Full version of an extended abstract published in Proceedings of PKC 2013, Springer-Verlag, 2013. Available from the IACR Cryptology ePrint Archive as Report 2012/xxx.

Verifiable Elections That Scale for Free

Melissa Chase
Microsoft Research Redmond
melissac@microsoft.com

Markulf Kohlweiss
Microsoft Research Cambridge
markulf@microsoft.com

Anna Lysyanskaya
Brown University
anna@cs.brown.edu

Sarah Meiklejohn
UC San Diego
smeiklej@cs.ucsd.edu

December 11, 2012

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* [25, 12, 18, 19, 21], 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 [14, 27, 7]; 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 [24, 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 [25, 26, 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 [8], 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, 17] required complex interactions between shuffling authorities and, for non-interactive verification, were based on the Fiat-Shamir [16] 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 $(c_1^{(M)}, \ldots, c_L^{(M)})$, 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 $(c_1^{(0)}, \ldots, c_L^{(0)})$. 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 $(c_1^{(M-1)}, \ldots, c_L^{(M-1)})$. 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 $(c_1^{(i-1)}, \ldots, c_L^{(i-1)})$ in order to obtain the proof π_i attesting to the validity of $(c_1^{(i)}, \ldots, c_L^{(i)})$; 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 [23].

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 $(c_1^{(i-1)}, \ldots, c_L^{(i-1)})$, result

in $(c_1^{(i)}, \ldots, c_L^{(i)})$. 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 [21], 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 [10] 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 $(c_1^{(M)}, \ldots, c_L^{(M)})$, outputs the decryption shares $(d_1^{(i)}, \ldots, d_L^{(i)})$ and the proofs $(\pi_1^{(i)}, \ldots, \pi_L^{(i)})$ 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 [14, 27, 7] 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 [8] 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 Appendix A); 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 [13, 20]) 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. [8] *Let $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ 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 NIZKPoK 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. [8] *For a malleable NIZK proof system* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ *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. [8] *For a verifiable shuffle* $(\mathsf{Setup}, \mathsf{ShuffleKg}, \mathsf{Shuffle}, \mathsf{Verify})$ *with respect to an encryption scheme* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{Dec})$*, 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.* $params \xleftarrow{\$} \mathsf{Setup}(1^k)$, $(pk, sk) \xleftarrow{\$} \mathsf{EncKg}(params)$, $(T = \{pk_i\}_i, \{sk_i\}_i) \xleftarrow{\$} \mathsf{ShuffleKg}(1^k)$.
- *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 [11], 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 Appendix B.

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 $\mathsf{Dec}(sk, c) = \mathsf{ShareDec}(pk, vk, sk, c, (\perp, \perp, \perp))$.

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. *For a threshold encryption scheme* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify})$ *with N participants, 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* $SD_{(pk,\tau,vk,\{sk_j\}_{j=1}^N)}(i, c, s, I, \pi))$
$(pk, vk', \{sk'_j\}_{j=1}^N) \xleftarrow{\$} \mathsf{EncKg}(1^k, N)$	$m \leftarrow \mathsf{Dec}(\{sk_j\}_{j=1}^N, c)$
$(vk, \{sk_j\}_{j \in S}, \tau) \xleftarrow{\$} \mathsf{SimKg}(pk, vk', N, S)$	$(s', \pi') \xleftarrow{\$} \mathsf{SimShareDec}(pk, vk, \tau, (i, c, s, I, \pi), m)$
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. *For a threshold encryption scheme* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify})$ *with n participants, an adversary $\mathcal{A}$, define the following game:*

- *Step 1.* $(pk, vk, \{sk_i\}_i) \xleftarrow{\$} \mathsf{EncKg}(1^k, n)$.
- *Step 2.* $(c, m, \pi) \xleftarrow{\$} \mathcal{A}(pk, vk, \{sk_i\}_i)$,

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. [24] 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, $\mathcal{A}$ picks the decryption authorities to corrupt as $[N] \supset S \xleftarrow{\$} \mathcal{A}(1^k)$. Then, $b \xleftarrow{\$} \{0,1\}$, $params \xleftarrow{\$} \mathsf{Setup}(1^k)$, $(\{pk_i\}_i, \{sk_i\}_i) \xleftarrow{\$} \mathsf{ShuffleKg}(params)$, $(pk, vk, \{dk_j\}_j) \xleftarrow{\$} \mathsf{EncKg}(params)$. At the end of the setup phase $(params, pk, vk)$ 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, $1 \leq i \leq L$, 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 [21]. 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 [8, 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.

