Verifiable Elections That Scale for Free

Melissa Chase[1], Markulf Kohlweiss[2], Anna Lysyanskaya[3], and Sarah Meiklejohn[4]

[1] Microsoft Research Redmond
melissac@microsoft.com
[2] Microsoft Research Cambridge
markulf@microsoft.com
[3] Brown University
anna@cs.brown.edu
[4] UC San Diego
smeiklej@cs.ucsd.edu

Abstract. In order to guarantee a fair and transparent voting process, electronic voting schemes must be verifiable. Most of the time, however, it is important that elections also be anonymous. The notion of a *verifiable shuffle* describes how to satisfy both properties at the same time: ballots are submitted to a public bulletin board in encrypted form, verifiably shuffled by several mix servers (thus guaranteeing anonymity), and then verifiably decrypted by an appropriate threshold decryption mechanism. To guarantee transparency, the intermediate shuffles and decryption results, together with proofs of their correctness, are posted on the bulletin board throughout this process.

In this paper, we present a verifiable shuffle and threshold decryption scheme in which, for security parameter k, L voters, M mix servers, and N decryption servers, the proof that the end tally corresponds to the original encrypted ballots is only $O(k(L + M + N))$ bits long. Previous verifiable shuffle constructions had proofs of size $O(kLM + kLN)$, which, for elections with thousands of voters, mix servers, and decryption servers, meant that verifying an election on an ordinary computer in a reasonable amount of time was out of the question.

The linchpin of each construction is a *controlled-malleable proof* (cm-NIZK), which allows each server, in turn, to take a current set of ciphertexts and a proof that the computation done by other servers has proceeded correctly so far. After shuffling or partially decrypting these ciphertexts, the server can also update the proof of correctness, obtaining as a result a *cumulative* proof that the computation is correct so far. In order to verify the end result, it is therefore sufficient to verify just the proof produced by the last server.

1 Introduction

Electronic voting is one of the most compelling applications of cryptography [3]. An approach popular in cryptographic literature is voting via a *verifiable shuffle* [22, 11, 16, 17, 19], which consists of L voters $V_1, \ldots, V_L$, M mix servers $S_1, \ldots,$

S_M (that are needed for the election to be anonymous) and N threshold decryption servers $D_1, \ldots, D_N$ (that are responsible for setting up the system and, in the end, tallying the results). This approach requires a secure rerandomizable encryption scheme, in which given the public key and a ciphertext c for some message m, one can efficiently find a random ciphertext c' for the same message m. Further, it requires that there be a threshold realization of the cryptosystem [13, 24, 6]; i.e., the secret key can be split up into "shares" such that each server can use its share to partially decrypt a ciphertext, and the correct decryption can be obtained by putting all the decryption shares together.

On a high level, once the decryption servers set up the system, a verifiable election works in the following three phases [21, 4]: first, each voter V_i submits to a public bulletin board a ciphertext $c_i^{(0)}$ containing his or her encrypted ballot (in one variation [22, 23, 2], a trusted device submits this ciphertext on the user's behalf, so that the user does not know the randomness that went into forming the encryption and thus is unable to demonstrate that he voted a certain way). Next, in the ballot processing phase, each mix server S_i in turn takes as input the set of encrypted ballots $(c_1^{(i-1)}, \ldots, c_L^{(i-1)})$ and randomizes and permutes (i.e., shuffles) them, posting to the public bulletin board the ciphertexts $(c_1^{(i)}, \ldots, c_L^{(i)})$ together with a zero-knowledge proof π_i that this was done correctly. Finally, in the tallying phase, on input $(c_1^{(M)}, \ldots, c_L^{(M)})$, each decryption server D_i publicly outputs its decryption shares $(d_1^{(i)}, \ldots, d_L^{(i)})$, together with a zero-knowledge proof π'_i that this was done correctly. The tally is now publicly computable by putting together the decryption shares for each ciphertext.

How much data does an elections monitor have to process in order to verify the tally? Suppose the monitor observes and verifies *every* step of both mixing and decrypting. This means verifying that, in the ballot processing step, the mix servers correctly formed LM ciphertexts, and then that the decryption servers correctly computed LN decryption shares. This multiplicative blow-up is very unfortunate if these algorithms are used on a large scale; indeed, the very vision of universally verifiable elections is that it should be easy for anyone, including the voters themselves, to participate in guaranteeing both the anonymity and the correctness of the election. This means that it should scale well as the number of mix and decryption servers grows. Can the work of the elections monitors be reduced to $O(k(L + M + N))$ for security parameter k?

(Note that a verifiable shuffle has the attractive property that the set of ballots output in the end is the same as the set of ballots that were encrypted and submitted to the bulletin board. In particular, this allows for write-in candidates. If an election is simply binary, then an encrypted tally can be computed if the underlying cryptosystem is additively homomorphic, and the resulting ciphertext can be decrypted by the decryption servers.)

In a recent result [7], we (referred to ask CKLM in what follows to distinguish between our current and prior work) proposed an idea for overcoming this blow-up as far as the ballot processing phase was concerned. Before, all known aggregation results [1, 15] required complex interactions between shuffling authorities and, for non-interactive verification, were based on the Fiat-Shamir [14] heuristic and thus the random oracle model. The crucial observation of CKLM is that the monitor does not need to verify every step of the shuffle: it is sufficient to just verify the last set of ciphertexts $(c_1^{(M)}, \dots, c_L^{(M)})$, as long as the proof π_M produced by the last mix server S_M attests to the fact that these were correctly computed from the *original* ballots $(c_1^{(0)}, \dots, c_L^{(0)})$. Of course, the last mix server S_M does not have the witness to this statement: it knows only the randomness it used to randomize and shuffle the ciphertexts $(c_1^{(M-1)}, \dots, c_L^{(M-1)})$. To nevertheless allow π_M to suffice for the entire shuffle, CKLM proposed a cryptographic tool, called *controlled-malleable proofs* (cm-NIZKs), that allows each server S_i to build on the proof π_{i-1} that attests to the validity of the ciphertexts $(c_1^{(i-1)}, \dots, c_L^{(i-1)})$ in order to obtain the proof π_i attesting to the validity of $(c_1^{(i)}, \dots, c_L^{(i)})$; importantly, cm-NIZKs allow π_i to be the same size as π_{i-1}. As a result, the proof π_M suffices, and the elections monitor need not verify any of the intermediate ciphertexts and proofs. CKLM then gave a construction of cm-NIZKs by taking advantage of certain convenient properties of GS proofs [20].

The CKLM result came with a significant caveat that made it almost irrelevant in practice as far as verifiable shuffles are concerned: they used permutation matrices to represent the statement that there exists a permutation and a randomization that, when applied to $(c_1^{(i-1)}, \dots, c_L^{(i-1)})$, result in $(c_1^{(i)}, \dots, c_L^{(i)})$. A permutation matrix is $L \times L$, and so, by necessity, each proof π_i was $\Theta(L^2 k)$ bits, for the security parameter k. The elections monitor would thus have to read $\Theta(k(L^2 + M))$ bits in order to verify the correctness of a shuffle, rather than $\Theta(LMk)$ bits when using, for example, the verifiable shuffle of Groth and Lu [19], which does require the monitor to check intermediate ciphertexts and proofs (hence the factor of M), but in which each proof is only of size $\Theta(Lk)$ because Groth and Lu represent a permutation as a list rather than a matrix. The CKLM solution is therefore asymptotically superior only in the case where there are more mix servers than voters. In recent follow-up work, CKLM extended their results [8] in a way that would allow permutations to be represented as lists rather than matrices, but the extension does not apply for the scenario at hand because it can only tolerate a constant number of mix servers. A natural question, therefore, is the following: Is it possible to combine the CKLM techniques with the Groth-Lu techniques to get a cm-NIZK for the correctness of a shuffle of size $\Theta(k(L + M))$? In this paper, we answer it in the affirmative, obtaining a verifiable shuffle construction in which elections monitors only read $\Theta(k(L + M))$ bits to verify that the ballot processing step was done correctly.

