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 obfuscate a secret shuffle of ciphertexts: shuffling becomes a public operation. 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 public-key obfuscations of a decryption shuffle based on the Boneh-Goh-Nissim (BGN) cryptosystem and a re-encryption shuffle based on the Paillier cryptosystem. Both allow *efficient* distributed verifiable decryption.
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 small, yet large enough to handle a number of practical cases, 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 [9] 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. [2] first formalized obfuscation as simulatability from black-box access. Goldwasser and Tauman-Kalai [15] extended this definition to consider auxiliary inputs. Some simple programs have been successfully obfuscated [8, 26]. 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 [21] 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.

* with support from the Caltech/MIT Voting Technology Project and the Knight Foundation.

1.1 Our Contributions

We show how to obfuscate the shuffle phase of a mix-net: shuffling becomes a public operation, leaving only verifiable decryption to be performed privately. We show how any homomorphic cryptosystem can provide obfuscated mixing, though the resulting mix-net becomes inefficient. We show how special and distinct properties of the Boneh-Goh-Nissim [7] and Paillier [22] cryptosystems enable obfuscated mixing efficient enough to be practical in some settings.

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. Of course, in a mix-net setting, one cannot expect a single party to generate a complete and correct obfuscated shuffle: we describe an efficient zero-knowledge proof of the correct obfuscation of a shuffle and a protocol that allows a set of parties to jointly and robustly generate an obfuscated randomly chosen shuffle. Our shuffles require considerably more exponentiations, roughly quadratic in the number of senders instead of linear, 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.

1.2 Previous Work

Most mix-nets in the literature are based on homomorphic cryptosystems and use the re-encryption-permutation paradigm introduced by Park et al. [23] and made universally verifiable by Sako and Kilian [24]. Each mix server in turn re-encrypts and permutes the ciphertexts. The first efficient zero-knowledge shuffle proofs were given independently by Neff [20] and Furukawa and Sako [14]. Groth [17] generalized Neff's approach and improved its efficiency. A third, different, approach was given recently by Wikström [28]. 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 [27, 28]. Wikström and Groth [30] 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 [10, 25, 11, 13, 3, 17]. 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 [21]. 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. 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 *doubly encrypted* messages, $\mathcal{E}'_{pk'}(\mathcal{E}_{pk}(m_i))$, that must then be decrypted verifiably. 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. Both additive and multiplicative homomorphisms are provided in the following sense:

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

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 permutation matrix of $\pi \in \Sigma_N$. 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 and inversion.

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 $F \in \mathcal{F}_\kappa$.*

Specifics of the Model. In the original Ostrovsky and Skeith definition [21], plaintext inputs are processed by an obfuscated program into ciphertext outputs. In our setting, inputs are already encrypted. Thus, the original functionality and the obfuscator depend on the *same* public key (and possibly the secret key). The security of the obfuscation—i.e. the indistinguishability property—is then defined separately, following the pattern of Ostrovsky and Skeith.

In addition, as this is a *public-key* obfuscator, the output of the obfuscated program requires a decryption. We call the reader's attention to the *difference between the encryption layers*: though they may use the same public key, the obfuscation-related encryption and the inputs' encryption have distinct purposes.

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, $(pk, sk) \in \mathcal{G}(1^\kappa)$, $a \in \mathsf{M}_{pk}$, and define $F_a(pk, sk, x) = ax$, where $x \in \mathsf{M}_{pk}$. An obfuscated circuit for 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 (known as $\mathsf{IND\text{-}CPA}$ security for public-key cryptosystems) to our 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.*

The obfuscator in Example 1 is polynomially indistinguishable if $\mathcal{CS}$ is polynomially indistinguishable ($\mathsf{IND\text{-}CPA}$ secure.)

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 the plaintexts in sorted 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*

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

Another way to implement a mix-net is to use the re-encryption-permutation paradigm of Park et al. [23]. 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. Both types of shuffles are illustrated, in their obfuscated form, in Figure 1.

Definition 6 (Re-encryption Shuffle). *A* $\mathcal{CS}$-re-encryption shuffle*, for a homomorphic cryptosystem* $\mathcal{CS}$ *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*

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

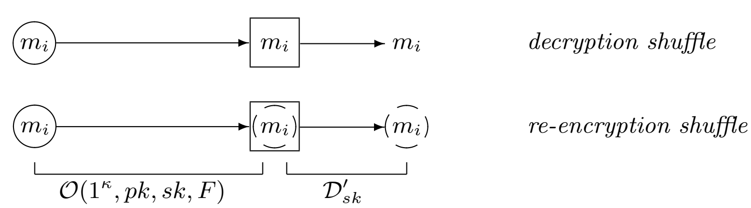

Fig. 1. The obfuscation of two types of shuffles. The circle denotes the encryption scheme under which inputs are encrypted. The square denotes the encryption scheme used by the obfuscator, which may depend on the circle encryption scheme and its keypair. The left-most inputs and right-most outputs do not include the square-layer encryption, which is only used by the public-key obfuscation process. An obfuscated decryption shuffle "swaps" one encryption scheme for the other, while an obfuscated re-encryption shuffle "layers" the two encryption schemes. The dashed circle denotes a re-encryption of the original ciphertext.

