$.

If $u \in S$, then the expected number of steps needed by the extractor of the subprotocol π_{rp} is bounded by $T_{\gamma/2}(\kappa)$. By Markov's inequality the probability that more than $2T_{\gamma/2}(\kappa)$ steps are needed is bounded by $1/2$. The probability that one of the first $\omega = 2T_{\gamma/2}(\kappa)$ samples of B is one is bounded by $1 - (1 - p_B(1))^\omega \leq 1 - e^{\omega(-p_B(1)+p_B(1)^2)} \leq 1 - e^{-1/2}$, since $\epsilon(\kappa) < \gamma/4$.

Thus, Step 1c executes in expected time $8t(\kappa)/\gamma$, and from independence follows that it halts due to the extractor finding a witness with probability at least $1 - \frac{1}{2}(1 - e^{-1/2})$. In other words the expected number of restarts of the lth iteration of Step 1 is constant.

From Lemma 2 and independence of the $u_{l,j}$ follow that the probability that the go to statement of Step 1b is executed is negligible. This means that the extractor runs in expected time $cNt(\kappa)/(\gamma - 4\epsilon(\kappa))$ for some constant c. This concludes the proof, since $cNt(\kappa)$ is polynomial and $4\epsilon(\kappa)$ is negligible.

Proof (Proposition 2). Let $\mathcal{A}$ be any adversary in the polynomial indistinguishability experiment run with the obfuscator $\mathcal{O}^{\mathsf{pai}}$. Denote by $\mathcal{A}_{ind}$ the polynomial indistinguishability adversary that takes a public key pk as input and then simulates this protocol to $\mathcal{A}$. When $\mathcal{A}$ outputs two challenge circuits $(RS_0^{\mathsf{pai}}, RS_1^{\mathsf{pai}})$ with corresponding matrices (M_0, M_1), i.e., the matrices are permutation matrices with the ones replaced by re-encryption factors, $\mathcal{A}_{ind}$ outputs (M_0, M_1). When the polynomial indistinguishability experiment returns $\mathcal{E}_{pk}^{\mathsf{pai}}(M_b)$ it forms the obfuscated circuit and hands it to $\mathcal{A}$. Then $\mathcal{A}_{ind}$ outputs the output of $\mathcal{A}$. It follows that the advantage of $\mathcal{A}_{ind}$ in the polynomial indistinguishability experiment with the Paillier cryptosystem and using polynomial length list of ciphertexts is identical to the advantage of $\mathcal{A}$ in the polynomial indistinguishability experiment with $\mathcal{O}^{\mathsf{pai}}$. It now follows from a standard hybrid argument that the polynomial indistinguishability of the Paillier cryptosystem is broken if $\mathcal{O}^{\mathsf{pai}}$ is not polynomially indistinguishable.

Proof (Proposition 4). Completeness and the public-coin property follow by inspection. The honest verifier zero-knowledge simulator simply picks $e \in [0, n2^{\kappa_r}]$ and $f \in [0, n^3 2^{\kappa_r}]$ and $b \in \{0, 1\}$ randomly and defines $\alpha = ((c')^b c^{1-b})^{h_2^e} h_3^f \bmod n^3$. The resulting view is statistically close to a real view, since $2^{-\kappa_r}$ is negligible.

For soundness, note that if we have $c^{h_2^e} h_3^f = \alpha = (c')^{h_2^{e'}} h_3^{f'} \bmod n^3$ with $e, f, e', f' \in \mathbb{Z}$, then we can divide by h_3^f and take the h_2^eth root on both sides. This gives

$$c = (c')^{h_2^{e'-e}} h_3^{(f'-f)/h_2^e} \bmod n^3 \ ,$$

which implies that the basic protocol is special 1/2-sound. The protocol is then iterated in parallel κ_c times which gives negligible error probability $2^{-\kappa_c}$. The proof of knowledge property follows immediately from special soundness.

A.1 Proof of Proposition 5

The proof proceeds as most proofs of security in the UC-framework. First we define an ideal adversary $\mathcal{S}$ that runs the real adversary $\mathcal{A}$ as a black-box. Then we show that if the environment can distinguish the ideal model run with the ideal adversary from the real model run with the real adversary with non-negligible probability, then we can reach a contradiction.

The Ideal Adversary. The ideal adversary simulates the view of the real model to the real adversary. Denote by $I_{\mathcal{M}}$ and $I_{\mathcal{P}}$ the indices of the corrupted mix-servers and senders correspondingly. The ideal adversary corrupts the corresponding dummy parties. Then it simulates the real model as follows.