Next, we focus on the application of cm-NIZKs to the verification of threshold decryption (i.e., the tallying phase). In a naïve approach, each decryption server D_i, on input the ciphertexts $(c_1^{(M)}, \dots, c_L^{(M)})$, outputs the decryption shares $(d_1^{(i)}, \dots, d_L^{(i)})$ and the proofs $(\pi_1^{(i)}, \dots, \pi_L^{(i)})$ that these decryption shares were correct. It is natural to ask whether, by taking turns processing these ciphertexts and using cm-NIZK techniques, it is possible to achieve *compact* verifiable threshold decryption, in which each server builds on the decryption share and proof of the previous server to arrive, at the end, at the vector of decryptions $(m_1, \ldots, m_L)$ for the original L ciphertexts and a single vector of proofs $(\pi_1^{(N)}, \ldots, \pi_L^{(N)})$ that attests to the correct decryption and requires $\Theta(k(L+N))$ bits to verify. In this paper we answer this question in the affirmative as well. Rather than have each decryption server produce its own decryption share and proof of correctness, we instead have the decryption servers pass around a single *cumulative* share, along with a malleable proof of correctness. When one authority receives the share and proof from the previous authority, it can therefore fold in its own share, and update, or "maul", the proof to obtain a new proof of correctness that takes into account this new share.

To the best of our knowledge, the question of compact verifiable threshold decryption has not been previously considered: the standard approach in threshold cryptography [13, 24, 6] is that, on input the ciphertext and a share of the secret key, each decryption server computes a share of the decryption and a proof that this share was computed correctly. These shares are then publicly output, and the decryption can be computed; one can verify that the decryption is correct by verifying the proofs. In a t-out-of-N threshold cryptosystem, $t+1$ correct shares are sufficient, while no malicious coalition of t servers can break the security of the cryptosystem or cause incorrect decryption. An advantage of this approach is that no communication need be required between servers; in the public bulletin board model of electronic voting, however, this is not as important as compact verification. Our approach, instead, has the decryption servers communicate via the public bulletin board. Each server, in turn, takes as input the cumulative decryptions and their proofs of correctness so far (if any), carries out its share of the decryption, and outputs the resulting cumulative decryption shares and the resulting cumulative cm-NIZK proof of their correctness. The overall process results in the correct decryption if no server fails to produce a valid proof.

2 Definitions and Notation

In this section, we present building blocks and definitions for our voting scheme. First, we recall the malleable proof system due to CKLM [7] used by both our shuffle and threshold decryption constructions. Then, we give the CKLM definition of a verifiable shuffle, which takes into account that one proof is used to prove correctness of the entire shuffle. Next, we give a new definition, analogous to the definition for the shuffle, of compact threshold encryption; here, the malleable proof is used to prove correct partial decryption. Finally, in order to show that these two notions fit together, we present a simple definition of a secure voting scheme.

2.1 Controlled malleable proofs (cm-NIZKs)

As defined by CKLM, a controlled malleable proof for a relation R and transformation class $\mathcal{T}$ consists of four algorithms $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$: $\mathsf{CRSSetup}$ generates a common reference string crs, the prover $\mathcal{P}$ takes as input the crs, the instance x, and a witness w for the truth of the statement $(x, w) \in R$ and outputs a proof π, and the verifier $\mathcal{V}$ takes as input the crs, an instance x, and a proof π and either accepts or rejects the proof.

These three algorithms constitute a regular non-interactive proof (which we define formally in the full version of the paper [9]); such a proof is further called *zero knowledge* (NIZK) if there exists a PPT simulator (S_1, S_2) such that an adversary can't distinguish between proofs formed by the prover and proofs formed by the simulator, and a *proof of knowledge* (NIZKPoK) if there exists a PPT extractor (E_1, E_2) that can produce a valid witness from any accepting proof.

The fourth algorithm, specific to malleable proof systems, is ZKEval, which, on input crs, a transformation $T = (T_{\text{inst}}, T_{\text{wit}})$ (in some transformation class $\mathcal{T}$), an instance x, and a proof π, outputs a mauled proof π' for instance $T_{\text{inst}}(x)$. The main definition of CKLM for controlled malleable proofs then reconciles malleability with extractability (specifically, simulation-sound extractability [12, 18]) and requires that, for any instance x, if an adversary can produce a valid proof π that $x \in L_R$ then an extractor can extract from π either a witness w such that $(x, w) \in R$ or a previously proved instance x' and transformation $T \in \mathcal{T}$ such that $x = T_{\text{inst}}(x')$. Intuitively this guarantees that any proof that the adversary produces is either generated from scratch using a valid witness, or formed by applying a transformation from the class $\mathcal{T}$ to an existing proof. They define this formally as follows:

Definition 2.1. [7] *Let* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *be a NIZKPoK system for an efficient relation* R*, with a simulator* (S_1, S_2) *and an extractor* (E_1, E_2)*. Let* $\mathcal{T}$ *be an allowable set of unary transformations for the relation* R *such that membership in* $\mathcal{T}$ *is efficiently testable. Let* SE_1 *be an algorithm that, on input* 1^k*, outputs* $(\mathsf{crs}, \tau_s, \tau_e)$ *such that* (crs, τ_s) *is distributed identically to the output of* S_1*. Let* $\mathcal{A}$ *be given, and consider the following game:*

- *Step 1.* $(\mathsf{crs}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)$.
- *Step 2.* $(x, \pi) \xleftarrow{\$} \mathcal{A}^{S_2(\mathsf{crs}, \tau_s, \cdot)}(\mathsf{crs}, \tau_e)$.
- *Step 3.* $(w, x', T) \leftarrow E_2(\mathsf{crs}, \tau_e, x, \pi)$.

The proof system satisfies controlled-malleable simulation-sound extractability *(CM-SSE, for short) with respect to* $\mathcal{T}$ *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that the probability (over the choices of* SE_1*,* $\mathcal{A}$*, and* S_2*) that* $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ *and* $(x, \pi) \notin Q$ *(where* Q *is the set of queried statements and their responses) but either (1)* $w \neq \perp$ *and* $(x, w) \notin R$*; (2)* $(x', T) \neq (\perp, \perp)$ *and either* $x' \notin Q_x$ *(the set of queried instances),* $x \neq T_{inst}(x')$*, or* $T \notin \mathcal{T}$*; or (3)* $(w, x', T) = (\perp, \perp, \perp)$ *is at most* $\nu(k)$*.*

In addition, CKLM define the notion of *strong derivation privacy* for such proofs, in which simulated proofs are indistinguishable from those formed via transformation. This is defined formally as follows:

Definition 2.2. [7] *For a malleable NIZK* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *with an associated simulator* (S_1, S_2)*, a given adversary* $\mathcal{A}$*, and a bit* b*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

- *Step 1.* $(\sigma_{sim}, \tau_s) \xleftarrow{\$} S_1(1^k)$.
- *Step 2.* $(\mathsf{state}, x_1, \pi_1, \ldots, x_q, \pi_q, T) \xleftarrow{\$} \mathcal{A}(\sigma_{sim}, \tau_s)$.
- *Step 3. If* $\mathcal{V}(\sigma_{sim}, x_i, \pi_i) = 0$ *for some* i*,* $(x_1, \ldots, x_q)$ *is not in the domain of* T_{inst}*, or* $T \notin \mathcal{T}$*, abort and output* $\perp$*. Otherwise, form*

$$\pi \xleftarrow{\$} \begin{cases} S_2(\sigma_{sim}, \tau_s, T_{inst}(x_1, \ldots, x_q)) & \text{if } b = 0 \\ \mathsf{ZKEval}(\sigma_{sim}, T, \{x_i, \pi_i\}_i) & \text{if } b = 1. \end{cases}$$

- *Step 4.* $b' \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \pi)$.