3 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. 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 the first cryptosystem:

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

3.1 The Obfuscator

Consider a permutation matrix $\Lambda^\pi = (\lambda_{ij}^\pi)$ 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., $C^\pi = (\mathcal{E}'_{pk'}(\lambda_{ij}^\pi, r_{ij}))$. Then given $d = (d_1, d_2, \ldots, d_N) \in \mathsf{C}_{pk}^N$ it is possible to perform homomorphic matrix multiplication as

$$d \star C^\pi = \left(\prod_{i=1}^{N} (c_{ij}^\pi)^{d_i} \right) \text{ giving } \mathcal{D}_{sk}(\mathcal{D}'_{sk'}(d \star C^\pi)) = (m_{\pi(i)})_{i=1}^N \ .$$

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 each list $(d_{1,l}, \ldots, d_{N,l})$ is applied to the encrypted permutation matrix as before. This gives lists $(d'_{1,l}, \ldots, d'_{N,l})$ from which the output $((d'_{1,l})_l, \ldots, (d'_{N,l})_l)$ 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: the inner, intermediate ciphertext $\mathcal{E}_{pk}(m_i)$ is exactly the input ciphertext, which means it cannot be revealed without trivially leaking the permutation. Given this constraint, we know of no efficient way, not even a cut-and-choose approach, to prove correct decryption. 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 of the same size. 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 $(q_1, q_2, \mathbb{G}_1, g, \mathbb{G}_2, e(\cdot,\cdot))$ 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 $sk = (pk, q_1)$.

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 $\log_{g^{q_1}}(c^{q_1})$.

Since 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 the uniform distributions on $\mathbb{G}_1$ and the unique order q_1 subgroup in $\mathbb{G}_1$ respectively.

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:

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

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 $c_1 \otimes c_2 \stackrel{\text{def}}{=} e(c_1, c_2)$ for ciphertexts in $\mathbb{G}_1$.

4.2 The Obfuscator

Our obfuscator is based on the fact 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 $C = (c_{ij}) = (\mathcal{E}^{\mathsf{bgn}}_{pk}(a_{ij}))$ and a $N_2 \times N_3$-matrix $C' = (d_{jk}) = (\mathcal{E}^{\mathsf{bgn}}_{pk}(b_{jk}))$, and let $A = (a_{ij})$ and $B = (b_{jk})$. Define homomorphic matrix multiplication by

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

Definition 8 (Obfuscator). *The obfuscator $\mathcal{O}^{\mathsf{bgn}}$ for the decryption shuffle $\mathcal{DS}^{\mathsf{bgn}}_N$ takes input $(1^\kappa, pk, sk, DS^{\mathsf{bgn}}_\pi)$, where $(pk, sk) \in \mathcal{G}^{\mathsf{bgn}}(1^\kappa)$ and $DS^{\mathsf{bgn}}_\pi \in \mathcal{DS}^{\mathsf{bgn}}_{N(\kappa),\kappa}$, computes $C^\pi = \mathcal{E}^{\mathsf{bgn}}_{pk}(\Lambda^\pi)$, 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.

Proposition 2. *The obfuscator $\mathcal{O}^{\mathsf{bgn}}$ for $\mathcal{DS}^{\mathsf{bgn}}_N$ is polynomially indistinguishable if the BGN cryptosystem is polynomially indistinguishable.*

Composition. Ciphertexts in $\mathbb{G}_2$ cannot be efficiently converted back into equivalent ciphertexts in $\mathbb{G}_1$. In addition, we do not know how to select groups $\mathbb{G}_1$ and $\mathbb{G}_2$ such that $\mathbb{G}_2$ exhibits a new bilinear map into a third group. Thus, the BGN-based shuffle construction we propose here is not composable: we can only mix once. In Section 7, we explain how to achieve the distributed generation of a BGN-based shuffle.

5 Obfuscating a Paillier Re-encryption Shuffle

We show how to obfuscate a re-encryption shuffle for the Paillier cryptosystem [22] by exploiting its additive homomorphism and its generalization introduced by Damgård et al. [12]. We expose a previously unnoticed homomorphic property of this generalized Paillier construction.

5.1 The Paillier Cryptosystem

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

Key Generation. On input 1^κ, $\mathcal{G}^{\mathsf{pai}}$ chooses safe κ-bit primes $p = 2p' + 1$ and $q = 2q' + 1$ randomly, defines a modulus $n = pq$, 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. [12] 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 [12] 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 well known alternative encryption algorithms. 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^{s+1}th residues, then we may define encryption of a message $m \in \mathbb{Z}_{n^s}$ as $v^m h_{s+1}^r \bmod n^{s+1}$ 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

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

This homomorphic operation is similar to the generic additive operation from Section 3, with the inner "1" replaced by an encryption of 0. As a result, though the output is also a doubly encrypted m_i, a re-encryption has occurred on the inner ciphertext. This technique extends the layered-Paillier homomorphic property first observed by Lipmaa [18].