Definition 3.1. *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; i.e., there is an efficiently computable invertible function $F_s(params, \cdot)$ that maps $L_R \mapsto G^{d_s}$ for some d_s, and simlarly there is an efficiently computable invertible function $F_w(params, \cdot)$ that maps $W_R \mapsto G^{d_w}$ for some d_w and $W_R := \{w \mid \exists x : (x, w) \in R\}$.*
2. *Representable transformations. Any transformation in $\mathcal{T}$ can be represented as a set of group elements; i.e., there is an efficiently computable invertible function $F_t(params, \cdot)$ that maps $\mathcal{T} \mapsto G^{d_t}$ for some d_t.*
3. *Provable statements. Proving satisfaction of the set S constitutes a computationally sound proof for the statement "$(x, w) \in R$" using the above representations for x and w; i.e., for $params \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ it holds that (1) if $(x, w) \in R$ then $F_s(params, x)$ and $F_w(params, w)$ satisfy S, and (2) for a PPT adversary $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that the probability (over the randomness used in $\mathsf{CRSSetup}$ and $\mathcal{A}$) that $\mathcal{A}$ can produce (X, W) such that X and W satisfy S but $(F_s^{-1}(params, X), F_w^{-1}(params, W)) \notin R$ is at most $\nu(k)$.*
4. *Provable transformations. Proving satisfaction of the set 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; i.e., for $params \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ it holds that (1) if $T \in \mathcal{T}$ and $T_{inst}(x') = x$ then $F_t(params, T)$, $F_s(params, x)$, and $F_s(params, x')$ satisfy S', and (2) for a PPT adversary $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that the probability (over the randomness used in $\mathsf{CRSSetup}$ and $\mathcal{A}$) that $\mathcal{A}$ can produce (T', X, X') such that T', X, and X' satisfy S'*

but $F_t^{-1}(params, T') \notin \mathcal{T}$ *or* $F_t^{-1}(params, T')(F_s^{-1}(params, X')) \neq F_s^{-1}(params, X)$ *is at most* $\nu(k)$.

5. *Transformable statements. For any* $T \in \mathcal{T}$*, the statement "*$(x, w) \in R$*" (phrased using* S *as above) can be transformed using some* valid[1] *set of transformations into the statement "*$(\hat{x}, \hat{w}) \in R$*" where* $\hat{x} = T_{inst}(x)$ *and* $\hat{w} = T_{wit}(w)$.
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 "*$\hat{T_{inst}}(x') = \hat{x}$ *for* $\hat{T} \in \mathcal{T}$*" where* $\hat{T} = T' \circ T$ *and* $\hat{x} = T'_{inst}(x)$.

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. We first recall the assumptions used by Groth and Lu (which they prove secure in the generic group model):

Assumption 3.2 (Permutation Pairing assumption)**. [21]** *Given* (p, G, G_T, e, g) *and* $(\{g_i := g^{x_i}\}_i, \{\gamma_i := g^{x_i^2}\}_i)$ *for random* $x_1, \ldots, x_n \xleftarrow{\$} \mathbb{F}_p$*, it is hard to compute elements* $(\{a_i\}_i, \{b_i\}_i)$ *such that* $\prod_i a_i = \prod_i g_i$*,* $\prod_i b_i = \prod_i \gamma_i$*, and* $e(a_i, a_i) = e(g, b_i)$ *for all* i*,* $1 \leq i \leq n$*, but* $\{a_i\}_i$ *is not a permutation of* $\{g_i\}_i$.

