Obfuscated Ciphertext Mixing

Ben Adida* and Douglas Wikström**

Abstract. Mixnets are a type of anonymous channel composed of a handful of trustees that, each in turn, shuffle and rerandomize a batch ciphertexts. For applications that require verifiability, each trustee provides a proof of correct mixing. Though mixnets have recently been made quite efficient, they still require *secret computation* and proof generation *after* the mixing process.

We introduce and implement *Obfuscated Ciphertext Mixing*, the obfuscation of a mixnet program. Using this technique, all proofs can be performed *before* the mixing process, even before the inputs are available. In addition, the mixing program does not need to be secret: anyone can *publicly compute* the shuffle (though not the decryption). We frame this functionality in the strongest obfuscation setting proposed by Barak et. al. [4], tweaked for the public-key setting. For applications where the secrecy of the shuffle permutation is particularly important (e.g. voting), we also consider the *Distributed Obfuscation of a Mixer*, where multiple trustees cooperate to generate an obfuscated mixer program such that no single trustee knows the composed shuffle permutation.

1 Introduction

Robust Mixnets are a type of verifiable anonymous channel first introduced by Chaum [11]. In recent years, they have been the focus of much research activity [2, 3, 23, 16, 20, 24, 21, 32]. Mixnets typically perform anonymization via *private* shuffling and rerandomization of ciphertext inputs. The mixing computation is then followed by a zero-knowledge proof of correct shuffling, indicating that inputs were neither duplicated nor lost. Most modern mixnets use a cryptosystem with reencryption capability, like El Gamal [17] or Paillier [26], though some recent constructions perform rerandomization through partial decryption [32]. In all cases, the shuffle computation remains *private*, and the proof of correctness is performed *after* the inputs have been mixed.

Program obfuscation, first formalized by Barak et. al. [4], is a functionality-preserving transformation of a program's source code such that it yields no more information than black-box access to the unobfuscated program. Generalized program obfuscation, though it would be fantastically useful in practice, has been proven impossible in even the weakest of settings [4, 18]. To date, point functions are the only specific class of programs known to be obfuscatable [9, 30].

1.1 The Big Picture

We show how to obfuscate a mixnet program. More precisely, we show how to shuffle ciphertexts using *public code* that implements a secret permutation. This approach is a proof-of-concept alternative to a classic mixnet, which performs shuffling in private. Our approach is interesting because of its operational properties: all mixing is public, and all proofs of correctness can be performed *prior to having the input ciphertexts*. Our solution is less efficient than

* Computer Science and Artifical Intelligence Laboratory; Massachusetts Institute of Technology; 32 Vassar Street; Cambridge, MA 02139, USA. The author wishes to acknowledge support from the Caltech/MIT Voting Technology Project and the Knight Foundation. Email: ben@mit.edu.

** KTH Nada. Email: dog@nada.kth.se

private shuffling, as it requires $O(n^2)$ time and space. Obfuscated mixing is thus unworkable for large shuffles, but may be practical enough for small shuffles, like a precinct-level election. We anticipate this new cryptographic concept will become more practical with continued research.

1.2 Our Results: Obfuscated Ciphertext Mixing

We define and implement all the necessary components to obfuscate the functionality of a specific type of reencryption mixnet. We provide security definitions of our obfuscatable functionality and appropriate security proofs of our implementation.

Deterministic Mixing. We define $\mathcal{F}_{DM}$, a *deterministic mixing* black-box functionality, which is a slightly constrained version of a typical mixnet. Specifically, $\mathcal{F}_{DM}$ has the following properties:

- the shuffling permutation π is fixed before any inputs are provided,
- mixing can be performed multiple times on different inputs,
- a given sequence of inputs will always yield the same outputs.

We define *chosen-permutation security* for deterministic mixing, where an adversary cannot distinguish which of two permutations a particular deterministic mixer implements. This closely mirrors previous security definitions for mixnets [31].

Obfuscation in the Public-Key Setting. We tweak the definition of obfuscation [4] for the public-key setting. Once the obfuscation distinguisher Δ is chosen, a key pair (SK, PK) is generated. The obfuscation function $\mathcal{O}$, the adversary $\mathcal{A}$, the simulator $\mathcal{S}$, all have access to the public key PK. None have access to the private key SK. All other details remain the same as obfuscation in the plain setting. We note that this setup maps fairly naturally to the real world: a program can be obfuscated using a public key, and its obfuscated version remains functional, though unintelligible without the secret key.

Extending a Public-Key Cryptosystem. We define the notion of a *public-key cryptosystem with obfuscatable ciphertext mixing*, which provides, in the public-key setting, an obfuscation function for $\mathcal{F}_{DM}$. Note, again, that, just like a mixnet, the obfuscation function does *not* require the cryptosystem's private key, only its public key. More formally, obfuscated mixing is defined as an extension of a public-key cryptosystem. In addition to the generation $\mathcal{G}$, encryption $\mathcal{E}$, and decryption $\mathcal{D}$ algorithms, the additional mixer obfuscation algorithm $\mathcal{O}$ is provided as follows:

$\mathcal{O}(\pi, PK)$:
on inputs π, a N-permutation, and PK, the public key generated by $\mathcal{G}$, outputs an obfuscated program $\tilde{\mathcal{P}}_\pi$ such that $\tilde{\mathcal{P}}_\pi(c_0, \ldots, c_{N-1})$ outputs $c'_0, \ldots, c'_{N-1}$ such that, $\forall i, \mathcal{D}_{SK}(c'_{\pi(i)}) = \mathcal{D}_{SK}(c_i)$.

We show the impact of obfuscation on the security of the underlying public-key cryptosystem: if $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ implements a chosen-permutation-secure $\mathcal{F}_{DM}$, then $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ cannot be CCA2-secure, but must be at least semantically secure.

Verifiable Obfuscated Mixing. We also define the notion of *verifiable obfuscated mixing*. In this setting, we define a protocol where a party $\mathcal{P}$, given π, PK, and the randomness used in computing $\tilde{\mathcal{P}}_\pi = \mathcal{O}(\pi, PK)$, can prove to a verifier $\mathcal{V}$ the correctness of $\tilde{\mathcal{P}}_\pi$ as an obfuscated mixing function. This notion of verifiable obfuscation, which has not been presented before, is necessary here because the outputs of $\tilde{\mathcal{P}}_\pi$ are ciphertexts, meaning that $\mathcal{V}$ cannot check the correctness of $\tilde{\mathcal{P}}_\pi$ on his own.

Implementation of One-Time Obfuscated Mixing. Using a cryptosystem with additive and one-time multiplicative homomorphisms – specifically Boneh, Goh and Nissim's recent cryptosystem (BGN) [8]–, we present a one-time obfuscated mixing scheme on ciphertexts of short messages, secure against chosen-permutation attacks and according to our obfuscation definition. The obfuscation function $\mathcal{O}$ is implemented as the element-wise encryption of the matrix representation of the desired permutation. The obfuscated mixing function $\tilde{\mathcal{P}}_\pi$ simply consists of a homomorphic matrix multiplication – made possible by the additive and one-time multiplicative homomorphisms – of this encrypted permutation matrix by the vector of input ciphertexts.

We give an efficient, honest-verifier zero-knowledge protocol for proving the correctness of the obfuscated mixing function. Briefly, the verifer $\mathcal{V}$ uses the obfuscated program to shuffle a vector of randomly selected plaintexts. $\mathcal{P}$ must then prove the correctness of this one random shuffle instance. We call this the *vector-challenge proof*, which is based in large part on concepts introduced by Neff [23].

Distributed Obfuscation of Mixing Function. We consider the distributed obfuscation of $\mathcal{F}_{DM}$. A group of mutually distrusting parties can jointly generate an instance of $\mathcal{F}_{DM}$ such that no subset smaller than the entire group can distinguish between two obfuscated permutations.

We show how to perform distributed mixer obfuscation using our homomorphic matrix multiplication scheme. The idea is to pass the rows of an identity matrix through a classic reencryption mixnet. The mixing actions of all mix servers are effectively encapsulated into the resulting encrypted permutation matrix. The resulting encrypted matrix is the obfuscated mixing program $\tilde{\mathcal{P}}_\pi$ where π is the composition of all the mix servers' permutations. The verification of each mix server can be performed by a variant of the *vector-challenge proof* sketched above.