The proof system is strongly derivation private *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

Putting these two definitions together, if a proof system is CM-SSE, strongly derivation private, and zero knowledge, then CKLM call it a cm-NIZK.

2.2 Compactly verifiable shuffles

In a compact verifiable shuffle, as defined by CKLM, a single (malleable) proof is used to prove the correctness of an entire multi-step shuffle. Formally, a compact verifiable shuffle $(\mathsf{Setup}, \mathsf{ShuffleKg}, \mathsf{Shuffle}, \mathsf{Verify})$ is parameterized by a re-randomizable encryption scheme $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{Dec})$: Setup generates parameters *params*; $\mathsf{ShuffleKg}$ outputs key pairs (pk_j, sk_j) chosen from a hard relation R_{pk} that are used by mix servers as a stamp of participation; $\mathsf{Shuffle}$ takes the original ciphertexts $\{c_i, \pi_i\}_i$, the shuffled ciphertexts $\{c_i'\}_i$ and proof thus far $(\pi, \{pk_j\}_j)$, and a pair of keys $(pk_m, sk_m) \in R_{pk}$, and outputs $(\{c_i''\}_i, \pi', \{pk_j\}_j \cup \{pk_m\})$; and Verify ensures that the shuffle has been performed correctly.

Before giving the compact verifiability definition, we recall the relation that is proved by the shuffle. Instances are of the form $(pk, \{c_i\}_i, \{c_i'\}_i, \{pk_j\}_j)$, where pk is a public key produced by EncKg, $\{c_i\}_i$ are ciphertexts produced by Enc through the voting process, $\{c_i'\}_i$ are the shuffled ciphertexts, and $\{pk_j\}_j$ are the public keys for R_{pk} that are used to identify the mix servers that have participated in the shuffle thus far. Witnesses are of the form $(\varphi, \{R_i\}_i, \{sk_j\}_j)$, where φ is a permutation, $\{R_i\}_i$ are re-randomization factors, and $\{sk_j\}_j$ are the secret keys for the mix servers. Then the relation R is defined by

$$\begin{aligned} &((pk, \{c_i\}_i, \{c_i'\}_i, \{pk_j\}_j), (\varphi, \{R_i\}_i, \{sk_j\}_j)) \in R \\ \Leftrightarrow &\{c_i'\}_i = \{\mathsf{ReRand}(pk, \varphi(c_i); R_i)\}_i \wedge \forall j (pk_j, sk_j) \in R_{pk}. \end{aligned}$$

Definition 2.3. [7] *Let* $(\mathsf{Setup}, \mathsf{ShuffleKg}, \mathsf{Shuffle}, \mathsf{Verify})$ *be a verifiable shuffle with respect to an encryption scheme* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{Dec})$*. For an adversary* $\mathcal{A}$ *and a bit* $b \in \{0, 1\}$*, let* $p_b^{\mathcal{A}}(k)$ *be the probability that* $b' = 0$ *in the following experiment:*

- *Step 1. params* $\xleftarrow{\$}$ $\mathsf{Setup}(1^k)$*,* $(pk, sk) \xleftarrow{\$} \mathsf{EncKg}(params)$*, and* $(T = \{pk_i\}_i, \{sk_i\}_i) \xleftarrow{\$} \mathsf{ShuffleKg}(1^k)$*.*
- *Step 2.* $\mathcal{A}$ *gets params, pk, T, and access to the following two oracles: an initial shuffle oracle that, on input* $(\{c_i, \pi_i\}_i, pk_\ell)$ *for* $pk_\ell \in T$*, outputs* $(\{c'_i\}_i, \pi, \{pk_\ell\}_\ell)$ *(if all the proofs of knowledge* π_i *verify), where* π *is a proof that the* $\{c'_i\}_i$ *constitute a valid shuffle of the* $\{c_i\}_i$ *performed by the user corresponding to* pk_ℓ *(i.e., the user who knows* sk_ℓ*); and a shuffle oracle that, on input* $(\{c_i, \pi_i\}_i, \{c'_i\}_i, \pi, \{pk_j\}_j, pk_m)$ *for* $pk_m \in T$*, outputs* $(\{c''_i\}_i, \pi', \{pk_j\}_j \cup \{pk_m\})$*.*
- *Step 3. Eventually,* $\mathcal{A}$ *outputs a tuple* $(\{c_i, \pi_i\}_i, \{c'_i\}_i, \pi, T' = \{pk_j\}_j)$*.*
- *Step 4. If* $\mathsf{Verify}(params, (\{c_i, \pi_i\}_i, \{c'_i\}_i, \pi, \{pk_j\}_j)) = 1$ *and* $T \cap T' \neq \emptyset$ *then continue; otherwise simply abort and output* $\perp$*. If* $b = 0$ *give* $\mathcal{A}$ $\{\mathsf{Dec}(sk, c'_i)\}_i$*, and if* $b = 1$ *then give* $\mathcal{A}$ $\varphi(\{\mathsf{Dec}(sk, c_i)\}_i)$*, where* φ *is a random permutation.*
- *Step 5.* $\mathcal{A}$ *outputs a guess bit* b'*.*

Then the shuffle is compactly verifiable *if for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$*.*

In addition to defining such a shuffle, CKLM also provide a generic construction using a hard relation [10], a proof of knowledge, and a cm-NIZK. Since we use this generic construction as a template for our shuffle construction in Section 3, for completeness we provide an outline of it in the full version of the paper.

2.3 Threshold encryption

As discussed in the introduction, the previous model for threshold encryption had each participant generate a share and proof of correctness separately; the proofs of correctness would then be verified separately, and the shares would all be combined at the end to produce the decrypted ciphertext. As we now assume that the participants compute a single share and proof cumulatively (by computing their own shares and then folding them into a single one that gets passed around and mauling the accompanying proof appropriately), the model must be changed to reflect these differences.

With this in mind, we define a threshold encryption scheme to be a tuple of four algorithms ($\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify}$). The first, EncKg, generates a public encryption key pk, a verification key vk that is used to check the validity of a share, and a set of secret key shares $\{sk_i\}_i$. The next, Enc, performs regular public-key encryption. The next, $\mathsf{ShareDec}$, takes in a share sk_i of the secret key, a ciphertext c, and the decryption share/proof thus far. It first computes its own partial decryption of c, and then folds this value into the cumulative share and outputs this new share; it also mauls the proof to take into account that the value it has folded in is correct, and thus the new share is the correct cumulative share for the participants thus far. Finally, $\mathsf{ShareVerify}$ takes in the cumulative share and proof and verifies that the share is indeed correct. In this paper we focus on n-out-of-n threshold decryption, in which all n parties must participate in the decryption; our results should generalize to the t-out-of-n case as well, but we leave that as an open problem.