Assumption 3.3 (Simultaneous Pairing assumption)**. [21]** *Given* (p, G, G_T, e, g) *and* $(\{g_i := g^{x_i}\}_i, \{\gamma_i := g^{x_i^2}\}_i)$ *for random* $x_1, \ldots, x_n \xleftarrow{\$} \mathbb{F}_p$*, it is hard to compute elements* $\{\mu_i\}_i$ *that are not all equal to* 1 *such that* $\prod_i e(\mu_i, g_i) = 1$ *and* $\prod_i e(\mu_i, \gamma_i) = 1$.

Following Groth and Lu, the instantiation we use for the shuffle encryption scheme is Boneh-Boyen-Shacham (BBS) encryption [6], 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] Briefly, valid transformations are those that can be performed on a set of pairing product equations; for more details, we refer the reader to the definition of CKLM [8, Definition 4.1].

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, $1 \le i \le L$, 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:

$$(1)\ \prod_{i=1}^{L} e(a_i, u'_i) = \prod_{i=1}^{L} e(a_i, f^{r'_i}) e(g_i, u_i), \qquad (2)\ \prod_{i=1}^{L} e(b_i, u'_i) = \prod_{i=1}^{L} e(b_i, f^{r'_i}) e(\gamma_i, u_i),$$

$$(3)\ \prod_{i=1}^{L} e(a_i, v'_i) = \prod_{i=1}^{L} e(a_i, h^{s'_i}) e(g_i, v_i), \qquad (4)\ \prod_{i=1}^{L} e(b_i, v'_i) = \prod_{i=1}^{L} e(b_i, h^{s'_i}) e(\gamma_i, v_i),$$

$$(5)\ \prod_{i=1}^{L} e(a_i, w'_i) = \prod_{i=1}^{L} e(a_i, g^{r'_i} g^{s'_i}) e(g_i, w_i), \qquad (6)\ \prod_{i=1}^{L} e(b_i, w'_i) = \prod_{i=1}^{L} e(b_i, g^{r'_i} g^{s'_i}) e(\gamma_i, w_i),$$

$$(7)\ \prod_{i=1}^{L} a_i g_i^{-1} = 1, \qquad (8)\ \prod_{i=1}^{L} b_i \gamma_i^{-1} = 1,$$

$$(9)\ e(a_i, a_i) = e(g, b_i) \text{ for all } i,\ 1 \le i \le L, \qquad (10)\ e(f^{r'_i}, g) = e(f, g^{r'_i}) \text{ for all } i,$$

$$(11)\ e(h^{s'_i}, g) = e(h, g^{s'_i}) \text{ for all } i.$$

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 $X = (pk, \{(u_i, v_i, w_i)\}_i, \{(u'_i, v'_i, w'_i)\}_i)$ and $X' = (pk', \{(U_i, V_i, W_i)\}_i, \{U'_i, V'_i, W'_i\}_i)$, 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 themselves 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.

Theorem 3.4. *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.*

Proof. To show this, we take an adversary $\mathcal{A}$ that can break one of the properties in Definition 3.1 with non-negligible probability ϵ and use it to construct an adversary $\mathcal{B}$ that can break either the Permutation Pairing or Simultaneous Pairing assumptions with the same probability ϵ.

To start, $\mathcal{B}$ will get as input $params := (p, G, G_T, e, g, \{g_i\}_i, \{\gamma_i\}_i)$, where $g_i = g^{x_i}$ and $\gamma_i = g^{x_i^2}$ for all i. It can then pass $params$ along to $\mathcal{A}$. At some point, $\mathcal{A}$ will produce either a pair (X, W) or a triple (T', X, X'). In the former case, the pair should be of the form $(X := ((f,h), \{(u_i, v_i, w_i)\}_i, \{(u'_i, v'_i, w'_i)\}_i), W := ((\{a_i\}_i, \{b_i\}_i), \{(f^{r'_i}, h^{s'_i}, g^{r'_i + s'_i})\}_i))$. In order for $\mathcal{A}$ to win, S must be satisfied by X and W and, for $x := F_s^{-1}(params, X)$ and $w := F_w^{-1}(params, W)$, one of three cases must hold: (1) x is not a valid (i.e., properly formatted) instance for R, (2) w is not a valid witness for R, or (3) $(x, w) \notin R$; i.e., $\{(u'_i, v'_i, w'_i)\}_i$ is not a permutation and re-randomization of $\{(u_i, v_i, w_i)\}_i$. We deal with each of these cases in turn.