Application to Voting. The matrix construction requires $O(n^2)$ space and time. Thus, it is not as efficient as modern mixnets, and its practical application is therefore limited. However, the constant factors are such that this scheme is a reasonable proof-of-concept for certain applications such as small elections. We estimate that, for an election with 2,000 voters and 32-byte ballots (enough text for about two races), the obfuscated mixing function would take approximately 5 days to prepare and 11 days to execute on an average single-CPU PC. Note also that the size of an obfuscated mixing program is pre-set, which requires the use of "filler votes" in most real-life elections.

The advantage of obfuscated mixing is the interesting process it enables. All mixing proofs can be pre-computed in the weeks leading up to an election, before anyone has voted. The mixing process itself becomes entirely deterministic. Most importantly, the mixing code is public, thus allowing an election precinct to operate its own anonymization service without

any private computation: there are no secrets to divulge [1]. Obfuscated mixing has the potential to greatly simplify election-day tasks.

1.3 Related Work

In voting schemes where only an aggregate tally of well-formed inputs is required — i.e. the candidates are pre-determined and only the candidate totals matter — alternatives to mixnets have been proposed using homomorphic encryption schemes, initially by Benaloh [7, 6]. These homomorphic schemes have seen significant advances in recent years, including various techniques to handle multiple races and multiple candidates per race [13, 29, 12, 14, 5, 20]. Homomorphic tallying is similar to obfuscated mixing 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.

Recent work on streaming-data search using homomorphic encryption [25] defined the notion of *public-key program obfuscation*. In Ostrovsky et al.'s definition, an obfuscated program is run on plaintext inputs and provides ciphertext outputs: the operator is able to tell that the program is searching the data, but cannot determine what keywords it is searching. This work is conceptually quite similar to our notion of obfuscated mixing, with one notable difference: the mixing inputs are ciphertexts while the keyword-search inputs are plaintexts. This indicates that the techniques of Ostrovsky et al. cannot be immediately applied to the obfuscated mixing problem, and that obfuscated mixing requires different security definitions. As such, this paper uses a different term: *program obfuscation in the public-key setting*.

One may also notice a similarity between obfuscated mixing and oblivious transfer. We show in section 3.5 how any obfuscated mixing scheme can easily be turned into an oblivious transfer scheme, and note how the reverse is unlikely.

1.4 Outline.

In section 2, we review mixnets, program obfuscation and the BGN cryptosystem. In section 3, we formally define the black-box deterministic mixing functionality, the notion of chosen-permutation security, and the properties of a public-key cryptosystem with obfuscatable ciphertext mixing. In section 4, we build a one-time obfuscated mix using the BGN cryptosystem (or any similarly homomorphic encryption scheme), and in section 5 we show how to generate one in a distributed fashion. In section 6, we briefly discuss how this work applies to electronic voting, before concluding with some open problems in section 7.

2 Preliminaries

2.1 Mixnets

First introduced by Chaum [11], a *mixnet* is a type of anonymous channel that shuffles and rerandomizes ciphertext inputs such that the input-output permutation is unknown to an outside observer. *Verifiable mixnets* offer, in addition, a *proof of correct mixing* to guarantee that no input was duplicated or lost. In general, mixnets are composed of a sequence of mutually distrustful mix servers $\mathcal{M}_j$, each of which shuffles and randomizes the inputs in turn.

[1] We note, again, that threshold decryption of the mixed ballots remains a private operation. This paper does not address decryption; it focuses on making the anonymization process much more public.

Many modern mixnets perform randomization by *reencryption* of inputs in a rerandomizable scheme like El Gamal [17], though some constructions, including very recent ones, perform rerandomization through partial decryption [32, 10].

2.2 Program Obfuscation

Informally, program obfuscation transforms a program such that the resulting source code preserves the original functionality but provides no information beyond what black-box access to the functionality naturally yields. The strongest definition of Barak et. al. [4], the one we consider here, requires computational indistinguishability between an adversary with source code access and a simulator with black-box access.

$\mathbb{P}_n$ refers to a class of programs on input of size n, and $\mathcal{P}$ refers to a single such program.

Definition 1. *An algorithm $\mathcal{O}$ is an obfuscator for a class of programs $\{\mathbb{P}_n\}_{n\in\mathbb{N}}$ if:*

1. **Functionality***: $\mathcal{O}(\mathcal{P})$ computes the same function as $\mathcal{P}$.*
2. **Polynomial Blowup:** *There is a polynomial $l(\cdot)$ such that, for every $\mathcal{P}$ in $\mathbb{P}$, $|\mathcal{O}(\mathcal{P})| \leq l(|\mathcal{P}|)$*
3. **Virtual Black-Box Indistinguishability:** *for all non-uniform PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ and a negligible function $\nu(\cdot)$ such that, for every $n \in \mathbb{N}$, for every $\mathcal{P} \in \mathbb{P}_n$, for every PPT distinguisher Δ,*

$$Pr[\Delta(\mathcal{A}(\mathcal{O}(\mathcal{P}))) = 1] - Pr[\Delta(\mathcal{S}^{\mathcal{P}}(1^n)) = 1] < \nu(n).$$

2.3 BGN Cryptosystem

The BGN Cryptosystem [8] provides additive and one-time multiplicative homomorphic operations. It operates in groups $\mathbb{G}_1$ and $\mathbb{G}_2$, both of order $n = q_1q_2$, where q_1 and q_2 are prime. (We use multiplicative notation in both $\mathbb{G}_1$ and $\mathbb{G}_2$.) $\mathbb{G}_1$ and $\mathbb{G}_2$ exhibit a polynomial-time computable bilinear map $e(\cdot,\cdot)$ such that (1) g generates $\mathbb{G}_1$, (2) $e(g,g)$ generates $\mathbb{G}_2$, and (3) $\forall u, v \in \mathbb{G}_1$ and $\forall a, b \in \mathbb{Z}, e(u^a, v^b) = e(u,v)^{ab}$.

Boneh et al. formulate the *Subgroup Decision Assumption*, which states that no *PPT* $\mathcal{A}$ can distinguish between a uniform distribution on $\mathbb{G}_1$ and a uniform distribution on a subgroup of $\mathbb{G}_1$ of order q_1. The cryptosystem defines the following operations:

- $\mathcal{G}(1^k)$: generate $(q_1, q_2, \mathbb{G}_1, \mathbb{G}_2, e(\cdot,\cdot))$. Let $n = q_1q_2$. Pick generators $g, u \xleftarrow{R} \mathbb{G}_1$. Set $h = u^{q_2}$. Output $PK = (n, \mathbb{G}_1, \mathbb{G}_2, e(\cdot,\cdot), g, h)$ and $SK = q_1$.
- $\mathcal{E}_{PK}(m)$: pick $r \xleftarrow{R} \mathbb{Z}_n^*$ and produce $c = g^m h^r \in \mathbb{G}_1$.
- $\mathcal{D}_{SK}(c)$: given a secret key SK and a ciphertext c, compute $c^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m \in \mathbb{G}_1$. Taking the discrete log of c^{q_1} base g^{q_1} yields m. m must be at most polynomial in k.

Similar operations $\mathcal{E}', \mathcal{D}'$ work in $\mathbb{G}_2$, with $G = e(g,g)$ and $H = e(g,h)$.

Homomorphisms. This cryptosystem offers an additive homomorphism. With $c_1 = \mathcal{E}_{PK}(m_1, r_1)$ and $c_2 = \mathcal{E}_{PK}(m_2, r_2)$, then $c_1 \cdot c_2 = \mathcal{E}_{PK}(m_1 + m_2, r_1 + r_2)$. The additive homomorphism gives a reencryption algorithm $\mathcal{RE}$. It also offers a single multiplicative homomorphism operation: $e(c_1, c_2) = \mathcal{E}_{PK}(m_1m_2, \tilde{r})$, with $\tilde{r}$ a function of m_1, m_2, r_1, r_2. Once the ciphertext is in $\mathbb{G}_2$, only additive homomorphic operations are possible.

Semantic Security. Boneh et. al. prove *semantic security* [19] of this cryptosystem under the subgroup decision assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$.