There are a number of desirable properties of a threshold encryption scheme. Functionally, we require completeness, which says that if everyone is behaving honestly then the scheme works as it should; i.e., the proofs of correctness verify and the ciphertexts decrypt correctly. Completeness therefore requires that the threshold encryption scheme also yields a regular encryption scheme: the Dec algorithm would take as input $sk := \{sk_j\}_j$ and compute the cumulative shares; it would then output the final cumulative share, which by completeness is equal to the message m. This essentially means $\mathsf{Dec}(sk, c) = \mathsf{ShareDec}(pk, vk, sk, c, (\bot, \bot, \bot))$.

In terms of security properties, we would first like our scheme to satisfy IND-CPA security; to capture this, we can use the usual IND-CPA security experiment, in which an adversary $\mathcal{A}$ outputs message (m_0, m_1) such that $|m_0| = |m_1|$ and is asked to guess which one of them a challenge ciphertext c^* encrypts. In addition to IND-CPA security, in the threshold setting we would also like to guarantee that partial decryption shares do not reveal anything about the secret key shares, even in the face of malicious participants (which also means that these malicious participants should not be able to recover the message without a sufficient number of collaborators). To capture this requirement, which we call *share simulatability*, we have the following definition:

Definition 2.4. *Let* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify})$ *be a threshold encryption scheme with* N *decryption participants. For an adversary* $\mathcal{A}$ *and a bit* b*, let* $p_b^{\mathcal{A}}(k)$ *be the probability of the event that* $b' = 0$ *in the following game:*

– *Step 1.* $\{1, \ldots, N\} \supset S \xleftarrow{\$} \mathcal{A}(1^k, N)$.
– *Step 2.* $(pk, vk, \{sk_i\}_i) \xleftarrow{\$} K(1^k, N, S)$.
– *Step 3.* $b' \xleftarrow{\$} \mathcal{A}^{SD}(pk, vk, \{sk_i\}_{i \in S})$,

where (K, SD) *are defined as* $(\mathsf{EncKg}, \mathsf{ShareDec})$ *if* $b = 0$ *and the following algorithms if* $b = 1$*:*

Procedure $K(1^k, n, S)$	*Procedure* $SD(t := (i, c, s, I, \pi))$
$(pk, vk', \{sk'_j\}_{j=1}^N) \xleftarrow{\$} \mathsf{EncKg}(1^k, N)$	$m \leftarrow \mathsf{Dec}(\{sk_j\}_{j=1}^N, c)$
$(vk, \{sk_j\}_{j \in S}, \tau) \xleftarrow{\$} \mathsf{SimKg}(pk, vk', N, S)$	$(s', \pi') \xleftarrow{\$} \mathsf{SimShareDec}(pk, vk, \tau, t, m)$
output $(pk, vk, \{sk_j\}_{j \in S} \cup \{sk'_j\}_{j \in [N] \setminus S})$	*output* $(s', I \cup \{i\}, \pi')$

Then the threshold encryption scheme is share simulatable *if there exist PPT algorithms* SimKg *and* $\mathsf{SimShareDec}$ *as used above such that for all PPT algorithms* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $|p_0^{\mathcal{A}}(k) - p_1^{\mathcal{A}}(k)| < \nu(k)$.

As $\mathsf{SimShareDec}$ can therefore simulate the decryption process without access to the secret key, we can argue that the shares produced by $\mathsf{ShareDec}$ do not reveal anything more than what an honest decryption would reveal. Finally, we require that the proof of correctness is meaningful; i.e., that it is hard for an adversary to produce a ciphertext c, a message m and an accepting proof π such that $m \neq \mathsf{Dec}(sk, c)$. More formally:

Definition 2.5. *Let* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify})$ *be a threshold encryption scheme with N decryption participants. For an adversary $\mathcal{A}$, define the following game:*

– *Step 1.* $(pk, vk, \{sk_i\}_i) \overset{\$}{\leftarrow} \mathsf{EncKg}(1^k, N)$.
– *Step 2.* $(c, m, \pi) \overset{\$}{\leftarrow} \mathcal{A}(pk, vk, \{sk_i\}_i)$,

Then the threshold encryption scheme is sound *if for all PPT algorithms $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that the probability that* $\mathsf{ShareVerify}(pk, vk, c, (m, [N], \pi)) = 1$ *but* $m \neq \mathsf{Dec}(\{sk_i\}_i, c)$ *is at most* $\nu(k)$.

Putting everything together, we say that a threshold encryption scheme is secure if it satisfies IND-CPA security, share simulatability, and soundness.

2.4 Compactly verifiable voting

In order for ballots to be cast and elections to be publicly verifiable, verifiable voting schemes use a public space (in practice, an append-only authenticated storage system) commonly referred to as a *bulletin board*. To describe an election, we break it up into several phases, which we describe here. To ease exposition, we implicitly assume that all parties are informed and agree about when a particular phase ends and the next one starts; e.g., by a particular symbol being written on the bulletin board.

– Setup. All authorities meet and jointly compute the public parameters of the election, while also keeping some correlated secrets private. All public parameters are published on the bulletin board.
– Voting. Each voter now uses these public parameters to encrypt his vote v and produce a ballot. All ballots are written on the bulletin board.
– Ballot processing. Next, once all ballots have been cast, they are examined to weed out invalid or duplicate ballots, and a set of mix authorities shuffle the remaining valid ballots.
– Tallying. Finally, a set of decryption authorities work together to decrypt the shuffled ballots. After decryption, the actual count of the votes can be performed publicly.

This multi-phase model of elections is inspired by the work of Juels et al. [21] and Bernhard et al. [4], although with some crucial modifications: unlike the former, we do not address coercion resistance, and unlike the latter we consider both shuffling and threshold decryption.

As far as security is concerned, there are a wide variety of properties we might want a voting scheme to satisfy; e.g., keeping users' votes private, coercion resistance, end-to-end verifiability, etc. In this paper, we focus mainly on this first property. As did Benaloh [3], we observe that we can provide voter privacy only up to a certain point; for example, if the election consisted of only one vote (or only one vote not controlled by some adversary), then voter privacy would be quite difficult to enforce! We therefore follow Benaloh's approach in requiring that votes can be private only in elections in which different assignments of honest votes still lead to the same outcome. To capture this property formally, we say that an election with N decryption authorities, L voters, and M mix authorities satisfies basic vote privacy if, for a random bit $b \xleftarrow{\$} \{0,1\}$, no PPT adversary $\mathcal{A}$ can win the following game with more than negligible advantage:

- Setup. First, a random bit $b \xleftarrow{\$} \{0,1\}$ is chosen. Then, $\mathcal{A}$ picks the decryption authorities to corrupt as $[N] \supset S \xleftarrow{\$} \mathcal{A}(1^k)$. Then, $\mathit{params} \xleftarrow{\$} \mathsf{Setup}(1^k)$, $(\{pk_i\}_i, \{sk_i\}_i) \xleftarrow{\$} \mathsf{ShuffleKg}(\mathit{params})$, $(pk, vk, \{dk_j\}_j) \xleftarrow{\$} \mathsf{EncKg}(\mathit{params})$. At the end of the setup phase $(\mathit{params}, pk, vk)$ are added to the bulletin board, and $\mathcal{A}$ gets to see $\{dk_j\}_{j\in S}$ and $T := \{pk_i\}_i$.
- Voting. Proceeding adaptively, the adversary can either provide his own ballot B, or a vote pair (v_0, v_1). For the former, the ballot is simply added to the bulletin board, while for the latter he gets back the ballot B_b (i.e., the ballot corresponding to either v_0 or v_1), which is also added to the bulletin board. At the end of the phase (i.e., once there are L votes on the board), $\mathcal{A}$ automatically loses if the election outcome differs between $b = 0$ and $b = 1$.
- Ballot processing. In this phase, in addition to access to the bulletin board, we give the adversary access to two shuffle oracles: an initial shuffle oracle that, on input pk_ℓ for $pk_\ell \in T$, writes $(\{c'_i\}_i, \pi, \{pk_\ell\}_\ell)$ on the bulletin board (if all the ballots on the bulletin board are valid), where π is a proof that the $\{c'_i\}_i$ constitute a valid shuffle of the initial ballots $\{B_i\}_i$ performed by the user corresponding to pk_ℓ (i.e., the user who knows sk_ℓ); and a regular shuffle oracle that, on input $(\{c'_i\}_i, \pi, \{pk_j\}_j, pk_\ell)$ for $pk_\ell \in T$, adds both $(\{c'_i\}_i, \pi, \{pk_j\}_j)$ (if it cannot be found there already) and the shuffled $(\{c''_i\}_i, \pi', \{pk_j\}_j \cup \{pk_\ell\})$ to the bulletin board. The phase ends when the final shuffle $(\{c'_i\}_i, \pi, \{pk_j\}_j)$ such that $|\{pk_j\}_j| = M$ and $\{pk_j\}_j \cap T \neq \emptyset$ is written to the bulletin board, either by the shuffle oracle or by the adversary.
- Tallying. The adversary can ask for decryption shares for the shuffled $\{c'_i\}_i$ through an oracle that, on input (j, k, s_k, I, ϕ_k), computes $(s'_k, I \cup \{j\}, \phi'_k) \xleftarrow{\$} \mathsf{ShareDec}(pk, vk, dk_j, c'_k, (s_k, I, \phi_k))$ and posts both (s_k, I, ϕ_k) (if it cannot be found there already; $s_k = \perp$ and $I = \emptyset$ denotes an initial decryption) and the share $(s'_k, I \cup \{j\}, \phi'_k)$ with its new contribution. The phase ends when, for every i, $1 \le i \le L$, the final decryption share $(s_i, [N], \phi_i)$ is written to the bulletin board, either by the share decryption oracle or by the adversary.
- Winning the game. The adversary outputs b', and wins if $b' = b$.

While the above definition explicitly captures vote privacy, we could also attempt to extend it to deal with verifiability by requiring that, if π and ϕ_i verify for all i, then the expected outcome (based on the v_i and the decryption of c_i in the adversary's ballots) should match the real outcome. While the soundness of the proofs used in our construction in Section 5 should guarantee that this holds, we focus solely on privacy in this work and leave a formal proof of verifiability as an interesting open problem.

3 A Compactly Verifiable Shuffle

In this section, we show how to achieve a compactly verifiable shuffle, as defined in Definition 2.3, with parameter size $O(L)$ and proof size $O(L+M)$ by using the verifiable shuffle due to Groth and Lu [19]. To do this, we use the following outline: first, we show that an adapted version of the Groth-Lu construction is what CKLM call *CM-friendly*, meaning that a pairing-based cm-NIZK can be constructed based on it. We then observe that, once we have a cm-NIZK, we can plug it into the generic construction of CKLM to obtain a compactly verifiable shuffle.

In the definition of CM-friendliness as proposed by CKLM [7, Definition 4.3], they assigned the property of CM-friendliness to a relation and transformation; in the case of a shuffle, this relation and the set of transformations describe the permutation and randomization of ciphertexts, as we saw formally in Section 2.2. We propose a useful weakening of this definition that shifts the assignation of CM-friendliness from the relation to its specific instantiation using a sound proof system; as we will see, this allows the definition to accomodate computationally sound proofs (i.e., arguments) as well as the perfectly sound proofs that the previous definition required. We capture the previous definition as *perfect* CM-friendliness.

Due to space constraints, we present here only an informal version of our definition; the formal definition can be found in the full version of the paper.

Definition 3.1. *(Informal.) For sets S and S' of pairing product equations and a PPT setup algorithm params $\xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ that specifies some group G, we say that $(S, S', \mathsf{CRSSetup})$ is a* CM-friendly instantiation *for a relation R and transformation class $\mathcal{T}$ if the following six properties hold: (1) representable statements: any instance and witness of R can be represented as a set of group elements; (2) representable transformations: any transformation in $\mathcal{T}$ can be represented as a set of group elements; (3) provable statements: proving satisfaction of S constitutes a computationally sound proof for the statement "$(x, w) \in R$" using the above representations for x and w; (4) provable transformations: proving satisfaction of S' constitutes a computationally sound proof for the statement "$T_{inst}(x') = x$ for $T \in \mathcal{T}$" using the above representations for x and T; (5) transformable statements: for any $T \in \mathcal{T}$ the statement "$(x, w) \in R$" (phrased using S as above) can be transformed into the statement "$(T_{inst}(x), T_{wit}(w)) \in R$"; and (6) transformable transformations: for any $T, T' \in \mathcal{T}$, the statement "$T_{inst}(x') = x$ for $T \in \mathcal{T}$" (phrased using S' as above) can be transformed using valid transformations into the statement "$\hat{T}_x(x') = \hat{x}$ for $\hat{T} \in \mathcal{T}$" where $\hat{T} = T' \circ T$ and $\hat{x} = \hat{T}_x(x)$. We say that $(S, S', \mathsf{CRSSetup})$ is a* perfect *CM-friendly instantiation if the probabilities in the third and fourth* *properties are zero. A relation and transformation class $(R, \mathcal{T})$ are (perfectly) CM-friendly, if they have a (perfect) CM-friendly instantiation.*

To instantiate the shuffle relation and transformations from Section 2.2, we combine the proof of hard relation instances of CKLM and an adapted version of the Groth-Lu protocol for the permutation proof. We omit the proof that the $\{pk_j\}_j$ are the public keys for R_{pk} in our exposition as it is unchanged from the original CKLM shuffle.

Our adapted version Groth-Lu protocol is slightly less efficient than theirs and achieves a weaker notion of zero knowledge (theirs is perfect whereas ours is computational) but a stronger notion of soundness (theirs achieves the slightly non-standard notion of L_{co}-soundness, whereas ours is computationally sound). These tradeoffs seem necessary, as it is not clear how to accomodate the definition of CM-friendliness (or of a cm-NIZK or compact shuffle) to allow for L_{co}-soundness.

Following Groth and Lu, the instantiation we use for the shuffle encryption scheme is Boneh-Boyen-Shacham (BBS) encryption [5], which uses a prime-order bilinear group setting (p, G, G_T, g, e) with public keys of the form $pk := (f, h)$ for $f := g^\alpha$ and $h := g^\beta$ (for random $\alpha, \beta \xleftarrow{\$} \mathbb{F}_p$) and ciphertexts of the form $c := (u, v, w)$ for $u := f^r$, $v := h^s$, and $w := g^{r+s}m$ (for the message m and $r, s \xleftarrow{\$} \mathbb{F}_p$). Using this, we can show how to satisfy CM-friendliness, starting with $\mathsf{CRSSetup}(1^k)$:

– $\mathsf{CRSSetup}(1^k)$: First generate a prime-order bilinear group (p, G, G_T, e, g). To allow for a shuffle over L ciphertexts, pick $x_1, \ldots, x_L \xleftarrow{\$} \mathbb{F}_p$ and set $g_i := g^{x_i}$ and $\gamma_i := g^{x_i^2}$ for all i. Output $\mathsf{crs} := (p, G, G_t, e, g, \{g_i\}_i, \{\gamma_i\}_i)$.