Links Between Corrupted Parties and the Environment. The ideal adversary simulates the simulated environment $\mathcal{Z}'$ such that it appears as if $\mathcal{A}$ is communicating directly with $\mathcal{Z}$. Similarly, it simulates the corrupted dummy party $\tilde{\mathcal{M}}_j$ (or $\tilde{\mathcal{P}}_i$) for $j \in I_{\mathcal{M}}$ (or $i \in I_{\mathcal{P}}$) such that it appears as if $\mathcal{M}_j$ (or $\mathcal{P}_i$) is directly communicating with $\mathcal{Z}$. This is done by simply forwarding messages.

Simulation of Honest Senders. The ideal adversary clearly does not know the messages submitted by honest senders to $\mathcal{F}_{\text{MN}}$ in the ideal model. Thus, it must use some place holders in its simulation and then make sure that this is not noticed.

Honest senders $\mathcal{P}_i$ with $i \notin I_{\mathcal{P}}$ are simulated as follows. When $\mathcal{S}$ receives $(\mathcal{S}, \mathcal{P}_i, \texttt{Input}, f)$ it simulates $\mathcal{P}_i$ honestly on input $(\texttt{Send}, 0)$. It interrupt the simulation of $\mathcal{F}_{\text{BB}}$, when it is about to hand $(\mathcal{A}, \texttt{Input}, f', \mathcal{P}_i, c_i)$ to $\mathcal{C}_{\mathcal{I}}$. Then it stores (f, f') and continues the simulation.

When $\mathcal{F}_{\text{BB}}$ receives $(\mathcal{A}, \texttt{AcceptInput}, f')$ from $\mathcal{C}_{\mathcal{I}}$ it interrupts the simulation of $\mathcal{F}_{\text{BB}}$. Then it hands $(\texttt{AcceptInput}, f)$ to $\mathcal{F}_{\text{MN}}$ and waits until it receives a message from $\mathcal{F}_{\text{MN}}$ before it continues the simulation of $\mathcal{F}_{\text{BB}}$.

Extraction From Corrupt Senders. When a corrupt sender submits a message in the simulated real model, then the ideal adversary must instruct the corresponding corrupted dummy sender to submit the same message.

When some $\mathcal{M}_j$ with $j \notin I_{\mathcal{M}}$ receives $(\mathcal{M}_j, \texttt{Verifier}, \mathcal{P}_\gamma, (pk, c_\gamma), 1)$ from $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{kp}}$ in the simulation of Step 3b, the simulation is interrupted and the message m_γ encrypted in c_γ is extracted from the simulation of $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{kp}}$. Then $\tilde{\mathcal{P}}_\gamma$ is instructed to submit m_γ to $\mathcal{F}_{\text{MN}}$. When $\mathcal{S}$ receives $(\tilde{\mathcal{P}}_\gamma, \texttt{Input}, f)$ it hands $(\texttt{AcceptInput}, f)$ to $\mathcal{F}_{\text{MN}}$ and waits until it receives $(\tilde{\mathcal{P}}_i, \texttt{Send})$ from $\mathcal{F}_{\text{MN}}$. Then the simulation of $\mathcal{M}_j$ is continued.

Simulation of Honest Mix-Servers. When an honest dummy mix-server signals that it wishes to start the mixing process, the corresponding simulated mix-server must do the same.

When $\mathcal{S}$ receives $(\tilde{\mathcal{M}}_j, \texttt{Input}, f)$ from $\mathcal{F}_{\text{MN}}$, with $j \notin I_{\mathcal{M}}$, it gives the simulated mix-server $\mathcal{M}_j$ the input `Run`. When $\mathcal{F}_{\text{BB}}$ is about to output $(\mathcal{A}, \texttt{Input}, f', \mathcal{M}_j, \texttt{Run})$ the simulation is interrupted and (f, f') stored before the simulation is continued. When $\mathcal{F}_{\text{BB}}$ receives $(\mathcal{A}, \texttt{AcceptInput}, f')$ the simulation of $\mathcal{F}_{\text{BB}}$ is interrupted and $\mathcal{S}$ hands $(\texttt{AcceptInput}, f)$ to $\mathcal{F}_{\text{MN}}$. When it returns the simulation is continued. Note that it normally returns $(\tilde{\mathcal{M}}_j, \texttt{Run})$ or $(\tilde{\mathcal{M}}_j, \texttt{Output}, L')$, but the empty message can be returned if the accept instruction is ignored.