3 Definitions

We formally define the black-box functionality for *deterministic ciphertext mixing*, as well as the associated concept of chosen-permutation security. We then consider a tweak to the obfuscation definition: *obfuscation in the public-key setting.* Based on these two concepts, we define *public-key encryption with obfuscatable mixing.*

3.1 Deterministic Ciphertext Mixing

Informally, a deterministic ciphertext mixer is a black-box that can repeatedly mix ciphertexts on demand. Its permutation and any required randomness are fixed before mixing begins. Thus, if the same set of inputs is mixed twice by the same mixer, the exact same ciphertext outputs are obtained. However, no information about the permutation should be gleaned from any amount of mixing (without the secret key SK, of course). We now formalize this concept.

Definition 2. *A family of black-box functionalities $\mathcal{F}_{DM}$ for a semantically-secure cryptosystem $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ with reencryption capability $\mathcal{RE}$ is said to perform* secure deterministic ciphertext mixing *if, given $(SK, PK) \leftarrow \mathcal{G}(1^k)$, a N-permutation π, and random tape r, $\mathcal{F}_{DM}[PK, \pi, r]$ denotes an instance of a black-box functionality with the following properties:*

1. **Correctness:** *on input ciphertexts $(c_0, c_1, \ldots, c_{N-1})$, $\mathcal{F}_{DM}[PK, \pi, r]$ outputs ciphertexts $(c'_0, c'_1, \ldots, c'_{N-1})$, such that, $\forall i \in [0, N-1], \mathcal{D}_{SK}(c'_{\pi(i)}) = \mathcal{D}_{SK}(c_i)$.*
2. **Determinism:** *there is a fixed sequence of polynomial-time computable functions $(\mathsf{f}_0, \mathsf{f}_1, \ldots, \mathsf{f}_{N-1})$, such that, for every invocation of $\mathcal{F}_{DM}[PK, \pi, r]$, given $\mathcal{R}$ the randomness extracted from random tape r at the time of black-box instantiation, $c_{\pi(i)} = \mathcal{RE}(PK, c_i, \mathsf{f}_i(\mathcal{R}, c_0, c_1, \ldots, c_{N-1}))$.* [2]
3. **Chosen-Permutation Security:** *there exists a negligible function $\nu(\cdot)$ such that, for any non-uniform PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, random tape r:*

$$Pr\big[(SK, PK) \xleftarrow{R} \mathcal{G}(1^k); (\pi_0, \pi_1) \xleftarrow{R} \mathcal{A}_1(PK)\ ; \\ b \xleftarrow{R} \{0,1\}; \mathcal{A}_2^{\mathcal{F}_{DM}[PK, \pi_b, r]}(PK) = b\ \big] < \frac{1}{2} + \nu(k).$$

3.2 Obfuscation in the Public-Key Setting

Before we can consider obfuscating the functionality just described, we need a tweaked definition of obfuscation that accounts for the public-key setting. Informally, the obfuscator function should have the public key, as should the adversary, simulator, and distinguisher. The key pair is generated after the distinguisher has been selected. We do not attempt to obfuscate the program from a distinguisher with access to the private key (a fair, real-world assumption, much like the generic public-key setting.)

[2] In particular, this property implies that, for any given sequence of inputs $\{c_i\}$, $\mathcal{F}_{DM}[PK, \pi, r](\{c_i\})$ yields the exact same ciphertext outputs $\{c'_i\}$ on multiple invocations.

Definition 3. *The tuple of PPT algorithms* $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ *provides* Program Obfuscation in the Public-Key Setting *with security parameter* k *for a class of programs* $\mathbb{P} = \{\mathbb{P}_n\}, n = poly(k)$*, if:*

1. $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ *form a classic public-key encryption scheme [19]:*
   - $\mathcal{G}$*: on input* 1^k*, produces a pair* (PK, SK)*.*
   - $\mathcal{E}$*: on input* PK*, message* $m \in \{0,1\}^k$*, produces* $c \in \{0,1\}^*$
   - $\mathcal{D}$*: for all* $(SK, PK) \xleftarrow{R} \mathcal{G}(1^k)$ *and* $m \in \{0,1\}^k$*,* $\mathcal{D}_{SK}(\mathcal{E}_{PK}(m)) = m$*.*
2. **Functionality***: Given* $(SK, PK) \leftarrow \mathcal{G}(1^k)$ *and a program* $\mathcal{P} \in \mathbb{P}_n$*,* $\mathcal{O}(PK, \mathcal{P})$ *computes the same function as* $\mathcal{P}$*.* [3]
3. **Polynomial Blowup:** *There is a polynomial* $l(\cdot)$ *such that, given* $(SK, PK) \leftarrow \mathcal{G}(1^k)$*, for every* $\mathcal{P}$ *in* $\mathbb{P}$*,* $|\mathcal{O}(\mathcal{P}, PK)| \leq l(|\mathcal{P}|)$*.*
4. **Virtual Black-Box Indistinguishability:** *for all non-uniform PPT adversaries* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S}$ *and a negligible function* $\nu(\cdot)$ *such that, for every* n*, for every* $\mathcal{P} \in \mathbb{P}_n$*, for every PPT distinguisher* Δ*, for every* $(SK, PK) \xleftarrow{R} \mathcal{G}(1^k)$*:*

$$Pr[\Delta(PK, \mathcal{A}(\mathcal{O}(\mathcal{P}, PK))) = 1] - Pr[\Delta(PK, \mathcal{S}^{\mathcal{P}}(PK)) = 1] < \nu(k).$$

   Note, again, that the distinguisher Δ *has access to* PK*, but not* SK*.*

3.3 Obfuscatable Ciphertext Mixing

With the black-box functionality family $\mathcal{F}_{DM}$ and the concept of program obfuscation in the public-key setting in place, we can now simply define obfuscated ciphertext mixing.

Definition 4. *A tuple of PPT algorithms* $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ *is a* public-key cryptosystem with obfuscatable ciphertext mixing *if it provides program obfuscation in the public-key setting for a class of programs* $\mathbb{P}$ *that implements deterministic mixing functionality* $\mathcal{F}_{DM}$*.*

In addition, we say that $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ is *secure against chosen-permutation attack* if its program class $\mathbb{P}$ implements a deterministic functionality also secure against chosen-permutation attack.

3.4 Consequences to Public-Key Security

The very existence of the mixing obfuscation functionality in a public-key cryptosystem has an impact on the security requirements and limitations of the underlying cryptosystem $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. Intuitively, this underlying cryptosystem must be semantically secure, and cannot be CCA2 secure. We prove this in detail in Appendix A.1.

3.5 Reduction to Two-Round 1-out-of-l Oblivious Transfer

An obfuscatable mixing scheme is reducible to a two-round, 1-out-of-l Oblivious Transfer scheme [27]. We show this in Appendix A.3.

[3] Some definitions allow the obfuscated program to fail with negligible probability. We're interested in this stronger definition – stronger from the point of view of a positive result.

4 Implementation of a One-Time Obfuscated Mix

We are now ready to present our construction of a public-key cryptosystem with one-time obfuscatable mixing functionality. We call this scheme **the Matrix Mixer**, for reasons that will become immediately clear. This construction is based on any semantically-secure cryptosystem with additive and one-time homomorphic properties. Though BGN [8] is the only such cryptosystem known, our construction remains generic, so that it may apply to future cryptosystems.

First, we define a deterministic mixing scheme based on homomorphic matrix multiplication and prove that it is secure against chosen-permutation attacks. Then, we show that this homomorphic matrix multiplication can be easily obfuscated, as the encrypted matrix is both functional and permutation-hiding.

4.1 Matrix-Based Deterministic Mixing

Homomorphic Matrix Multiplication. Consider matrices $A = [a_{ij}]$, of dimensions m-by-h, and $B = [b_{jk}]$, of dimensions h-by-n. Consider the matrix $C = A \times B = [c_{ik}]$ of resulting dimensions m-by-n. We know that:

$$\forall i \in [0, m-1], \forall k \in [0, n-1] : c_{ik} = \sum_{j=0}^{h-1} a_{ij} b_{jk}.$$

Now consider the notation $\tilde{M} = \mathcal{E}_{PK}(M)$ to indicate the element-wise encryption of a matrix, such that $\tilde{M} = [\mathcal{E}_{PK}(m_{ij})]$. Note how the matrix multiplication operation above requires a number of additions, but only one multiplication per input matrix element. Thus, using a cryptosystem like BGN with homomorphic addition $\oplus$ and one-time homomorphic multiplication $\otimes$, one can compute homomorphic matrix multiplication on element-wise encryptions of matrices. Consider matrices $\tilde{A}$ and $\tilde{B}$, the element-wise encryptions of matrices A and B as above, respectively. We can homomorphically compute $\tilde{C} = [\mathcal{E}_{PK}(c_{ik})]$ as follows:

$$\forall i \in [0, m-1], \forall k \in [0, n-1] : \tilde{c}_{ik} = \bigoplus_{j=0}^{h-1} \tilde{a}_{ij} \otimes \tilde{b}_{jk}.$$

Thus, a cryptosystem with additive and one-time homomorphic properties – e.g. BGN – allows for one homomorphic matrix multiplication.