We start by observing that the first case, in which X is not properly formatted, is in fact impossible. Looking at S, we know that it can be satisfied only by values in G; this means that in order for X to satisfy S, it must consist of elements in G and we can parse it as such. If we look first at the public key (f, h), we know that an honest public key is formed as $f := g^\alpha$ and $h := g^\beta$. As g is a generator of the group, any pair of values in G will therefore represent a public key. If we consider next the ciphertexts $\{(u_i, v_i, w_i)\}_i$ and $\{(u'_i, v'_i, w'_i)\}_i$, we notice that for BBS encryption, once again any three values in G constitute a valid ciphertext. To see this, observe that for any values $U, V, W \in G$, they can be represented respectively as g^u, g^v, and g^w for $u, v, w \in \mathbb{F}_p$. Then if we use $r := u/\alpha$, $s := v/\beta$, $m := w - (r + s)$, and $M := g^m$, we get $U = g^u = g^{r\alpha} = f^r$, $V = g^v = g^{s\beta} = h^s$, and $W = g^w = g^m g^{r+s} = g^{r+s} M$, so (U, V, W) is a valid encryption of M.

We next turn to the second case, in which W is not properly formatted. For the re-randomization factors $\{F_i, H_i, G_{ri}, G_{si}\}_i$, we first observe that any two values $F_i, H_i \in G$ can be written respectively as $f^{r'_i}$ and $h^{s'_i}$, as if we write $F_i := g^{f_i}$ and $H_i := g^{h_i}$ for some f_i and h_i, we can set $r'_i := f_i/\alpha$ and $s'_i := h_i/\beta$. If additionally Equations 10 and 11 are satisfied, then we know that $G_{ri} = g^{r'_i}$ and $G_{si} = g^{s'_i}$, so the re-randomization values have the appropriate form. This leaves as the only possibility the case in which the $\{a_i\}_i$ and $\{b_i\}_i$ do not actually represent a permutation φ. If this is true then we observe that, because S is assumed to be satisfied, in particular Equations 7, 8, and 9 will all be satisfied. Given $(\{a_i\}_i, \{b_i\}_i)$ that therefore satisfy these equations but do not represent a valid permutation, $\mathcal{B}$ can just output this tuple itself to break the Permutation Pairing assumption.

Finally, we look at the third case. In this case, we know that $\mathcal{A}$ has produced $\{(u'_i, v'_i, w'_i)\}_i$ and $\{(u_i, v_i, w_i)\}_i$ such that the former is not a shuffle of the latter; this means that, for the permutation φ and re-randomization factors $\{f^{r'_i}, h^{s'_i}, g^{r'_i}, g^{s'_i}\}_i$ represented by W, there is an index i such that either (1) $u'_i \neq \varphi(u_i) \cdot f^{r'_i}$, (2) $v'_i \neq \varphi(v_i) \cdot h^{s'_i}$, or (3) $w'_i \neq \varphi(w_i) \cdot g^{r'_i} g^{s'_i}$. To argue that none of these cases can in fact occur, we observe that because $a_i = \varphi(g_i)$, then for any set $\{x_i\}_i$ it holds that

$$\prod_{i=1}^{L} e(g_i, x_i) = \prod_{i=1}^{L} e(\varphi^{-1}(a_i), x_i) = \prod_{i=1}^{L} e(a_i, \varphi(x_i)).$$

This means that, in Equation 1, we can substitute in $\prod_i e(a_i, \varphi(u_i))$ for $\prod_i e(g_i, u_i)$ and divide both sides by $\prod_i e(a_i, u'_i)$ to get

$$1 = \prod_{i=1}^{L} e(a_i, f^{r'_i}) e(a_i, \varphi(u_i)/u'_i) = \prod_{i=1}^{L} e(a_i, \varphi(u_i) f^{r'_i}/u'_i).$$