Extraction From Corrupt Mix-Servers. When a corrupted mix-server signals that it wishes to start the mixing process in the simulated real model, the corresponding corrupted dummy mix-server must do the same.

When $\mathcal{F}_{\text{BB}}$ is about to hand $(\mathcal{A}, \texttt{AcceptInput}, f')$ to $\mathcal{C}_{\mathcal{I}}$ and $(\mathcal{M}_j, \texttt{Run})$ has been stored in the database D_1 of $\mathcal{F}_{\text{BB}}$, the simulation is interrupted. Then $\mathcal{S}$ instructs $\tilde{\mathcal{M}}_j$ to hand `Run` to $\mathcal{F}_{\text{MN}}$ and waits until it receives $(\tilde{\mathcal{M}}_j, \texttt{Input}, f)$ from $\mathcal{F}_{\text{MN}}$. Then it hands $(\texttt{AcceptInput}, f)$ to $\mathcal{F}_{\text{MN}}$. When it returns the simulation of $\mathcal{F}_{\text{BB}}$ is continued.

Simulation of Decryption. Note that when the decryption step is simulated by $\mathcal{F}_{\text{KG}}$, $\mathcal{S}$ already knows the output L' of $\mathcal{F}_{\text{MN}}$. Simulation proceeds slightly differently in the BGN and Paillier cases, but in both cases a list $(m'_1, \ldots, m'_N)$ is formed by padding L' with zeros until it has size N.

BGN Case. Choose $\pi_{sim} \in \Sigma_N$ randomly and use $(m'_{\pi_{sim}(1)}, \ldots, m'_{\pi_{sim}(N)})$ in the simulation of $\mathcal{F}_{\text{KG}}$.

Paillier Case. In this case the key generator is called twice to decrypt the outer and inner layer of the ciphertexts respectively. Choose $\pi_{sim} \in \Sigma_N$ and $\bar{r}_i \in \mathbb{Z}_n^*$ randomly and compute

$c'' = (c''_1, \ldots, c''_N) = (\mathcal{E}^{\mathsf{pai}}_{n,2}(m'_{\pi_{sim}(1)}, \bar{r}_1), \ldots, \mathcal{E}^{\mathsf{pai}}_{n,2}(m'_{\pi_{sim}(N)}, \bar{r}_N))$. In the first call use c'' in the simulation of $\mathcal{F}_{\mathrm{KG}}$ and in the second call use $(m'_{\pi_{sim}(1)}, \ldots, m'_{\pi_{sim}(N)})$.

Reaching a Contradiction. Consider any real adversary $\mathcal{A}$ and environment $\mathcal{Z}$. We show that if $\mathcal{Z}$ can distinguish the ideal model run with $\mathcal{S}$ from the real model run with $\mathcal{A}$, then we can break the indistinguishability of the underlying cryptosystem.

Denote by T_{ideal} a simulation of the ideal model run with $\mathcal{S}$ and $\mathcal{Z}$ and denote by T_{real} a simulation of the real model run with $\mathcal{A}$ and $\mathcal{Z}$. Denote by T_l the simulation of $T_0 = T_{ideal}$ except for the following modifications of simulation honest senders $\mathcal{P}_i$ for $i \notin I_{\mathcal{P}}$ and $i < l$ and $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}_{kp}}$.

1. $\mathcal{P}_i$ submits $(\texttt{Prover}, (pk, c), \perp)$ to $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}_{kp}}$, i.e., it does not submit any witness. Instead, the simulation of $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}_{kp}}$ is modified to behave as if it received a witness from these senders.
2. Instead of giving $\mathcal{P}_i$ the zero input, $\mathcal{S}$ peeks into the ideal functionality $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}_{kp}}$ and uses the message m_i submitted by the corresponding honest dummy sender $\tilde{\mathcal{P}}_i$.

Claim 1. $|\Pr[T_0 = 1] - \Pr[T_N = 1]|$ is negligible.

Proof. This follows by a standard hybrid argument. We need only observe that the secret key of the cryptosystem is not used by $\mathcal{S}$ in its simulation. More precisely, define $\mathcal{A}_l$ to be the polynomial indistinguishability adversary for the cryptosystem that takes a public key pk as input and simulates T_l, except that if $l \notin I_{\mathcal{P}}$ it interrupts the simulation when $\mathcal{P}_l$ is about to compute it submission ciphertext. Then it hands $(M_0, M_1) = (m_l, 0)$ to the experiment, where m_l is the message that $\tilde{\mathcal{P}}_l$ handed to $\mathcal{F}_{\mathrm{MN}}$. It is given a challenge ciphertext $c_l = \mathcal{E}_{pk}(M_b)$ for a randomly chosen $b \in \{0, 1\}$ which it uses in the continued simulation.