Deterministic Mixing by Matrix Multiplication. We construct $\mathcal{F}_{DM}$, a family of black-box mixing functionalities, using this homomorphic matrix multiplication. Consider a homomorphic cryptosystem like BGN, represented by the tuple $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. Given $(SK, PK) \xleftarrow{R} \mathcal{G}(1^k)$, π a N-permutation, and r a random tape, the family $\mathcal{F}_{DM}$ is defined as:

$\mathcal{F}_{DM}[PK, \pi, r]$:

1. π is encoded as its corresponding permutation matrix $\mathbb{M}_\pi$.
2. $\tilde{\mathbb{M}}_\pi$, the element-wise encryption of $\mathbb{M}_\pi$, is generated using randomization factors $(r_0, r_1, \ldots, r_{N^2-1})$ obtained from the random tape r.
3. Repeat the mixing as often as queried:
   on inputs $\mathsf{C} = (c_0, c_1, \ldots, c_{N-1})$, output $\mathsf{C}' = \mathsf{C} \otimes \mathbb{M}_\pi$.

Simple inspection shows that $\mathcal{F}_{DM}$ provides correct and deterministic mixing functionality.

4.2 Chosen-Permutation Security

We now show that instances of the deterministic functionality family $\mathcal{F}_{DM}$ defined above are secure against chosen-permutation security. For modularity's sake, the proof begins with a lemma addressing matrix-level semantic security (which will be useful in the next section).

Lemma 1. *Given a semantically-secure public-key cryptosystem $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ with additive and one-time multiplicative homomorphism properties , for all non-uniform PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:*

$$Pr\big[(SK, PK) \xleftarrow{R} \mathcal{G}(1^k); (\pi_0, \pi_1) \xleftarrow{R} \mathcal{A}_1(PK) \ ;$$
$$b \xleftarrow{R} \{0,1\}; \tilde{\mathbb{M}}_{\pi_b} \xleftarrow{R} \mathcal{E}_{PK}(\mathbb{M}_{\pi_b}); \mathcal{A}_2(\tilde{\mathbb{M}}_{\pi_b}) = b \ \big] < \frac{1}{2} + \nu(k).$$

where $\mathcal{E}_{PK}(\mathbb{M}_\pi)$ denotes the element-wise encryption of $\mathbb{M}_\pi$.

Proof. This matrix-level semantic security follows from a generic hybrid argument, given that an element-wise encrypted matrix is just a sequence of ciphertexts. We give a better, tight reduction in Appendix A.2.

Given this lemma, it is clear that the black-box matrix functionality is chosen-permutation secure: oracle-access to a computation that the adversary could perform directly with the encrypted matrix certainly cannot yield any more information than the encrypted matrix itself.

Theorem 1. *The Matrix Mixer deterministic mixing functionality is secure against chosen-permutation attack under the assumption that the underlying cryptosystem is semantically secure.*

Proof. Assume the Matrix Mixer is not secure against chosen-permutation attack. Then there exists an adversary $\mathcal{A}$ which can distinguish between $\mathcal{F}_{DM}[PK, \pi_0, r]$ and $\mathcal{F}_{DM}[PK, \pi_1, r']$. Using $\mathcal{A}$, we construct $\mathcal{A}'$, a new adversary which breaks Lemma 1.

$\mathcal{A}'(PK)$:

1. Run $\mathcal{A}$ to produce π_0 and π_1, and output them to the challenger.
2. Receive $\tilde{\mathbb{M}} = \mathcal{E}_{PK}(\mathbb{M}_{\pi_b})$ for $b \in \{0,1\}$, from the challenger.
3. Using $\tilde{\mathbb{M}}$, simulate an instance of $\mathcal{F}_{DM}$ for $\mathcal{A}$. Note how this uses exactly the same code as a known-permutation $\mathcal{F}_{DM}$, which means the simulation is perfect. Note also that $\mathcal{A}$ can interact with this simulated functionality as much as it wants.
4. On guess b' from $\mathcal{A}$, return the same guess b' to the challenger.

If $\mathcal{A}$ succeeds – i.e. $b = b'$ – with better than negligible probability, then so does $\mathcal{A}'$.

4.3 Obfuscating the Matrix Mixer

We now have a well-defined black-box functionality family $\mathcal{F}_{DM}$, secure against chosen-permutation attack. As it turns out, obfuscation of an instance of $\mathcal{F}_{DM}$ is easy: simply publish the encrypted permutation matrix. Anyone can compute the homomorphic matrix multiplication, and semantic security enables easy simulation. More formally, given a BGN-like cryptosystem $(\mathcal{G}, \mathcal{E}, \mathcal{D})$, consider the mixing obfuscation function:

$\mathcal{O}(PK, \pi)$:

1. Compute $\tilde{\mathbb{M}}_\pi = \mathcal{E}_{PK}(\mathbb{M}_\pi)$.
2. Output the program $\mathcal{P}_{\tilde{\pi}}$ with source code:
   (a) on input $\mathsf{C} = (c_0, c_1, \ldots, c_{N-1})$, each a ciphertext in $(\mathcal{G}, \mathcal{E}, \mathcal{D})$,
   (b) output the result of $\mathsf{C}' = \mathsf{C} \otimes \tilde{\mathbb{M}}_\pi$.

Theorem 2. *The homomorphic matrix-mixer obfuscator $\mathcal{O}$ is a secure obfuscator of deterministic mixing functionality $\mathcal{F}_{DM}$ in the public-key setting, assuming the underlying $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is semantically secure.*

Proof. We construct a simulator $\mathcal{S}$ for program obfuscation in the public-key setting, such that, if Δ succeeds at distinguishing $\mathcal{A}$ and $\mathcal{S}$, a new adversary $\mathcal{A}'$, built upon Δ and $\mathcal{A}$, can break Lemma 1 and thus the underlying semantic security of the cryptosystem. Note that, in the case of obfuscated mixing, the obfuscated program $\mathcal{O}(\mathcal{P}, PK)$ fed to $\mathcal{A}$ is effectively $\tilde{\mathbb{M}}_\pi$, an element-wise encryption of a permutation matrix.

$\mathcal{S}^{\mathcal{F}_{DM}[PK,\pi,r],\mathcal{A}}(PK)$:

1. Generate a random permutation π' and its associated $\tilde{\mathbb{M}}_{\pi'}$.
2. run $\mathcal{A}(PK, \tilde{\mathbb{M}}_{\pi'})$, and output to Δ the exact output of $\mathcal{A}$.

If Δ distinguishes between $\mathcal{S}$ and $\mathcal{A}$, then $\mathcal{A}'$ can instantiate two $\mathcal{A}$'s, feed an encrypted matrix to each one, and use the output from Δ on the outputs of both $\mathcal{A}$ to distinguish between the encrypted matrices. Thus, by contradiction, Δ has a negligible property of distinguishing between $\mathcal{S}$ and $\mathcal{A}$, and $\mathcal{O}$ is a secure obfuscation function for deterministic mixing.

4.4 Verifiable Obfuscated Matrix Mixing

In a number of applications, a party may want some proof that $\mathcal{P}$ is indeed performing as specified. This feature is particularly important in the public-key setting, where the program's operator cannot observe correct operation simply by looking at the program's output. This proof should be, of course, zero-knowledge, such that a program operator is guaranteed nothing more than that $\mathcal{P}$ performs *some* permutation on the inputs.