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 $d = (d_1, \ldots, d_N) \in \mathsf{C}_{n^2}^N$ and C^π:

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

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. Again, $\mathcal{O}^{\mathsf{pai}}$ does not use sk.

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

Proposition 3. *The obfuscator $\mathcal{O}^{\mathsf{pai}}$ for $\mathcal{RS}_N^{\mathsf{pai}}$ is polynomially indistinguishable if the Paillier cryptosystem is polynomially indistinguishable.*

Composition. It may be possible to compose Paillier re-encryption shuffles using additional layers in the Damgård et. al. Paillier generalization. However, because an extra layer of encryption is added at each step, the re-encryption actions are not truly composed with one another, e.g., the second stage re-encryption acts on the first stage's obfuscation layer, while the innermost ciphertext is re-encrypted only on the first pass. Thus, in Section 7, we explain how to generate and obfuscate a Paillier re-encryption shuffle in a distributed way.

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 $((pk, C, C'), r)$ 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 doubly 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 ciphertexts have been correctly re-encrypted and permuted, a so-called "proof of shuffle."

Definition 11. *Denote by $\mathcal{R}_{rp}$ the relation consisting of pairs $((pk, d, d'), r)$ 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 [20, 14, 17, 28] 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 has 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 $d = (\prod_{i=1}^N c_{ij}^{u_i})$ and $d' = (\prod_{i=1}^N (c'_{ij})^{u_i})$.
3. They run the proof of a shuffle π_{rp} on common input (pk, d, d') and private input $\pi, r' = (\sum_{i=1}^N r_{ij} u_i)$.

Proposition 4. *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 [20, 17, 28]).

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* $((n, c, c'), (r, s))$, *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, but reuse the challenges.

Proposition 5. *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

Our two constructions can be efficiently generated in a distributed fashion. Roughly, we begin with the trivial encryption of the identity matrix. (In the Paillier case, a sub-protocol is required to generate the inner-layer encryptions of the 0-diagonal using successive re-encryptions by the parties.) We then let each party in turn shuffle and re-encrypt the rows of this matrix. In the end, the resulting permutation matrix captures the composition of the shuffles from each party: it is as if the actions of a mix-net were captured ahead of time into an encrypted matrix, then unleashed onto the ciphertext inputs at shuffle time. The details of this process, including the proofs of correct shuffling and the UC proof of security, are provided in the full version of this paper.

8 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 [19]). 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 doubly 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. We refer the reader to the Scheme program in the full paper for details on our estimates. For practical parameters we get the estimates in Fig. 2.

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. 2. 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 [16] 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. Furthermore, all constructions are easily parallelized.

9 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.

10 Acknowledgments

We thank Ronald L. Rivest, Susan Hohenberger, Rafael Pass, Shafi Goldwasser, and Guy Rothblum for productive and insightful discussions.

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. D. Chaum. Untraceable electronic mail, return addresses and digital pseudo-nyms. *Communications of the ACM*, 24(2):84–88, 1981.
10. 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.
11. 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.
12. 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.

13. 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.
14. 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.
15. 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.
16. T. Granlund. Gnu multiple precision arithmetic library (GMP). Software available at `http://swox.com/gmp`, March 2005.
17. 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.
18. Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Information Security – ISC 2005*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer Verlag, 2005.
19. A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
20. 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.
21. R. Ostrovsky and W. E. Skeith III. Private searching on streaming data. Cryptology ePrint Archive, Report 2005/242, 2005. `http://eprint.iacr.org/`.
22. 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.
23. 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.
24. 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.
25. 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.
26. H. Wee. On obfuscating point functions. In *37th ACM Symposium on the Theory of Computing (STOC)*, pages 523–532. ACM Press, 2005.
27. 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.
28. 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 [29]).
29. 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/`.
30. 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 and Proposition 2). We only detail the first proof, since the second follows by a trivial modification. 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 $\lambda^{\pi_0}_{ij} = 1$. Note that $c^{(1-b)}$ 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. By construction $\Pr[\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS}',\mathcal{A}}(\kappa) = 1] = \Pr[\mathsf{Exp}^{\mathsf{oind}-b}_{\mathcal{DS}_N,\mathcal{CS}',\mathcal{O},\mathcal{A}}(\kappa) = 1]$.

Proof (Proposition 3). 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 $(RS^{\mathsf{pai}}_0, RS^{\mathsf{pai}}_1)$ 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 experiment returns $\mathcal{E}^{\mathsf{pai}}_{pk}(M_b)$ 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 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 [29]).

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'. We assume that the same negligible function $\epsilon(\kappa)$ bounds the failure probability in Lemma 2.

Consider a fixed common input (pk, C, C') and prover $\mathcal{P}$. Denote by γ the probability that $\mathcal{P}$ convinces $\mathcal{V}$. We assume that $\epsilon(\kappa) < \gamma/4$, i.e., the knowledge error will increase somewhat compared to the knowledge error of π_{rp}.

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.

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

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

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

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

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

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

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

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

In Paillier Case. Again $b_{ki} = 0 \bmod \eta$ for all k and i, but the order of 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

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

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 alternative Paillier encryption.

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 5). 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.