With this in place, we now describe how the six properties of CM-friendliness are met; in what follows, we highlight the involvement of the permutation by using $\varphi(g_i)$ in place of $g_{\varphi(i)}$ (and similarly for other variables):

1. Representable statements. Because we are using BBS encryption, instances will use $pk = (f, h)$, $c_i = (u_i, v_i, w_i)$, and $c'_i = (u'_i, v'_i, w'_i)$. We represent the witness as follows: to represent φ, we use $(\{a_i\}_i, \{b_i\}_i)$, where $a_i = \varphi(g_i)$ and $b_i = \varphi(\gamma_i)$ for all i, $1 \leq i \leq L$, and to represent R_i we use $(f^{r'_i}, h^{s'_i}, g^{r'_i}, g^{s'_i})$ for random $r'_i, s'_i \xleftarrow{\$} \mathbb{F}_p$.
2. Representable transformations. We represent $T_{(\varphi, \{R_i\}_i)} = (T_{\text{inst}}, T_{\text{wit}})$ in the same form as witnesses; i.e., $(\{a_i\}_i, \{b_i\}_i)$ for φ and $(f^{r'_i}, h^{s'_i}, g^{r'_i}, g^{s'_i})$ for all R_i.
3. Provable statements. To prove that, under the public key $pk = (f, h)$, the set of ciphertexts $\{(u'_i, v'_i, w'_i)\}_i$ is a shuffle of $\{(u_i, v_i, w_i)\}_i$ using the permutation represented by $(\{a_i\}_i, \{b_i\}_i)$ and re-randomization represented by $\{(f^{r'_i}, h^{s'_i}, g^{r'_i}, g^{s'_i})\}_i$, we use the set S of pairing product equations defined

as follows:

$$\textbf{(1)}\ \prod_{i=1}^{L} e(a_i, u_i') = \prod_{i=1}^{L} e(a_i, f^{r_i'})e(g_i, u_i),\ \ \textbf{(2)}\ \prod_{i=1}^{L} e(b_i, u_i') = \prod_{i=1}^{L} e(b_i, f^{r_i'})e(\gamma_i, u_i),$$
$$\textbf{(3)}\ \prod_{i=1}^{L} e(a_i, v_i') = \prod_{i=1}^{L} e(a_i, h^{s_i'})e(g_i, v_i),\ \ \textbf{(4)}\ \prod_{i=1}^{L} e(b_i, v_i') = \prod_{i=1}^{L} e(b_i, h^{s_i'})e(\gamma_i, v_i),$$
$$\textbf{(5)}\ \prod_{i=1}^{L} e(a_i, w_i') = \prod_{i=1}^{L} e(a_i, g^{r_i'}g^{s_i'})e(g_i, w_i),$$
$$\textbf{(6)}\ \prod_{i=1}^{L} e(b_i, w_i') = \prod_{i=1}^{L} e(b_i, g^{r_i'}g^{s_i'})e(\gamma_i, w_i),$$
$$\textbf{(7)}\ \prod_{i=1}^{L} a_i g_i^{-1} = 1,\ \ \textbf{(8)}\ \prod_{i=1}^{L} b_i \gamma_i^{-1} = 1,\ \textbf{(9)}\ e(a_i, a_i) = e(g, b_i) \text{ for all } i,\ 1 \le i \le L,$$
$$\textbf{(10)}\ e(f^{r_i'}, g) = e(f, g^{r_i'}) \text{ for all } i, \text{and } \textbf{(11)}\ e(h^{s_i'}, g) = e(h, g^{s_i'}) \text{ for all } i.$$

4. Provable transformations. To prove $T_{\text{inst}}(x') = x$ for $T \in \mathcal{T}$, we use the same equations from the above set S. We must additionally prove that the transformation does not change pk or $\{c_i\}_i$; to do this, we form an augmented set S', which consists of all the equations in S as well as equations to check that these values stay fixed. More formally, if we represent X as $(pk, \{(u_i, v_i, w_i)\}_i, \{(u_i', v_i', w_i')\}_i)$ and X' as $(pk', \{(U_i, V_i, W_i)\}_i, \{U_i', V_i', W_i'\}_i)$, then our extra checks ensure that $pk = pk'$ and $u_i = U_i$, $v_i = V_i$, $w_i = W_i$ for all i, $1 \le i \le L$. We can then run the checks in S using T_{inst} as the witness and $X_T := (pk, \{(u_i', v_i', w_i')\}_i, \{(U_i', V_i', W_i')\}_i)$ as the instance.
5. Transformable statements. CKLM already show how to permute variables by a permutation φ and multiply re-randomization factors into ciphertexts using valid transformations; we therefore assume these operations exist and are valid. To change the statement $(x, w) \in R$ into $(T_{\text{inst}}(x), T_{\text{wit}}(w)) \in R$ for $X = (pk, \{(u_i, v_i, w_i)\}_i, \{(u_i', v_i', w_i')\}_i)$, $W = ((\{a_i\}_i, \{b_i\}_i), \{R_i\}_i)$, and $T = (\varphi', \{R_i'\}_i)$, we therefore begin by permuting the values $\{(u_i', v_i', w_i')\}_i$, $\{a_i\}_i$, and $\{b_i\}_i$ by φ'; this operation affects Equations 1 through 9 in S. We then multiply the additional randomness $\{R_i'\}_i$ into Equations 1 through 6, as well as Equations 10 and 11.
6. Transformable transformations. To change the statement $T_{\text{inst}}(x') = x$ into $T_{\text{inst}}' \circ T_{\text{inst}}(x') = T_{\text{inst}}'(x)$, we leave the additional checks in S' (i.e., the checks that ensure that pk and $\{c_i\}_i$ go unchanged) as they are. We then transform S as we did above using the values $(\varphi', \{R_i'\}_i)$ specified in T_{inst}', so that we permute the values $\{(u_i', v_i', w_i')\}_i$, $\{a_i\}_i$, and $\{b_i\}_i$ by φ' and multiply the additional randomness into Equations 1 through 6 and 10 and 11.

Due to space constraints, a proof of the following theorem can be found in the full version of the paper:

Theorem 3.1. *If both the Permutation Pairing and Simultaneous Pairing assumptions hold, then* $(S, S', \mathsf{CRSSetup})$ *as defined above is a CM-friendly instantiation for the shuffle relation* R *(defined in Section 2.2) and the transformation class* $\mathcal{T}$ *consisting of all valid shuffles.*

Now that we have a CM-friendly instantiation for the shuffle relation, we can use the results of CKLM to construct a cm-NIZK for this relation. As we slightly weakened the notion of CM-friendliness, we argue in the full version of the paper that their results still carry through to produce a cm-NIZK; we mention here that our proof is nearly identical, as the notion of soundness used for cm-NIZKs is already computational.