In our implementation of deterministic mixing, this implies proving that $\tilde{\mathbb{M}}_\pi$ is an element-wise encryption of some permutation matrix. Though there are tedious ways of proving this correct form directly using proofs of partial knowledge and the cryptosystem's homomorphic properties, there is a significantly more efficient approach that tests the encrypted matrix's functionality: the verifier can challenge the prover with a random vector of inputs. Since the plaintexts of the random challenge vector are known, the matrix multiplication can be performed using homomorphic addition and repeated squaring. The prover can then use a classic Neff/Groth proof of shuffle of known plaintexts. Intuitively, if the encrypted matrix is anything but a permutation matrix, it is extremely unlikely that it will behave as a permutation matrix on a random input.

More formally, consider (SK, PK) the key pair for the public-key setting, π the obfuscated permutation, $\tilde{\mathbb{M}}_\pi$ the element-wise encrypted matrix, and $\mathcal{R}$ the randomness used in creating the encrypted matrix. The proof protocol between $\mathcal{P}(PK, \pi, \mathcal{R}, \tilde{\mathbb{M}}_\pi)$ and $\mathcal{V}(PK, \tilde{\mathbb{M}}_\pi)$ is as follows:

VectorProof:

1. $\mathcal{V}$ challenges $\mathcal{P}$ with a random vector of plaintexts $[m_i]$.

2. Both $\mathcal{V}$ and $\mathcal{P}$ compute $[c_i] = [m_i] \otimes \tilde{\mathbb{M}}_\pi$.
3. $\mathcal{P}$ and $\mathcal{V}$ perform a HVZK proof of mixing from $[m_i]$ to $[c_i]$.

This proof protocol is based on known techniques due to Neff [23]. The matrix multiplication with a plaintext vector, proof of known-plaintext shuffle, and proof that only a permutation matrix can reliably permute a random challenge vector are detailed in Appendix B.

Theorem 3. VectorProof *is an Honest Verifier Zero-Knowledge Proof that the encrypted matrix* $\tilde{\mathbb{M}}$ *encodes a permutation matrix.*

Proof Sketch. We prove the usual properties of HVZK protocols:

- *Completeness:* an honest $\mathcal{P}$ clearly knows π and the encryption exponents of the matrix. As described in detail in Appendix B, this gives $\mathcal{P}$ the reencryption factors from $[m_i]$ to $[c_i]$, which lets $\mathcal{P}$ perform the Neff/Groth HVZK proof of known plaintext shuffle.
- *Soundness:* assuming the mixing proof is sound, one is left to verify that a non-permutation matrix multiplied by a random input vector will yield some permutation of that vector for only a negligible fraction of random challenge vectors. This is detailed in Appendix B.
- *HVZK:* $\mathcal{P}$ provides no information beyond its execution of the HVZK protocol of correct mixing. Thus, the mixing proof simulator can simulate this higher-level transcript.
- *Note on PoK:* if the underlying proof of mixing is a proof of knowledge, then VectorProof also becomes a proof of knowledge, as the underlying extractor can be used directly to extract the permutation.

Note that the random message-vector challenge can be composed of short messages (50 bits or so). This significantly speeds up the modular exponentiations required to perform the proof while maintaining the same security properties.

4.5 Performance

Consider N the number of ciphertext inputs to a permutation we wish to obfuscate. The obfuscation function $\mathcal{O}$ requires $O(N^2)$ space and time to produce $\mathcal{P}_{\tilde{\pi}}$, and $\mathcal{P}_{\tilde{\pi}}$ requires $O(N^2)$ space and time to operate.

5 Distributed Obfuscation and Encapsulated Mixing

So far, we have shown how a single party can obfuscate and prove the correctness of a deterministic mixing function. However, for many applications, e.g. voting, single-party obfuscation is insufficient: that single party will then know the shuffling permutation, breaking the anonymity of the overall protocol. Unfortunately, we show in Appendix C that generic, composable mixing is unlikely. Thus, we develop a method for the efficient, distributed obfuscation of a deterministic matrix mixer. We consider the most straight-forward case, where t trustees generate $\mathcal{P}_{\tilde{\pi}}$ in such a way that *all* t *trustees* must collude in order to determine π.

In our approach, the trustees mix the rows of an encrypted permutation matrix using a typical mixnet. Instead of proving correct mixing of every row – which would require $O(N^2)$ time for each trustee proof –, we propose a proof using the same input-vector challenge approach as that used for single-party verifiable obfuscation.

5.1 Distributed Obfuscation of a Matrix Mixer

Cryptosystems like BGN clearly provide reencryption capability through the homomorphic addition of an encryption of 0. Thus, the distributed obfuscation of a matrix mixer proceeds as follows:

1. $\mathbb{M}_{ID}$, the identity N-permutation-matrix, is encrypted with randomization 0 and used as initial input $\tilde{\mathbb{M}}^{(0)}$.
2. Each trustee T_l performs reencryption and shuffling on the previous trustee's output $\tilde{\mathbb{M}}^{(l)}$:
   (a) generate secret N-permutation π_l,
   (b) generate N^2 reencryption factors r_{ij},
   (c) output a matrix whose elements are $\mathbb{M}_{ij}^{(l+1)} = \mathcal{RE}(\mathbb{M}_{i\pi_l(j)}^{(l)}, r_{ij})$.
3. The final output is $\tilde{\mathbb{M}}_\pi$ where π is the composition of all π_l.

5.2 Verifying the Distributed Obfuscation of a Mixer

The challenge-vector approach to obfuscation verifiability is not immediately applicable to the distributed obfuscation of a mixer, because no single trustee knows the overall permutation. Instead, each trustee must prove that its *modification* to the matrix is a permutation. This can be done with a variation of the vector-challenge proof.

The following proof protocol occurs after the t trustees have remixed the encrypted matrix, publicly producing $\tilde{\mathbb{M}}^{(1)}, \ldots, \tilde{\mathbb{M}}^{(t)}$ along the way.

MultiVectorProof:

1. $\mathcal{V}$ interacts with each trustee T_l in turn:
   (a) $\mathcal{V}$ produces a random challenge vector $[m_i^{(l)}]$.
   (b) $\mathcal{V}$ and T_l each compute $\mathsf{C}_{premix}^{(l)} = [m_i^{(l)}] \otimes \tilde{\mathbb{M}}^{(l-1)}$
   (c) $\mathcal{V}$ and T_l each compute $\mathsf{C}_{postmix}^{(l)} = [m_i^{(l)}] \otimes \tilde{\mathbb{M}}^{(l)}$
   (d) $\mathcal{V}$ and T_l engage in a HVZK proof of correct mixing from $\mathsf{C}_{premix}^{(l)}$ to $\mathsf{C}_{postmix}^{(l)}$, using the Neff/Groth techniques that apply to a shuffle of homomorphic encryptions of unknown plaintexts (Appendix B).
2. If a given trustee succeeds in proving correct mixing of its random challenge vector, then, using the same argument as for VectorProof, that particular trustee indeed permuted and reencrypted the rows of the encrypted matrix it received as input. If all trustees succeed, then $\tilde{\mathbb{M}}^{(t)}$ is a verified encrypted permutation matrix that contains π, the composition of all trustees' secret permutations π_l.

This protocol is clearly complete and honest-verifier zero-knowledge, since, once again, each trustee provides no information apart from that of the included sub-protocol for proving a correct random vector mix. Soundness is guaranteed for each trustee by the same proof as that of VectorProof. Again, if the underlying proof of mixing is turned into a proof of knowledge of the permutation, then this protocol also becomes a proof of knowledge of each trustee T_l's permutation π_l.

5.3 Encapsulated Mixing

At a high level, the protocols described here effectively capture the shuffling actions of the trustees into the resulting encrypted permutation matrix, with all verification performed before any mixing of real inputs has occurred. Upon using this encrypted permutation on real ciphertext inputs, the shuffling actions of the trustees are effectively replayed in a deterministic way. We call this process *encapsulated mixing*.

6 Application to Voting

6.1 Comparison to Existing Systems

Universally-verifiable voting currently employs two major types of anonymous routing and tabulation: mixnets and homomorphic encryption. Mixnets offer free-form ballots, though their operation requires significant private computation and a time-taking proof of correctness. Homomorphic schemes are more limited in their ballot form, but offer a much faster, more public operation.

In general, the most significant problem with homomorphic schemes is that, oftentimes, election regulation requires the availability of individual, anonymized ballots at recount time. Mixnets provide these individual ballots, but require significantly more secret computation on election day. (Note that joint-decryption remains a secret computation in all cases.)