Using a similar derivation for Equations 2 through 6, we can also see that

$$1 = \prod_{i=1}^{L} e(b_i, \varphi(u_i) f^{r'_i} / u'_i),$$
$$1 = \prod_{i=1}^{L} e(a_i, \varphi(v_i) h^{s'_i} / v'_i),$$
$$1 = \prod_{i=1}^{L} e(b_i, \varphi(v_i) h^{s'_i} / v'_i),$$
$$1 = \prod_{i=1}^{L} e(a_i, \varphi(w_i) g^{r'_i} g^{s'_i} / w'_i), \text{ and}$$
$$1 = \prod_{i=1}^{L} e(b_i, \varphi(w_i) g^{r'_i} g^{s'_i} / w'_i).$$

Note that each of the winning cases for $\mathcal{A}$ correspond to the case in which the right-hand side of the pairings are not equal to 1; e.g., if case 1 holds and $u'_i \neq \varphi(u_i) f^{r'_i}$ then $\varphi(u_i) f^{r'_i} / u'_i \neq 1$. By applying φ^{-1}, this further implies that $u_i \varphi^{-1}(f^{r'_i}) / \varphi^{-1}(u'_i) \neq 1$, so

$$1 = \prod_{i=1}^{L} e(a_i, \varphi(u_i) f^{r'_i} / u'_i) = \prod_{i=1}^{L} e(g_i, u_i \varphi^{-1}(f^{r'_i}) / \varphi^{-1}(u'_i)), \text{ and}$$
$$1 = \prod_{i=1}^{L} e(b_i, \varphi(u_i) f^{r'_i} / u'_i) = \prod_{i=1}^{L} e(\gamma_i, u_i \varphi^{-1}(f^{r'_i}) / \varphi^{-1}(u'_i)),$$

and similarly for the rest of the equations. In particular then, if case 1 holds then $\mathcal{B}$ can set $\mu_i := u_i \varphi^{-1}(f^{r'_i}) / \varphi^{-1}(u'_i)$ for all i, $1 \leq i \leq L$. Because case 1 holds, it must be the case that there is at least one index i such that $\varphi^{-1}(u'_i) \neq u_i \varphi^{-1}(f^{r'_i})$ and thus $\mu_i \neq 1$; this means that $\mathcal{B}$ has a non-trivial set $\{\mu_i\}_i$ such that $\prod_i e(g_i, \mu_i) = 1$ and $\prod_i e(\gamma_i, \mu_i) = 1$ and can thus output this set to break the Simultaneous Pairing assumption. By analogous arguments, if case 2 holds then $\mathcal{B}$ can output $\{v_i \varphi^{-1}(h^{s'_i}) / \varphi^{-1}(v'_i)\}_i$, and if case 3 holds then it can output $\{w_i \varphi^{-1}(g^{r'_i}) \varphi^{-1}(g^{s'_i}) / \varphi^{-1}(w'_i)\}_i$. As $\mathcal{B}$ therefore succeeds in each of the cases that $\mathcal{A}$ does, we know that if $\mathcal{A}$ outputs such a tuple (X, W) with non-negligible probability ϵ, $\mathcal{B}$ can break either the Permutation Pairing or Simultaneous Pairing assumption with the same non-negligible probability.

If instead $\mathcal{A}$ outputs (T, X', X), the argument here is very similar to the one above. If the pk and $\{c_i\}_i$ values in X and X' are not the same, then we know that S' will not be satisfied. Otherwise, we know by our outline above that S will be run on the remaining values. By all the same arguments as above, we therefore know that either it really is the case that X (or rather, the value represented by X) was obtained from X' by transformation using T, or $\mathcal{B}$ can use these values to break either the Permutation Pairing or Simultaneous Pairing assumption.