By inspection we see that $\Pr[\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS},\mathcal{A}_l}(\kappa) = 1] = \Pr[T_{l-b} = 1]$. The polynomial indistinguishability of the cryptosystem then implies that $|\Pr[T_l = 1] - \Pr[T_{l+1} = 1]|$ is negligible for $l = 1, \ldots, N$. The triangle inequality and the fact that N is polynomially bounded then implies that $|\Pr[T_0 = 1] - \Pr[T_N = 1]|$ is negligible as claimed.

Informally speaking we have now plugged back the correct messages in the simulation. The problem is that decryption is still simulated incorrectly. In other words T_N is still not identically distributed T_{real}. To prove that the distributions are indistinguishable we need to sample the latter distribution without using the secret key of the cryptosystem. We do this using the knowledge extractors of the proofs of knowledge of correct obfuscation. One of the honest mix-servers must also simulate its proof of correct re-encryption and permutation of a matrix of ciphertexts.

Denote by B_b the simulation of T_N or T_{real} (depending on if $b = 0$ or not) except that in the simulation, if a corrupt mix-server $\mathcal{M}_j$ with $j \in I_{\mathcal{M}}$ succeeds in Protocol 1 or Protocol 2 then the knowledge extractor of the protocol is invoked to extract the witness. In the Paillier case we assume that in the same way the knowledge extractors of corrupt mix-servers are invoked in Protocol 3. Denote the maximal knowledge error of Protocol 1 and Protocol 2 by $\epsilon = \epsilon(\kappa)$, and recall that the knowledge error of a proof of knowledge does not depend on the adversary, but is a parameter of the protocol itself. Denote then by $t(\kappa)/(\delta - \epsilon)$, where $t(\kappa)$ is some polynomial, the expected running time of the extractor in any of these protocols for a prover with success probability δ, and recall that also $t(\kappa)$ is a parameter of the protocol. In the simulation carried out by B_b the running time of any extractor is restricted to $t(\kappa)/\epsilon$ and if extraction fails it outputs 0.

Claim 2. $|\Pr[B_0 = 1] - \Pr[T_N = 1]|$ and $|\Pr[B_1 = 1] - \Pr[T_{real} = 1]|$ is negligible.

Proof. Recall that Protocol 1 assumes that the first matrix C is an encryption of a permutation matrix (or in the Paillier case a permutation matrix where the ones have been replaced by inner encryptions of zero). Due to the negligible error probability of the protocols, the probability that a mix-server succeeds to prove a false statement is negligible. Thus, we assume without loss that all statements for which the extractors are invoked there exists a witness.

Then consider the event E_l that in the lth proof computed by any mix-server $\mathcal{M}_j$ it succeeds with probability less than 2ϵ conditioned on the simulation up to this point, and still succeeds in the actual simulation. We clearly have $\Pr[E_l] < 2\epsilon$. The union bound then implies the claim, since there are at most $2k$ invocations of the protocols in total.

Claim 3. B_b runs in expected polynomial time.

Proof. This follows, since at any instance where an extractor is invoked for a prover on some statement, if the prover succeeds with probability $\delta > 2\epsilon$, then the extractor runs in expected time at most $2t(\kappa)/\delta$ (although it may not output a witness at all), and otherwise the running time is bounded by $t(\kappa)/\epsilon$. Thus, in total this part of the simulation runs in expected time $2t(\kappa)$. As the extractors are only invoked polynomially many times, the complete simulation runs in expected polynomial time.

Observe that B_1 can be sampled even without the secret key, since all the random permutations and all random exponents are extracted. More precisely, since the list of original inputs and how they are permuted (and inner-layer re-encrypted in the Paillier case), is known by the simulator it can simulate decryption perfectly. Assume from now on that this is done.

Denote by B'_b the simulation of B_b except for the following modification. Denote by $\mathcal{M}_l$ some fixed mix-server with $l \notin I_{\mathcal{M}}$. Instead of letting it execute the prover of Protocol 1, or Protocol 2 the honest verifier zero-knowledge simulator guaranteed to exist by Proposition 3 and Proposition 4 respectively is invoked by programming the coin-flipping functionality $\mathcal{F}_{\text{CF}}$.