Armed with our cm-NIZK, we now plug it into the generic verifiable shuffle construction of CKLM , which they already proved secure. We can even use the same representation of mix server keys as CKLM, which means $pk_j := g^{\alpha_j}$ and $sk_j := h^{\alpha_j}$ for $\alpha_j \xleftarrow{\$} \mathbb{F}_p$ and $h := g^{\beta}$ for some $\beta \xleftarrow{\$} \mathbb{F}_p$. As for the size, looking at the construction above we see that the CRS must contain the g_i and γ_i elements for all i (and adding in the parameters for R_{pk} adds only the single group element h), which means the parameters are of size $O(L)$. For the proofs, Equations 9, 10, and 11 in S are required for every i, so the size of the proof is also $O(L)$. In addition, a constant number of equations is required to check that $(pk_j, sk_j) \in R_{pk}$ for every value of j; if the number of mix authorities is M, then this adds a proof component of size $O(M)$ and thus our total proof size is $O(L + M)$.

4 Threshold Decryption

In this section, we provide our construction of a threshold encryption scheme that satisfies the notions of security defined in Section 2.3; i.e., IND-CPA security, share simulatability, and soundness. We provide first a construction using a generic malleable NIZK proof of knowledge (NIZKPoK), and then describe in the full version of the paper [9]how to instantiate this proof system concretely.

4.1 Our construction

In threshold decryption, the statement that each participant i wants to prove is that the share s he produces is a correct partial decryption of some ciphertext c. Formally, we represent instances as $x = (vk_c, c, s)$, where c is a ciphertext, and s is the cumulative decryption share produced by the combined user represented in vk_c, and witnesses as $(t, open)$, where t is a secret token (in our case, a bijection applied to the cumulative secret key) used to prove correctness of partial decryption, and $vk_c = \mathsf{Com}(t; open)$ for some commitment scheme Com. The statement we want to prove is then

$$\begin{aligned}((vk_c, c, s),(t, open)) \in R \Leftrightarrow& \\ &\exists sk_c : vk_c = \mathsf{Com}(t; open) \wedge t = F(sk_c) \wedge s = \mathsf{Dec}(sk_c, c), \qquad (1)\end{aligned}$$

where F is the bijection between cumulative secret keys and tokens.

Transformations for this relation correspond to a new set of users J folding in their shares. This means we represent transformations as $T = (\hat{s}, \hat{t}, \widehat{open})$, where $T_{\text{inst}}(vk_c, c, s) := (vk_c \cdot \mathsf{Com}(\hat{t}; \widehat{open}), c, s \cdot \hat{s})$ and $T_{\text{wit}}(t, open) = (t \cdot \hat{t}, open \cdot \widehat{open})$; the transformation is considered allowable if $\hat{s}$ is a valid share using the token $\hat{t}$; i.e., $\hat{s}$ was computed using the secret key $\widehat{sk}$ corresponding to $\hat{t}$.

Our concrete instantiation uses BBS encryption [5], which is multiplicatively homomorphic; this is why we multiply both the shares and the tokens to combine them. We also use a commitment scheme Com and a strongly derivation-private malleable NIZK proof of knowledge $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$. We will see later how to instantiate the NIZK concretely; for the commitment scheme (which we use to commit to the two components of t) we can use the instantiation of Groth-Sahai commitments under Decision Linear, which are almost identical to BBS encryption (and thus also multiplicatively homomorphic). We thus usually keep these parameters implicit.

- $\mathsf{EncKg}(1^k)$: Generate $\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ and $\mathsf{par} \xleftarrow{\$} \mathsf{ComSetup}(1^k)$; these are defined over a shared bilinear group (p, G, G_T, e, g). Pick random $\alpha, \beta \xleftarrow{\$} \mathbb{F}_p$, set $f := g^\alpha$ and $h := g^\beta$, and set $pk := (f, h)$. Next, to allow N parties to partake in decryption, compute $a_1, b_1, \ldots, a_{N-1}, b_{N-1} \xleftarrow{\$} \mathbb{F}_p$ and $a_N := -1/\alpha - \sum_i a_i$ and $b_N := -1/\beta - \sum_i b_i$. Next, for all i, set $t_{1i} := g^{a_i}$, $t_{2i} := g^{b_i}$, and form commitments $A_i \xleftarrow{\$} \mathsf{Com}(t_{1i}; open_{1i})$ and $B_i \xleftarrow{\$} \mathsf{Com}(t_{2i}; open_{2i})$ using random openings. Set $vk' := \{(A_i, B_i)\}_i$ and $sk_i := (a_i, b_i, t_{1i}, t_{2i}, open_{1i}, open_{2i})$ for all i, $1 \leq i \leq n$. Output pk, $vk := (\mathsf{crs}, \mathsf{par}, vk')$, and $\{sk_i\}_i$.
- $\mathsf{Enc}(pk, m)$: Parse $pk = (f, h)$ and pick random $r, s \xleftarrow{\$} \mathbb{F}_p$. Set $u := f^r$, $v := h^s$, $w := g^{r+s} m$, and output $c := (u, v, w)$.
- $\mathsf{Dec}(\{sk_i\}_{i=1}^n, c)$: Parse $c = (u, v, w)$ and $sk_i = (a_i, b_i, t_{1i}, t_{2i}, open_{1i}, open_{2i})$ for all i, and compute $a := \sum_i a_i$ and $b := \sum_i b_i$. Output $m := u^a \cdot v^b w$. (By definition, $a = -1/\alpha$ and $b = -1/\beta$, so this is just standard BBS decryption with a reconstructed key.)
- $\mathsf{ShareDec}(pk, vk, sk_j, c, (s, I, \pi))$: Parse $sk_j = (a_j, b_j, t_{1j}, t_{2j}, open_{1j}, open_{2j})$. If $(s, I, \pi) = (\bot, \bot, \bot)$, then this is the initial decryption. Compute the share $s_j := u^{a_j} v^{b_j} w$ and $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, (vk_j, c, s_j), (t_{1j}, t_{2j}, open_{1j}, open_{2j}))$, and output $(s_j, \{j\}, \pi)$.
  Otherwise, define $vk_c := \prod_{i \in I} vk'_i$ and check that $\mathcal{V}(\mathsf{crs}, (pk, vk_c, c, s), \pi) = 1$; abort and output $\bot$ if not. Otherwise continue and compute $s_j := u^{a_j} v^{b_j}$ and $s' := s \cdot s_j$; then set $T := (s_j, (t_{1j}, t_{2j}), (open_{1i}, open_{2i}))$. Compute $\pi' \xleftarrow{\$} \mathsf{ZKEval}(\mathsf{crs}, T, (vk_c, c, s), \pi)$, and output $(s', I' := I \cup \{j\}, \pi')$.
- $\mathsf{ShareVerify}(pk, vk, c, (s, I, \pi))$: Parse $vk = (\mathsf{crs}, \mathsf{par}, vk')$ and output $\mathcal{V}(\mathsf{crs}, (pk, \prod_{i \in I} vk'_i, c, s), \pi)$.

As the security of both BBS encryption and our cm-NIZK come from Decision Linear, we obtain the following theorem.

Theorem 4.1. *If Decision Linear holds in G then we can instantiate the above construction to obtain a secure threshold decryption scheme, as defined in Section 2.3.*

To prove this, we must prove that four properties are satisfied: completeness, IND-CPA security, soundness, and share simulatability. The first of these, completeness, follows directly by inspection; similarly, for IND-CPA security, as we use BBS encryption, IND-CPA follows directly from their result and holds under Decision Linear.