To conclude, the *params* that $\mathcal{B}$ gives to $\mathcal{A}$ are distributed identically to those that $\mathcal{A}$ expects, so interactions with $\mathcal{B}$ are identical to honest interactions and $\mathcal{A}$ should succeed with the same non-negligible probability ϵ. As $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does, we therefore know that $\mathcal{B}$ will succeed with the same probability ϵ in breaking either the Permutation Pairing or Simultaneous Pairing assumption. □

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 Appendix C 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 (given in Appendix B), 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 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

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

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 [6], 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 $(s, I, \pi) = (\perp, \perp, \perp)$, then this is the initial decryption. Compute the share $s_j := u^{a_j} v^{b_j} w$. Now compute $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, (vk_j, c, s_j), (t_{1j}, t_{2j}, open_{1j}, open_{2j}))$, 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 $\perp$ 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. (See Section 4.2 for our construction of cm-NIZKs for partial decryption.)

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 4.2. *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.*

Proof. To prove this, we assume there exists an adversary $\mathcal{A}$ that can, with some non-negligible probability ϵ, produce a tuple (c, m, π) such that $\mathcal{V}(\mathsf{crs}, (pk, \prod_{i \in [N]} vk'_i, c, m), \pi) = 1$ but $m \neq \mathsf{Dec}(\{sk_i\}_{i \in [N]}, c)$, and use it to construct either an adversary $\mathcal{B}$ that breaks the extractability of the proof with some non-negligible probability $\epsilon_{\mathcal{B}}$ or an adversary $\mathcal{C}$ that breaks the binding of the commitment with non-negligible probability $\epsilon_{\mathcal{C}}$. First, consider an alternate game $G^\star$ in which crs is chosen along with a trapdoor τ_e, and after the adversary outputs (c, m, π), we run $w \leftarrow E_2(\mathsf{crs}, \tau_e, (pk, \prod_{i \in [N]} vk'_i, c, m), \pi)$ before evaluating the success condition. Note that since the real and extraction parameters must be indistinguishable, the adversary's success probability in this game can differ only negligibly from ϵ.

In this game, we observe that, if $\mathcal{A}$ produces (c, m, π), one of either two events must take place: for the witness $w \leftarrow E_2(\mathsf{crs}, \tau_e, (pk, \prod_{i \in [N]} vk'_i, c, m), \pi)$ extracted from π, either $(x, w) \notin R$ (if this

occurs and $\mathcal{A}$ succeeds we call this Event_0), or $(x, w) \in R$ (if this occurs and $\mathcal{A}$ succeeds we call this Event_1). If Event_0 occurs with probability e_0 and Event_1 occurs with probability e_1, then we know that $\epsilon = e_0 + e_1$, and thus either e_0 or e_1 is non-negligible; furthermore, we argue that $\mathcal{B}$ succeeds with probability e_0 and $\mathcal{C}$ succeeds with probability e_1, meaning either $\mathcal{B}$ or $\mathcal{C}$ succeeds with non-negligible probability.

If Event_0 occurs with non-negligible probability, consider the following adversary $\mathcal{B}$: it first receives as input (crs, τ_e). It then runs the rest of the code for EncKg honestly, and gives the resulting pk, $vk = (\mathsf{crs}, \mathsf{par}, vk')$, and $\{sk_i\}_i$ to $\mathcal{A}$. When $\mathcal{A}$ outputs (c, m, π), $\mathcal{B}$ outputs $(x := (pk, \prod_{i\in[N]} vk'_i, c, m), \pi)$. Whenever Event_0 takes place, we know that the proof π verifies (because $\mathcal{A}$ succeeds), but for $w \leftarrow E_2(\mathsf{crs}, \tau_e, x, \pi)$, $(x, w) \notin R$; this means that $\mathcal{B}$ wins the extractability game whenever Event_0 occurs. Note that this game is identical to $G^\star$, so Event_0 occurs with probability exactly e_0.

If, on the other hand, Event_1 occurs with non-negligible probability, we construct an adversary $\mathcal{C}$ that first receives as input par. It then generates crs along with the extraction trapdoor τ_e, and then runs the rest of the code for EncKg honestly and gives the resulting pk, $vk = (\mathsf{crs}, \mathsf{par}, vk')$, and $\{sk_i\}_i$ to $\mathcal{A}$. It stores for itself all the $open_{1i}$ and $open_{2i}$ values, as well as the values of $-1/\alpha$ and $-1/\beta$. When $\mathcal{A}$ outputs (c, m, π), $\mathcal{C}$ computes $w = ((t_1, t_2), (open_1, open_2)) \leftarrow E_2(\mathsf{crs}, \tau_e, x := (pk, vk_c := \prod_{i\in[N]} vk'_i, c, m), \pi)$ and outputs $(vk_c, (t_1, t_2, open_1, open_2), (g^{-1/\alpha}, g^{-1/\beta}, \prod_i open_{1i}, \prod_i open_{2i}))$.