Claim 4. $|\Pr[B_b = 1] - \Pr[B'_b = 1]|$ is negligible.

Proof. This follows directly from the honest verifier zero-knowledge property of Protocol 1 and Protocol 2.

BGN Case. Simply write B''_0 instead of B'_0 to allow a single proof for the two cases (we are taking care about some special features of the Paillier case below).

Paillier Case. In this case we need to take care of the fact that the inner re-encryption factors used by the simulator to simulate decryption are independently chosen from the true re-encryption factors generated in Protocol 3.

Denote by B''_0 the simulation of B'_0 except that the former uses the re-encryption factors extracted from the executions of Protocol 2.

Claim 5 (Paillier Case). $|\Pr[B'_0 = 1] - \Pr[B''_0 = 1]|$ is negligible.

Proof. Denote by $\mathcal{A}_{ind}$ the polynomial indistinguishability adversary that takes n as input and simulates B'_0 except that in Protocol 3 $\mathcal{M}_l$ computes its output c_j as follows. It generates two random lists $r_i^{(0)}, r_i^{(1)} \in (\mathbb{Z}_n^*)^N$ and hands these to the experiment. The experiment returns $c_j = \mathcal{E}_{n^3}^{\mathsf{pai}}(r_i^{(b)})$ for a random $b \in \{0,1\}$, which is used in the continued simulation. Then it defines $\bar{r}_i = r_i^{(1)} \prod_{j=l+1}^{k} r_{j,i}$, where $r_{j,i}$ are the values extracted by the knowledge extractors or chosen by simulated honest mix-servers. Note that if $b = 0$, the the simulation is identically distributed to B'_0 and otherwise statistically close distributed to B''_0.

The standard extension of polynomial indistinguishability to polynomial length lists of ciphertexts now implies the claim.

At this point we have reduced the difference between B_0'' and B_1'' to how decryption is simulated. In the former decryption is simulated incorrectly by simply outputting the correct messages in some randomlyt chosen order, whereas in the latter the corresponce between individual ciphertexts and output messages is preserved.

Claim 6. $|\Pr[B_0'' = 1] - \Pr[B_1'' = 1]|$ is negligible.

Proof. Consider the following polynomial indistinguishability adversary $\mathcal{A}_{ind}$ to the obfuscation of the decryption/re-encryption shuffle. It accepts a public key pk as input. Then it simulates B_0'' until some fixed simulated honest mix-server $\mathcal{M}_l$ with $l \notin I_{\mathcal{M}}$ is about to produce its output in the mix-net.

The adversary $\mathcal{A}_{ind}$ chooses $\pi^{(0)}, \pi^{(1)} \in \Sigma_N$ randomly and defines $\pi_{sim} = \pi^{(1)}\pi_{l+1} \cdots \pi_k$, and $\pi_l = \pi^{(0)}$. Due to the group properties of Σ_N, this does not change the distribution of π_{sim} in either B_0'' or B_1''.

BGN Case. The adversary $\mathcal{A}_{ind}$ hands $(DS^{\mathsf{bgn}}_{\pi^{(0)}}, DS^{\mathsf{bgn}}_{\pi^{(1)}})$ to the experiment and waits until it returns an obfuscation $\mathcal{O}(pk, sk, DS^{\mathsf{bgn}}_{\pi^{(b)}})$ for a random b. It extracts the encrypted permutation matrix $C^{\pi^{(b)}}$ from the obfuscated shuffle and uses it as $\mathcal{M}_l$'s output.

Paillier Case. Denote by $r = (r_1, \ldots, r_N)$ the joint random factors such that $\mathcal{D}^{\mathsf{pai}}_{p,3}(C_0)$ is the diagonal matrix with diagonal $(r_i^n \bmod n^2)$. These factors can be computed, from the witnesses extracted from the corrupt mix-servers in Protocol 2. The adversary $\mathcal{A}_{ind}$ hands $(RS^{\mathsf{pai}}_{\pi^{(0)},r}, RS^{\mathsf{pai}}_{\pi^{(1)},r})$ to the experiment and waits until it returns an obfuscation $\mathcal{O}(pk, sk, RS^{\mathsf{pai}}_{\pi^{(b)}})$ for a random b. It extracts the encrypted re-encryption and permutation matrix $C^{\pi^{(b)}}$ from the obfuscated shuffle and uses it as $\mathcal{M}_l$'s output.