Obfuscatable Mixing may provide a best-of-both-worlds solution: the mixnet source code becomes public with all proofs performed prior to election day. Anonymization on election day requires no secret computation (though decryption still does). In addition, ballots can be free-form, and all plaintext ballots are available at the end of the election, instead of a single total count.

6.2 Long Messages & Variable-Size Mixing

The current BGN-based scheme accepts plaintext of limited length, because a discrete logarithm is required at decryption time. For ballots longer than a few bytes, it may be necessary to break up the ciphertext into a few pieces, each no longer than a few bytes. Each ciphertext chunk will have to go through the deterministic mixer program in the same way.

Note also that mixnets can adapt to a variable mixing size, while an obfuscated mixing program has a pre-set size. It is necessary to compute all the outputs of an obfuscated mixer, even if some of the inputs are missing. Any filler input value can be used for the empty inputs, along with proofs that these are filler votes. When the ballots are eventually decrypted, the special filler values can be removed from the tally.

6.3 Proof of Anonymity in the Universally-Verifiable Setting

We have *not* proven that obfuscatable mixing remains permutation-hiding when used in the universally-verifiable setting. In such a setting, the output ciphertexts are eventually decrypted by a set of trustees, and their corresponding plaintexts published for the world to see, along with zero-knowledge proofs of correct decryption. If an adversary is given this one-time decryption oracle, potentially on multiple sets of non-adaptively chosen ciphertexts as per Section 6.2, do the obfuscation and chosen-permutation security properties survive?

Intuitively, we believe they do: a decryption simulator could fake decryption of the outputs according to some random, probably incorrect, permutation. If the adversary were able to distinguish between this fake decryption and a real one, then the semantic security of the scheme would be broken. If the adversary were unable to distinguish and could still successfully extract permutation information using fake decryption, then so could an adversary without the decryption Oracle, once again breaking semantic security. Note how this simulated decryption would not withstand multiple adaptive mixes.

For lack of space, and because we believe obfuscated mixing in the universally-verifiable setting deserves its own, in-depth analysis and proper definitions, we leave the detailed proof of this intuition to a later date.

6.4 Performance

Detailed analysis in Appendix D concludes that an election with 2,000 eligible voters, 32-byte ballots (enough for two races), and 5 trustees will require 5 days to prepare, and approximately 11 days to mix on a single CPU, without any parallelization. Note that both preparation and mixing are easily parallelizable with small numbers of CPUs, meaning a quad-CPU machine could prepare the election in 36 hours and mix it in just under 3 days.

This makes obfuscated mixing an interesting proof of concept, though not a system that's voting-ready just yet. However, these numbers are not altogether outrageous: there's reason to believe that further research and continued parallelizations may yield perfectly acceptable running times.

7 Conclusion

We introduced *obfuscated ciphertext mixing*, a new cryptographic construction that uses homomorphic properties to enable the obfuscation of a mixnet program. With this constrution, verifiable shuffling can be proven *before* the inputs are available, using *public code*. In devising a distributed obfuscation mechanism, we also defined *encapsulated mixing*, where a set of parties combine their mixing actions into an encrypted permutation, ready to be applied to ciphertexts at a later date.

Future Work. Much work remains. It would be interesting to devise an obfuscated mixing scheme on ciphertexts of full-length plaintexts. It will also be useful to explore partially composable obfuscated mixing, where a ciphertext may be mixable a small number of times. This could enable direct mixnet-like usage by a small number of mututally distrusting parties, without a tedious distributed generation process. Additionally, this proof-of-concept idea needs to be integrated into a universally-verifiable voting protocol. It should be considered, in its current form, only a proof-of-concept.

On a separate front, the ability to obfuscate a function as complex as mixing is particularly interesting. Certainly, the public-key setting is responsible for enabling such interesting obfuscation. What other functions might be obfuscatable in this way? What other reasonable definitions of obfuscation in the public-key setting might yield interesting applications?

References

1. Miracl cryptographic library. `http://indigo.ie/~mscott`.

2. Masayuki Abe. Mix-networks on permutation networks. In *ASIACRYPT '99: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 258–273, London, UK, 1999. Springer-Verlag.
3. Masayuki Abe and Fumitaka Hoshino. Remarks on mix-network based on permutation networks. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 317–324, London, UK, 2001. Springer-Verlag.
4. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahay, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Crypto '01*, pages 1–18, 2001. LNCS No. 2139.
5. Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 274–283, New York, NY, USA, 2001. ACM Press.
6. J. Benaloh and M. Yung. Distributing the power of government to enhance the power of voters. pages 52–62. ACM, 1986.
7. J. D. Cohen (Benaloh) and M. J. Fischer. A robust and verifiable cryptographically secure election scheme. pages 372–382, Portland, 1985.
8. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In Joe Killian, editor, *Proceedings of Theory of Cryptography Conference 2005*, volume 3378 of *LNCS*, pages 325–342. Springer, 2005.
9. Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1997.
10. David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.
11. David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.
12. Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Lecture Notes in Computer Science*, 1233:103–??, 1997.
13. Ronald J.F. Cramer, Matthew Franklin, L. A.M. Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.
14. Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*, pages 90–104, London, UK, 2001. Springer-Verlag.
15. Edward Fredkin and Tommaso Toffoli. Conservative Logic. pages 47–81, 2002.
16. Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 368–387, London, UK, 2001. Springer-Verlag.
17. T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.
18. Shafi Goldwasser and Yael Tauman Kalai. On the (im)possibility of obfuscating programs. In *Proc. 46th FOCS*, pages 553–562. IEEE, 2005.
19. Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 365–377, New York, NY, USA, 1982. ACM Press.
20. Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In *PKC '02: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, pages 145–160, London, UK, 2002. Springer-Verlag.
21. Colin Boyd Kun Peng and Ed Dawson. Simple and Efficient Shuffling with Provable Correctness and ZK Privacy. In *CRYPTO 2005*, 2005.
22. Helger Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. Cryptology ePrint Archive, Report 2004/063, 2004. `http://eprint.iacr.org/`.
23. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125, New York, NY, USA, 2001. ACM Press.
24. Lan Nguyen, Rei Safavi-Naini, and Kaoru Kurosawa. Verifiable shuffles: A formal model and a paillier-based efficient construction with provable security. Cryptology ePrint Archive, Report 2005/199, 2005. `http://eprint.iacr.org/`.

25. Rafail Ostrovsky and William E. Skeith III. Private Searching On Streaming Data. Cryptology ePrint Archive, Report 2005/242, 2005. http://eprint.iacr.org/.
26. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt 1999*, pages 223–238. Springer-Verlag. LNCS 1592.
27. M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.
28. Hugo Krawczyk Ran Canetti and Jesper Buus Nielsen. Relaxing Chosen-Ciphertext Security. In *CRYPTO 2003*, pages 565–582, 2003.
29. Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 148–164, London, UK, 1999. Springer-Verlag.
30. Hoeteck Wee. On obfuscating point functions. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 523–532. ACM, 2005.
31. Douglas Wikström. A universally composable mix-net. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 317–335. Springer, 2004.
32. Douglas Wikstrom. A sender verifiable mix-net and a new proof of a shuffle. Cryptology ePrint Archive, Report 2005/137, 2005. http://eprint.iacr.org/.

A Proofs

In this section, we detail some of the less-essential proofs of this work.

A.1 Obfuscated Mixing Effects on Public-Key Security

Note that, given our definition of obfuscatable mixing that requires $\mathcal{O}$ to successfully obfuscate a permutation of any size, the following theorems could be easily proven with permutations of specific sizes (i.e. sizes 2 and 1, respectively). We give more complete proofs, however, so that these theorems follow even with a weaker definition of mixing obfuscation where $\mathcal{O}$ accepts permutations of only certain sizes.