To see that this tuple successfully breaks binding whenever Event_1 occurs, we observe that for $\mathcal{A}$ to win it must be the case that $m \neq \mathsf{Dec}(\{sk_i\}_i, c)$; furthermore, we are also assuming (because Event_1 occurred) that $(x, w) \in R$ and thus, according to Equation 1 there exists a value $sk_c = (a, b)$ such that (1) $vk_c = (\mathsf{Com}(t_1; open_1), \mathsf{Com}(t_2; open_2))$, (2) $t_1 = F(a) = g^a$, $t_2 = F(b) = g^b$, and (3) $m = \mathsf{Dec}(sk_c, c)$. Using the way we formed vk and the homomorphic property of the commitment, we already know that

$$vk_c = \prod_{i\in[N]} \left(\mathsf{Com}(g^{a_i}; open_{1i}), \mathsf{Com}(g^{b_i}; open_{2i})\right) = \left(\mathsf{Com}(g^{-1/\alpha}; \prod_i open_{1i}), \mathsf{Com}(g^{-1/\beta}; \prod_i open_{2i})\right).$$

Combining this with property (1) of the relation, we see that $\mathsf{Com}(g^{-1/\alpha}; \prod_i open_{1i}) = \mathsf{Com}(t_1; open_1)$ and $\mathsf{Com}(g^{-1/\beta}; \prod_i open_{2i}) = \mathsf{Com}(t_2; open_2)$. Using property (3) of the relation, we further see that $m = \mathsf{Dec}((a, b), (u, v, w)) = u^a v^b w$; we also know, however, that if $\mathcal{A}$ won then $m \neq \mathsf{Dec}(\{sk_i\}_i, (u, v, w))$, meaning $m \neq u^{-1/\alpha} v^{-1/\beta} w$, and thus $(g^{-1/\alpha}, g^{-1/\beta}) \neq (g^a, g^b)$. We therefore have a commitment vk_c that opens to two values, (g^a, g^b) and $(g^{-1/\alpha}, g^{-1/\beta})$, that are not equal, and thus binding is broken. Furthermore, as $\mathcal{C}$ succeeds whenever Event_1 occurs, it succeeds with probability e_1. □

Lemma 4.3. *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.*

Proof. Assume without loss of generality that the adversary $\mathcal{A}$ corrupts $N - 1$ parties, and thus $|[N] \setminus S_{\mathcal{A}}| = 1$, and consider the following algorithms:

$\mathsf{SimKg}(pk, vk, N, S_{\mathcal{A}})$**:** Let $vk' = (\mathsf{crs}', \mathsf{par}', vk'')$. Let $i^* := [N] \setminus S_{\mathcal{A}}$, and generate $(\mathsf{crs}, \tau_s) \xleftarrow{\$} S_1(1^k)$; we require that crs uses the same bilinear group as crs'. For all $i \in S_{\mathcal{A}}$, pick random values $a_i, b_i \xleftarrow{\$} \mathbb{F}_p$ and set $t_{1i} := g^{a_i}$ and $t_{2i} := g^{b_i}$. Next, choose random openings $open_{1i}$ and $open_{2i}$ for all i, and set $sk_i := (a_i, b_i, t_{1i}, t_{2i}, open_{1i}, open_{2i})$; additionally, form commitments $vk_i := (\mathsf{Com}(t_{1i}; open_{1i}, \mathsf{Com}(t_{2i}; open_{2i}))$ for all i, and set $vk_{i^*} := (\mathsf{Com}(0; open_{1i}), \mathsf{Com}(0; open_{2i}))$. Define $vk := (\mathsf{crs}, \mathsf{par}', \{vk_i\}_i)$ and $\tau := (\tau_s, \{sk_i\}_{i\in S_{\mathcal{A}}})$, and output $(pk, vk, \{sk_i\}_{i\in S_{\mathcal{A}}}, \tau)$.