We conclude that $\mathsf{Exp}^{\mathsf{oind}-b}_{\mathcal{RS}^{\mathsf{pai}}_N, \mathcal{A}_{ind}}(\kappa)$ is identically distributed to B_b''. Then the claim follows from the polynomial indistinguishability of the cryptosystem, since an expected polynomial time distinguisher can be turned into a strict polynomial time distinguisher using Markov's inequality in the standard way.

Conclusion of Proof of Proposition. To conclude the proof we simply note that Claim 1-6 implies that $|\Pr[T_{ideal}(\mathcal{A}) = 1] - \Pr[T_{real}(\mathcal{A}) = 1]|$ is negligible for any real adversary $\mathcal{A}$.

B Ideal Functionalities

Functionality 2 (Bulletin Board). The ideal bulletin board functionality, $\mathcal{F}_{\mathrm{BB}}$, running with parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and ideal adversary $\mathcal{S}$ proceeds as follows. $\mathcal{F}_{\mathrm{BB}}$ holds two databases D_1 and D_2 indexed on integers. Initialize two counters $c_1 = 0$ and $c_2 = 0$.

- Upon receiving $(\mathcal{P}_i, \texttt{Write}, m_i)$, $m_i \in \{0,1\}^*$, from $\mathcal{C}_{\mathcal{I}}$, store $(\mathcal{P}_i, m_i)$ in D_2 by the index c_2 in the database, set $c_2 \leftarrow c_2 + 1$, and hand $(\mathcal{S}, \texttt{Input}, c_2, \mathcal{P}_i, m_i)$ to $\mathcal{C}_{\mathcal{I}}$.
- Upon receiving $(\mathcal{S}, \texttt{AcceptInput}, c)$ from $\mathcal{C}_{\mathcal{I}}$ check if a tuple $(\mathcal{P}_i, m_i)$ is stored in the database D_2 under c. If so, then store $(\mathcal{P}_i, m_i)$ in D_1 under the index c_1, set $c_1 \leftarrow c_1 + 1$, and hand $(\mathcal{S}, \texttt{AcceptInput}, c)$ to $\mathcal{C}_{\mathcal{I}}$.
- Upon receiving $(\mathcal{P}_j, \texttt{Read}, c)$ from $\mathcal{C}_{\mathcal{I}}$ check if a tuple $(\mathcal{P}_i, m_i)$ is stored in the database D_1 under c. If so hand $((\mathcal{S}, \mathcal{P}_j, \texttt{Read}, c, \mathcal{P}_i, m), (\mathcal{P}_j, \texttt{Read}, c, \mathcal{P}_i, m_i))$ to $\mathcal{C}_{\mathcal{I}}$. If not, hand $((\mathcal{S}, \mathcal{P}_j, \texttt{NoRead}, c), (\mathcal{P}_j, \texttt{NoRead}, c))$ to $\mathcal{C}_{\mathcal{I}}$.

Functionality 3 (Coin-Flipping). The ideal Coin-Flipping functionality, $\mathcal{F}_{\mathrm{CF}}$, with mix-servers $\mathcal{M}_1, \ldots, \mathcal{M}_k$, and adversary $\mathcal{S}$ proceeds as follows. Set $J_\kappa = \emptyset$ for all κ.

– On receipt of $(\mathcal{M}_j, \texttt{GenerateCoins}, \kappa)$ from $\mathcal{C}_\mathcal{I}$, set $J_\kappa \leftarrow J_\kappa \cup \{j\}$. If $|J_\kappa| = k$, then set $J_\kappa \leftarrow \emptyset$ choose $c \in \{0,1\}^\kappa$ randomly and hand $((\mathcal{S}, \texttt{Coins}, c), \{(\mathcal{M}_j, \texttt{Coins}, c)\}_{j=1}^k)$ to $\mathcal{C}_\mathcal{I}$.