Theorem 4. *Given a public-key cryptosystem with obfuscatable ciphertext mixing* $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ *secure against chosen-permutation attack, the public-key cryptosystem* $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ *must be semantically secure.*

Proof. Assume $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is not semantically secure. We now show that $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ cannot be secure against chosen-permutation attack. If $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is not semantically secure, there exists a PPT adversary $\mathcal{A}$ that selects messages m_0 and m_1 such that it can distinguish between encryptions of one or the other. We can then construct a new adversary $\mathcal{A}'$ that breaks chosen-permutation security:

1. when $\mathcal{A}'$ is prompted, it output (π_0, π_1) such that $\pi_0(0) = 0$ and $\pi_1(0) \neq 0$.
2. $\mathcal{A}'$ receives $\tilde{\mathcal{P}}_\pi$ from the Chosen-Permutation Challenger
3. $\mathcal{A}'$ prompts $\mathcal{A}$ for m_0 and m_1.
4. $\mathcal{A}'$ prepares a vector of N ciphertexts $(c_0, c_1, \ldots, c_{N-1})$ such that $\mathcal{D}_{SK}(c_0) = m_0$ and $\forall i \in [1, N-1], \mathcal{D}_{SK}(c_i) = m_1$. $\mathcal{A}'$ then computes $(c'_0, c'_1, \ldots, c'_{N-1}) = \tilde{\mathcal{P}}_\pi(c_0, c_1, \ldots, c_{N-1})$.
5. $\mathcal{A}'$ passes c'_0 to $\mathcal{A}$
6. $\mathcal{A}$ responds with bit b that indicates its guess that $\mathcal{D}_{SK}(c'_0) = m_b$.
7. $\mathcal{A}'$ outputs the same bit b, effectively guessing permutation π_b.

$\mathcal{A}'$ succeeds in breaking chosen-permutation security of $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ with the same probability as $\mathcal{A}$ succeeds in breaking semantic security of $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. Thus, by contradiction, if $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ is chosen-permutation secure, then $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ must be semantically secure.

Theorem 5. *Given a public-key cryptosystem with obfuscatable mixing* $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$ *secure against chosen-permutation attack, the public-key cryptosystem* $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ *is not CCA2-secure.*

Proof. It is well understood that a public-key cryptosystem with reencryption capability cannot be CCA2-secure, since an adversary with decryption-oracle access can simply reencrypt the challenge ciphertext, query the decryption oracle with this new ciphertext, and obtain the challenge decryption. We show that an obfuscated mixing program for a permutation of any size N trivially provides this reencryption capability.

$\mathcal{RE}(PK, c)$:

1. select a random permutation π of size N and compute $\tilde{\mathcal{P}}_\pi \xleftarrow{R} \mathcal{O}(\pi, PK)$.
2. generate ciphertexts $c_1, c_2, \ldots, c_{N-1}$ for random plaintexts.
3. compute $(c'_0, c'_1, \ldots, c'_{N-1}) = \tilde{\mathcal{P}}_\pi(c, c_1, \ldots, c_{N-1})$.
4. noting that π is known, output $c'_{\pi(0)}$.

Given that $\mathcal{O}$ is, by hypothesis, secure against chosen-permutation attack, the probability that $c'_{\pi(0)} = c_0$ is negligible, otherwise the permutation would easily be distinguishable. Thus, $\mathcal{RE}$ provides reencryption with overwhelming probability, and $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ cannot be CCA2-secure.

Note that the above proof does not rule out a RCCA-secure cryptosystem [28].

A.2 Semantic Security of an Encrypted Permutation Matrix

Assume there exists such an aversary $\mathcal{A}$. Using $\mathcal{A}$, one can construct $\mathcal{A}'$, a PPT adversary that breaks the semantic security of $(\mathcal{G}, \mathcal{E}, \mathcal{D})$:

$\mathcal{A}'(PK)$:

1. Select chosen plaintexts $m_0 = 0$ and $m_1 = 1$, output them to the semantic-security challenger, and collect $c = \mathcal{E}_{PK}(m_b)$ from the challenger, for $b = 0$ or $b = 1$.
2. Homomorphically compute $\bar{c}$, the encryption of $(1 - m_b)$ (assuming multiplicative notation for the homomorphic operation, $\bar{c} = \frac{\mathcal{E}_{PK}(1)}{c}$.)
3. run $\mathcal{A}$ to produce π_0 and π_1, the two permutations $\mathcal{A}$ will attempt to distinguish. These two permutations are encoded as permutation matrices $\mathbb{M}_{\pi_0}$ and $\mathbb{M}_{\pi_1}$.
4. Compose an encrypted matrix $\tilde{\mathbb{M}}'$ as follows: (a) for all elements that $\mathbb{M}_{\pi_0}$ and $\mathbb{M}_{\pi_1}$ have in common, simply include an encryption of that common element, and (b) for all elements where $\mathbb{M}_{\pi_0}$ and $\mathbb{M}_{\pi_1}$ differ, use c if the element in $\mathbb{M}_{\pi_0}$ is 0, and $\bar{c}$ if the element in $\mathbb{M}_{\pi_0}$ is 1. Note that, as a result of this manipulation, if c is the encryption of 0, then $\tilde{\mathbb{M}}'$ is the encryption of $\mathbb{M}_{\pi_0}$, but if c is the encryption of 1, then $\tilde{\mathbb{M}}'$ is the encryption of $\mathbb{M}_{\pi_1}$.
5. Provide $\tilde{\mathbb{M}}'$ to $\mathcal{A}$, and obtain guess b' in return.
6. Return b' to the SSC, effectively indicating a guess that $b = b'$.

If $\mathcal{A}$ succeeds – i.e. $b = b'$ – with better than negligible probability, then so does $\mathcal{A}'$. Note that this is a tight reduction: $\mathcal{A}'$ succeeds with the same probability as $\mathcal{A}$.

A.3 Obfuscated Mixing Implies Two-Round 1-out-of-l Oblivious Transfer

Consider a sender $\mathcal{S}$ and a recipient $\mathcal{R}$:

1. parameter l, the number of strings available to send via oblivious transfer, is public.
2. $\mathcal{R}$ sets up an obfuscatable mixing environment $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{O})$, and computes $(SK, PK) \leftarrow \mathcal{G}(1^k)$. $\mathcal{R}$ then selects a permutation π such that $\pi(0) = j$ where j is the desired index in the 1-out-of-l OT, and computes $\tilde{\mathcal{P}}_\pi = \mathcal{O}(\pi, PK)$. $\mathcal{R}$ sends $PK, \tilde{\mathcal{P}}_\pi$ to $\mathcal{S}$.
3. $\mathcal{S}$ encrypts each value v_i as c_i with the appropriate public key received from $\mathcal{R}$, checks the validity of $\tilde{\mathcal{P}}_\pi$, and applies it to $\{c_i\}$. $\mathcal{S}$ then takes the first output and sends it to $\mathcal{R}$.
4. $\mathcal{R}$ decrypts the output received.

This protocol is clearly correct: $\mathcal{R}$ will receive an encryption of input j. The value of j is clearly secret if the mixing scheme provides chosen-permutation security. The recipient receives only the indexed value. Note that the communication efficiency of this OT scheme is linear, which is not particularly impressive given known optimal OT schemes that achieve $\log^2$ communication [22].

B Vector-Challenge Proof

Neff [23] introduced techniques for efficiently proving shuffles using vector dot-product and proof of exponent product combinatorial arguments. Groth [20] later formalized this approach using homomorphic commitments in generic homomorphic cryptosystems. We now adapt these techniques to our setting, particularly BGN [8]. We also show that an encrypted matrix that successfully shuffles a random set of inputs almost certainly encodes a permutation.

B.1 Homomorphic Matrix Multiplication with Vector of Known Plaintexts

The central idea to proving that the encrypted matrix performs a shuffle on a random challenge vector is that, given a random set of inputs whose plaintexts are known, the matrix's permutation effect can be carried out without homomorphic multiplication (in BGN, this is equivalent to "staying in $\mathbb{G}_1$"). Specifically, consider the encrypted matrix $\tilde{\mathbb{M}} = [\tilde{a}_{ij}]$. To perform homomorphic matrix multiplication with a vector of known plaintexts, one can simply invoke the multiplication by a known constant operation, which derives from the additive homomorphism property. Given that the additive homomorphism is composable, all ciphertexts remain in the same group, which enables the application of the Neff techniques with little modification.