For the latter two, we prove them using the security of the commitment scheme and NIZK. Interestingly, while the proof system is required to be malleable, strongly derivation private, and zero knowledge, for soundness we require not the strong notion of CM-SSE for cm-NIZKs, but instead regular extractability (i.e., we require the proof to be a proof of knowledge). Intuitively, the reason for this is that in the soundness game the adversary is not provided with simulated proofs, and we can therefore always expect to be able to extract a witness (rather than just a transformation as we do with CM-SSE).

Lemma 4.1. *If* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *is extractable and* Com *is binding, the threshold encryption scheme describe above is sound, as defined in Definition 2.5.*

Lemma 4.2. *If* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *is zero knowledge and strongly derivation private, and* Com *is hiding, the threshold encryption scheme described above is share simulatable, as defined in Definition 2.4.*

Due to space constraints, the proofs of these lemmas and the concrete implementation of the cm-NIZK, using Groth-Sahai proofs, can be found in the full version of the paper. For our concrete instantiation, we mention here that we follow the same outline as in Section 3 to show that R has a CM-friendly instantiation. In fact, as we encode x, w, and T directly without relying on any computational assumptions, we can achieve *perfect* CM-friendliness.

5 A Secure Voting Scheme

In this section, we bring together the components constructed in the previous two sections to construct an electronic voting scheme from a compactly verifiable shuffle $(\mathsf{Setup}, \mathsf{ShuffleKg}, \mathsf{Shuffle}, \mathsf{Verify})$, a secure threshold decryption scheme $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify})$, and a simulation-sound extractable proof $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$.

- Setup. The voting authorities jointly compute the parameters $params \xleftarrow{\$} \mathsf{Setup}(1^k)$ and threshold keys $(pk, vk, \{dk_j\}_j) \xleftarrow{\$} \mathsf{EncKg}(params)$. The mix authorities compute the shuffling keys $(\{pk_i\}_i, \{sk_i\}_i) \xleftarrow{\$} \mathsf{ShuffleKg}(params)$, and the values $params$, pk, and vk are added to the bulletin board.

– Voting. Each voter i forms $c_i \xleftarrow{\$} \mathsf{Enc}(pk, v_i)$ (using some randomness r_i) and proves knowledge of his vote by computing $\pi_i \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, (pk, c), (v_i, r_i))$. The resulting ballot (c_i, π_i) is added to the bulletin board.
– Ballot processing. The mix authority with public key pk_k picks the most recent valid shuffle $(\{c'_i\}_i, \pi, \{pk_j\}_j)$; e.g., the one with the most public keys, or the one that has used the correct sequence of public keys (if an order has been imposed). It performs $(\{c''_i\}_i, \pi') \xleftarrow{\$} \mathsf{Shuffle}(params, \{c_i, \pi_i\}_i, \{c'_i\}_i, \pi, \{pk_j\}_j, (pk_k, sk_k))$ and posts $(\{c''_i\}_i, \pi', \{pk_j\}_j \cup \{pk_k\})$ to the bulletin board. The ballot processing phase ends once there is a valid sequence of shuffle proofs with sufficiently many mix authorities.
– Tallying. Let $(\{c'_i\}_i, \pi, \{pk_j\}_j)$ be the completed shuffle. Each decryption authority looks for the valid decryption shares (s_i, I, ϕ_i) with the largest set I. The k-th decryption authority performs $(s'_i, I \cup \{k\}, \phi'_i) \xleftarrow{\$} \mathsf{ShareDec}(pk, vk, dk_k, c_i, (s_i, I, \phi_i))$ for all i and posts $(s'_i, I \cup \{k\}, \phi'_i)$ on the bulletin board.

Theorem 5.1. *The voting scheme outlined above satisfies basic voter privacy, as defined in Section 2.4.*

To prove this, we proceed through a series of game transformations; due to space constraints, the transformations and proofs of their indistinguishability can be found in the full version of the paper.

Acknowledgments

We thank Stephan Neumann for spurring our interest in the application of malleable proofs to threshold decryption. Anna Lysyanskaya was supported by NSF grants 1012060, 0964379, 0831293, and by a Sloan Foundation fellowship, and Sarah Meiklejohn was supported by a MURI grant administered by the Air Force Office of Scientific Research.

References

1. M. Abe. Universally verifiable mix-net with verification work indendent of the number of mix-servers. In *Proceedings of EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447. Springer, 1998.
2. B. Adida and C. A. Neff. Efficient receipt-free ballot casting resistant to covert channels. Cryptology ePrint Archive, Report 2008/207, 2008. `http://eprint.iacr.org/2008/207`.
3. J. D. C. Benaloh. *Verifiable Secret-Ballot Elections.* PhD thesis, Yale University, 1987.
4. D. Bernhard, V. Cortier, O. Pereira, B. Smyth, and B. Warinschi. Adapting Helios for provable ballot privacy. In V. Atluri and C. Díaz, editors, *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 335–354. Springer, 2011.
5. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Proceedings of Crypto 2004*, volume 3152 of *LNCS*, pages 41–55. Springer-Verlag, 2004.

6. R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In J. Stern, editor, *EUROCRYPT '99*, volume 1592 of *LNCS*, pages 90–106. Springer Verlag, 1999.
7. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. In *Proceedings of Eurocrypt 2012*, pages 281–300, 2012.
8. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Succinct malleable NIZKs and an application to compact shuffles. Cryptology ePrint Archive, Report 2012/506, 2012. `http://eprint.iacr.org/2012/506`.
9. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Verifiable elections that scale for free. Cryptology ePrint Archive, 2012. `http://eprint.iacr.org/`.
10. I. Damgård. On sigma protocols. `http://www.daimi.au.dk/~ivan/Sigma.pdf`.
11. I. Damgård, J. Groth, and G. Salomonsen. The theory and implementation of an electronic voting system. In *Proceedings of Secure Electronic Voting (SEC)*, pages 77–100, 2003.
12. A. de Santis, G. di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Proceedings of Crypto 2001*, volume 2139 of *LNCS*, pages 566–598. Springer-Verlag, 2001.
13. Y. Desmedt and Y. Frankel. Threshold cryptography. In *CRYPTO '89*, volume 435 of *LNCS*, pages 307–315. Springer-Verlag, 1990.
14. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings of Crypto 1986*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, 1986.
15. J. Furukawa and H. Imai. An efficient aggregate shuffle argument scheme. In S. Dietrich and R. Dhamija, editors, *Financial Cryptography*, volume 4886 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2007.
16. J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 145–160. Springer-Verlag, 2003.
17. J. Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS*, volume 3531 of *LNCS*, pages 467–482. Springer-Verlag, 2005.
18. J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *Proceedings of Asiacrypt 2006*, volume 4284 of *LNCS*, pages 444–459. Springer-Verlag, 2006.
19. J. Groth and S. Lu. A non-interactive shuffle with pairing-based verifiability. In *Proceedings of Asiacrypt 2007*, volume 4833 of *LNCS*, pages 51–67. Springer-Verlag, 2007.
20. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Proceedings of Eurocrypt 2008*, volume 4965 of *LNCS*, pages 415–432. Springer-Verlag, 2008.
21. A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. A. Ryan, J. Benaloh, M. Kutylowski, and B. Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 37–63. Springer, 2010.
22. C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of ACM CCS 2001*, pages 116–125. ACM press, Nov. 2001.
23. D. Sandler, K. Derr, and D. S. Wallach. Votebox: A tamper-evident, verifiable electronic voting system. In *USENIX Security Symposium*, pages 349–364, 2008.
24. V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Proceedings of Eurocrypt 1998*, volume 1403 of *LNCS*, pages 1–16. Springer-Verlag, 1998.