Functionality 4 (Key Generator). The ideal key generator $\mathcal{F}_{\mathrm{KG}}$, running with mix-servers $\mathcal{M}_1, \ldots, \mathcal{M}_k$, senders $\mathcal{P}_1, \ldots, \mathcal{P}_N$, and ideal adversary $\mathcal{S}$ proceeds as follows

1. Initialize a database D. Compute $(pk, sk) = \mathcal{G}(1^\kappa)$ and hand $((\mathcal{S}, \texttt{PublicKey}, pk), \{(\mathcal{M}_j, \texttt{PublicKey}, pk)\}_{j=1}^k, \{(\mathcal{P}_i, \texttt{PublicKey}, pk)\}_{i=1}^N)$ to $\mathcal{C}_\mathcal{I}$.
2. Repeatedly wait for messages. Upon reception of $(\mathcal{M}_j, \texttt{Decrypt}, c)$, set $D \leftarrow D \cup \{(\mathcal{M}_j, c)\}$, and if $|\{j : (\mathcal{M}_j, c) \in D\}| > k/2$, then hand $((\mathcal{S}, \texttt{PublicKey}, pk), \{(\mathcal{M}_j, \texttt{PublicKey}, pk)\}_{j=1}^k)$ to $\mathcal{C}_\mathcal{I}$.

Functionality 5 (Zero-Knowledge Proof of Knowledge). Let $\mathcal{L}$ be a language given by a binary relation $\mathcal{R}$. The ideal *zero-knowledge proof of knowledge* functionality $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$ of a witness w to an element $x \in \mathcal{L}$, running with provers $\mathcal{P}_1, \ldots, \mathcal{P}_N$, and verifiers $\mathcal{M}_1, \ldots, \mathcal{M}_k$, proceeds as follows.

1. Upon receipt of $(\mathcal{P}_i, \texttt{Prover}, x, w)$ from $\mathcal{C}_\mathcal{I}$, store w under the tag $(\mathcal{P}_i, x)$, and hand $(\mathcal{S}, \mathcal{P}_i, \texttt{Prover}, x, R(x, w))$ to $\mathcal{C}_\mathcal{I}$. Ignore further messages from $\mathcal{P}_i$.
2. Upon receipt of $(\mathcal{M}_j, \texttt{Question}, \mathcal{P}_i, x)$ from $\mathcal{C}_\mathcal{I}$, if $J_{\mathcal{P}_i,x}$ is not initialized set $J_{\mathcal{P}_i,x} = \emptyset$ and otherwise $J_{\mathcal{P}_i,x} \leftarrow J_{\mathcal{P}_i,x} \cup \{j\}$. Let w be the string stored under the tag $(\mathcal{P}_i, x)$ (the empty string if nothing is stored). If $|J_{\mathcal{P}_i,x}| = k$, then hand $((\mathcal{S}, \mathcal{M}_j, \texttt{Verifier}, \mathcal{P}_i, x, R(x, w)), \{(\mathcal{M}_j, \texttt{Verifier}, \mathcal{P}_i, x, R(x, w))\}_{j=1}^k)$ to $\mathcal{C}_\mathcal{I}$ and otherwise $(\mathcal{S}, \mathcal{M}_j, \texttt{Question}, \mathcal{P}_i, x)$.

Definition 13. *Let $\mathcal{CS}$ be a cryptosystem. Then denote by $\mathcal{R}_{kp}$ the relation consisting of pairs $((pk, c), (m, r))$ such that $c = \mathcal{E}_{pk}(m, r)$ and $m \in \{0,1\}^{\kappa_m}$.*