We assume, for simple notation, that homomorphic addition is performed by ciphertext multiplication, and thus that multiplication by a known value is performed by exponentation. (This is clearly the case for BGN.) Thus, assuming a random challenge input vector $[m_j]_{j=0}^{N-1}$:

$$c_j = \prod_{i=0}^{N-1} \tilde{a}_{ij}^{m_j}$$

If $\tilde{a}_{ij} = \mathcal{E}_{PK}(a_{ij}, r_{ij})$, then the prover (the creator of the matrix) knows the set $\{r_{ij}\}$. If π is the permutation encoded by $\tilde{A}$, then in the computation above:

$$c_j = \mathcal{E}_{PK}(m_{\pi(j)}, \sum_{i=0}^{N-1} r_{ij} m_j)$$

Note that this randomization exponent is the dot product of the message vector and the j'th column of randomization exponents in the encrypted permutation matrix. Note also, of course, that the prover knowns all of these randomization factors, since it knows the set of $\{r_{ij}\}$ and the vector of known plaintexts $[m_j]$.

As the proofs below will make clear, the random message-vector challenge can be composed of short messages (50 bits or so). This significantly speeds up the modular exponentiations required to perform the proof while maintaining the same security properties.

B.2 Proof of Permutation Matrix

The important question in this proof is: if a matrix is not a permutation matrix, what is the probability that the product of that matrix with a random challenge vector is a permutation of that random challenge vector? Using Neff's arguments of vector-dot-product combinatorics, two cases arise:

1. **at least one row of the matrix is not an identity row:** the chance that the dot-product of that row with the challenge vector will give one of the input values is at most N/q, which is thus an upper bound on the fraction of challenge vectors that will succeed.
2. **all rows are identity rows, in which case at least two rows are identical and both select the j'th input:** given a fixed matrix, for each challenge vector that succeeds, there are $(q-1)$ that do not, thus at most $1/q$ of the challenge vectors succeed.

Thus, if the matrix is not a permutation matrix, at most N/q of the possible challenge vectors will succeed. Note that this bound is loose, and a tight bound would be even smaller. In any case, for any reasonable value of N and q, this is a negligible fraction of challenges.

Thus, if an encrypted matrix successfully shuffles a random challenge vector, then there is an overwhelming probability that this matrix is a permutation matrix. This holds if the challenge plaintexts are short (50 bits or so), as the probabilities of dot-product equality remain the same with such challenges.

B.3 HVZK Proof of Shuffle of Known Plaintexts

VectorProof calls for a proof that a random challenge vector is mixed properly. Using the techniques of Neff generalized by Groth to the generic homomorphic setting, it is straightforward to prove, in honest-verifier zero-knowledge, that $\{c_i\}$ is a permutation of the known inputs $\{m_i\}$. This is done by constructing two polynomials, one with roots $\{m_i\}$, another with roots the plaintexts of the outputs $\{c_i\}$, and showing that they take on the same value at a random challenge point t. This proof is performed by proof of equality of exponent products, as per the Neff technique.

B.4 Proving a Distributed Obfuscation

When multiple trustees T_l reencrypt a permutation matrix in the distributed generation of an obfuscated mix, trustees after T_1 do not know the correspondence between their inputs and the known challenge plaintexts, since they have been shuffled by the prior trustees.

However, if one denotes as $\tilde{\mathbb{M}}^{(l)}$ the encrypted matrix just before trustee T_l reencrypts and permutes its rows, then, given a random challenge of known plaintexts $[m_i]$, trustee T_l knows

the permutation π_l and reencryption exponents $\{r_{ij}^{(l)}\}$ that effectively shuffle $\mathsf{C}^{(l)} = [m_i] \otimes \tilde{\mathbb{M}}^{(l)}$ into $\mathsf{C}^{(l+1)} = [m_i] \otimes \tilde{\mathbb{M}}^{(l+1)}$, both of which can be computed by the verifier.

Thus, each trustee T_l can be so challenged to provide a Neff/Groth-like proof of shuffle of homomorphic encryptions between $\mathsf{C}^{(l)}$ and $\mathsf{C}^{(l+1)}$. The same technique of polynomial evaluation applies, and trivially adapting the Pedersen commitment scheme to BGN yields the homomorphic commitment scheme necessary for the protocol.

C Possibility of Composable Mixing

An immediate question arises from this work: what might it take to improve this one-time mixer such that it might become composable? We show that accomplishing such a task is no small feat: a composable mixing function might well enable universal computation on ciphertexts. The proof, detailed below, describes how to implement an encrypted Fredkin universal logic gate [15] using composable public mixing. We note that one might find wiggle room in this reduction, which depends on the ability to express an encrypted permutation as a series of normal ciphertexts within the same cryptosystem.

C.1 Universal Public Computation from Composable Public Mixing

We show how a composable public mixing scheme can be used to construct a public universal computation mechanism. Our approach is to build a Fredkin gate using public mixing. Fredkin gates are universal, thus it is sufficient to build a single Fredkin gate for our purposes.

The only assumption we make is that, in the public mixing scheme, the encrypted permutation can be expressed as a sequence of ciphertexts. In other words, an encrypted permutation can itself be fed through another encrypted permutation. This is a relatively mild assumption, but it may preclude certain public mixing schemes if the encrypted permutation is of a very different form than typical ciphertexts under the cryptosystem in question. Note that, in our example of a one-time public mix scheme, the permutations are indeed expressable as a sequence of ciphertexts.

The construction of an encrypted Fredkin gate is then fairly straight-forward. We consider a typical Fredkin gate with three inputs x, y, z and three outputs x', y', z' such that:

- $z' = z$
- if $z' = 1$, then $x' = y$ and $y' = x$
- if $z' = 0$, then $x' = x$ and $y' = y$

We then consider 2-by-2 encrypted permutations for our public-mix scheme. Note that there are only two such mixes: the identity permutation, and the flip permutation. We consider the bit-value of the flip permutation to be 1, while the bit-value of the identity permutation is 0. Thus, all circuit values are represented by encrypted permutations.

Then, the computation of a Fredkin gate output is as follows:

- $z' = z$ is simply passed through without modification
- x and y are publicly mixed using z as the encrypted matrix.

Thus, very simply, a Fredkin gate can be computed on ciphertext inputs that are specially formed. A composable Public Mix implies that any number of Fredkin gates can be computed on ciphertexts only, thus that complete ciphertext computation can be performed using a Composable Publix Mix.

D Performance in a Voting Setting

D.1 Setup

Consider an election with 2,000 eligible voters, each ballot is 32 bytes (enough for the names of two candidates), and the BGN scheme is used with a 1024-bit prime composite n. We consider a 2.5Ghz single-CPU Pentium (entry-level PC as of late 2005).

No performance numbers on the BGN scheme have been published, but one can extrapolate from elliptic curve performance numbers published with the MIRACL crypto library [1]. As MIRACL shows a 1Ghz Pentium performing an elliptic curve 256 bit exponentiation in 26ms, we estimate that a 512-bit exponentiation in $\mathbb{G}_1$ of BGN on our reference PC (2.5Ghz) will take 20ms. We also estimate that the bilinear mapping operation will be double the work of a 512-bit IBE encrypt operation, which yields performance of about 30ms for a BGN bilinear map.

D.2 Proof and Verification

Using Neff proofs, each trustee will need to perform 4,000,000 exponentiations in $\mathbb{G}_1$ with exponents up to 512 bits (one of the two prime factors of n for the randomness h^r) to prepare a mixing matrix. The time to compute the mix of the random challenge vector as well as the time to perform the Neff proof are negligible in comparison, since the prover can compute the reencryption factors with simple modular multiplications and additions, and the Neff proof is linear, not quadratic. The verifier, on the other hand, will need to perform 4,000,000 exponentiations to mix his random challenge, though these exponentiations will be much faster given the small message challenges. With 50-bit messages, the exponentiation will be about 10 times faster than full exponentiation on 500 bits, yielding an equivalence of 400,000 exponentiations.

Assuming 5 trustees, the mixer preparation, proof, and verification will require the equivalent time of 22,000,000 exponentiations with 500-bit exponents. According to our above estimations, this will take about 5 days, before any optimization or parallelization.

D.3 Mixing

The mixing process on real inputs will then take 4,000,000 bilinear maps per ciphertext chunk, of which we consider that there may be 8 (4 bytes per ciphertext), thus 32,000,000 bilinear maps. According to our above estimations, this will take approximately 11 days, again before any optimizations or parallelizations. Note that mixing is particularly parallelizable.

D.4 Conclusion on Performance

Obfuscated mixing is good enough as a proof of concept, but not quite fast enough yet for any real election. With significant parallelization (which is very much possible, especially for mixing), these performance numbers for small elections could become practical.