C Program for Estimation of Complexity

```
;; Program to estimate the complexity of sampling, obfuscating,
;; proving correctness of an obfuscation, and evaluating.

;; isbgn:   If equal to 1 we compute BGN complexity and otherwise Paillier
;; secp:    Main security parameter, number of bits in Paillier modulus.
;; secpr:   Bit-size of random padding in Schnorr proofs without mod-reduction.
;; secpc:   Bit-size of challenge elements.
;; logb:    Number of bits in each "chunk" in fixed-base exponentiation.
;; wsmall:  Width of simultaneous exponentiation when we have small exponents.
;; wbig:    Width of simultaneous exponentiation when we have big exponents.
;; oneexp:  The time is takes to compute one modular secp-bit exponentiation.
;; N:       Number of senders

(define (performance isbgn secp secpr secpc logb wsmall wbig oneexp N)

  ;; Displays time in minutes and hours to compute one exponentation
  ;; modulo a secp-bit integer
  (define (display-as-time noexp)
    (display (/ (truncate (round (/ (* oneexp noexp) 36))) 100)))

  ;; The cost in terms of multiplications to evaluate the bilinear map.
  (define (bmap-cost) (* 6 (* 1.5 secp)))

  ;; The cost in terms of modular exponentiations to evaluate the bilinear map.
  (define (ECC-cost) 8)

  ;; The cost of performing a fixed-base exponentation given precomputation.
  ;; The parameter is the number of bits in the exponent.
  (define (logb-fixedbase expsize)
    (let ((b (expt 2 logb)))
      (- (+ (* (/ (- b 1) (* b logb)) expsize) b) 3)))

  ;; Precomputation needed for wbig
  (define (w-simultaneous-wbig-precomp)
    (expt 2 wbig))
```

```
  ;; The cost of wsmall-wise simultaneous exponentiation.
  ;; The parameter is the number of bits in the exponent.
  (define (w-simultaneous-small expsize)
    (/ (- (+ (* 2 expsize) (expt 2 wsmall)) 4)
       wsmall))

  ;; The cost of wsmall-wise simultaneous exponentiation.
  ;; The parameter is the number of bits in the exponent.
  (define (w-simultaneous-big-withoutprecomp expsize)
    (/ (- (* 2 expsize) 4)
       wbig))

  ;; The cost of a proof of a shuffle of lists.
  ;; This value is rather arbitrarily chosen.
  (define (proof-of-shuffle) (* 8 N secp))

  ;; The cost of the proof of a shuffle of the rows in a matrix.
  (define (matrix-reencryption-proof)
    (if (> isbgn 0)
        (+ (* N N (w-simultaneous-small secpc))
           (proof-of-shuffle))
        (+ (* 9 N N (w-simultaneous-small secpc))
           (* 9 (proof-of-shuffle)))))

  ;; The cost of a single proof of double re-encryption.
  (define (double-reencryption)
    (* secpc (+ (* 9 (logb-fixedbase (+ (* 3 secp) (* 2 secpr))))
(* 4 (logb-fixedbase (+ secp (* 2 secpr))))
                (* 2 secp 1.5 9))))

  ;; Translate the number of input multiplications to the
  ;; corresponding cost in terms of exponentiations
  (define (mults-to-exps mults)
    (round (/ mults (* 1.5 secp))))

  ;; The cost of sampling and obfuscating a shuffle.
  ;; In other words the cost of computing a random encrypted
  ;; "permutation matrix".
  (define (sample-obfuscate-exps)
    (mults-to-exps
     (if (> isbgn 0)
         (* N N (logb-fixedbase secp) (ECC-cost))
         (* (+ (* 9 N N) (* 4 N))
            (logb-fixedbase (+ secp secpr))))))

  ;; The cost of proving the correctness of a matrix.
  (define (prove-exps)
    (mults-to-exps
     (if (> isbgn 0)
         (* (matrix-reencryption-proof) (ECC-cost))
         (+ (matrix-reencryption-proof)
            (* N (double-reencryption))))))

  ;; Cost of precomputation for wbig-simultaneous exponentiation
  (define (precompute-paillier-exps)
    (mults-to-exps (/ (* N N (w-simultaneous-wbig-precomp)) wbig)))

  ;; The cost of performing homomorphic matrix multiplication.
  (define (eval-exps)
    (mults-to-exps
     (if (> isbgn 0)
         (* N N (+ (bmap-cost) 1) (ECC-cost))
         (* 9 N N (w-simultaneous-big-withoutprecomp (* 2 secp))))))

  (define (display-result)
    (newline)
    (if (> isbgn 0)
        (display "BGN: ")
        (display "PAI: "))
    (display "secp=")
    (display secp)
    (display ", secpr=")
    (display secpr)
    (display ", secpc=")
    (display secpc)
    (display ", logb=")
    (display logb)
    (display ", wsmall=")
    (display wsmall)
    (display ", wbig=")
    (display wbig)
    (display ", bgood=")
    (display (round (logb-fixedbase (* 2 secp))))
    (display ", N=")
    (display N)
    (newline)

    (display "Sample and Obfuscate ")
    (display (sample-obfuscate-exps))
    (display "   ")
    (display-as-time (sample-obfuscate-exps))
    (newline)
    (display "Prove                ")
```

```
    (display (prove-exps))
    (display "   ")
    (display-as-time (prove-exps))
    (newline)
    (cond ((= isbgn 0)
           (display "Precomp. Eval        ")
           (display (precompute-paillier-exps))
           (display "   ")
           (display-as-time (precompute-paillier-exps))
           (newline)))
    (display "Evaluate             ")
    (display (eval-exps))
    (display "   ")
    (display-as-time (eval-exps)))

  (display-result)
  '()
  )

;; Compute for both BGN and Paillier
(performance 1 1024 50 50 6 5 18 0.012 350)
(performance 0 1024 50 50 6 5 18 0.012 2000)
```