$\mathsf{SimShareDec}(pk, vk, \tau, (i^*, c, s, I, \pi), m)$**:** Parse $\tau = (\tau_s, \{sk_i\}_{i \in S_\mathcal{A}})$, $c = (u, v, w)$, and parse each secret key as $sk_i = (a_i, b_i, t_{1i}, t_{2i}, open_{1i}, open_{2i})$. Compute $s_{i^*} := m/(w \prod_{j \in S_\mathcal{A}} u^{a_j} v^{b_j})$.

If this is the initial decryption (i.e., $s = \perp$), compute $\pi' \xleftarrow{\$} S_2(\mathsf{crs}, \tau_s, (pk, vk'_{i^*}, c, s_{i^*}))$ and output $(s_{i^*}, \{i^*\}, \pi')$.

Otherwise, compute $s' := s \cdot s_{i^*}$ and $vk'_c := \prod_{i \in I \cup \{i^*\}} vk'_i$. Compute $\pi \xleftarrow{\$} S_2(\mathsf{crs}, \tau_s, (pk, vk'_c, c, s'))$ and output $(s', I \cup \{i^*\}, \pi')$.

With these algorithms in mind, we now argue that they satisfy the definition of share simulatability. We do this through the following series of game transitions:

- Game G_0. The honest share simulatability game for $b = 0$, in which EncKg and Enc are used.
- Game G_1. In Step 2 we switch to using a simulated CRS in pk, and in step 3 we switch to using simulated proofs in the SD oracle only if it is the initial decryption. As this involves switching directly from proofs produced by the honest prover $\mathcal{P}$, under a CRS produced by the honest setup $\mathsf{CRSSetup}$, to proofs produced by the simulator S_2, under a CRS produced by S_1, it is indistinguishable from G_0 by zero knowledge.
- Game G_2. In Step 3 we switch to using simulated proofs in intermediate decryptions as well. As this involves switching directly from proofs produced by ZKEval to ones produced by S_2, it is indistinguishable from G_1 by strong derivation privacy.
- Game G_3. In Step 3, we change the sole honest decryptor (corresponding to the one uncorrupted index i^*) to "fix" the share it outputs to be consistent with the message. More formally, it outputs a share $s = m/w \prod_{j \in S_\mathcal{A}} u^{a_j} v^{b_j}$, as described in $\mathsf{SimShareDec}$ above. To see that this share is in fact identical to the one produced in G_2, observe that if $m = \mathsf{Dec}(\{sk_i\}_{i=1}^N)$ then by definition $m = wu^{-1/\alpha} v^{-1/\beta}$. From the way that a_i and b_i are computed, we know that $w \prod_{j \in [N]} u^{a_i} v^{b_i} = wu^{-1/\alpha} v^{-1/\beta} = m$, and thus $u^{a_{i^*}} v^{b_{i^*}} = m/(w \prod_{j \in S_\mathcal{A}} u^{a_i} v^{b_i})$ and G_2 and G_3 are identical.
- Game G_4. In Step 2, we switch to choosing the a_i and b_i values as we do in SimKg rather than in the honest EncKg; i.e., instead of choosing the first $N - 1$ elements at random and then setting $a_N := -1/\alpha - \sum_{j=0}^{N-1} a_j$ and $b_N := -1/\beta - \sum_{j=0}^{N-1} b_j$, we instead choose $a_i, b_i \xleftarrow{\$} \mathbb{F}_p$ for $i \in S_\mathcal{A}$ and then set $a_{i^*} := -1/\alpha - \sum_{j \in S_\mathcal{A}} a_j$ and $b_{i^*} := -1/\beta - \sum_{j \in S_\mathcal{A}} b_j$. Again, Games G_3 and G_4 are identical.
- Game G_5. In Step 2, we change from using $vk_{i^*} = (\mathsf{Com}(g^{a_i}; open_{1i}), \mathsf{Com}(g^{b_i}; open_{2i}))$ to using $vk_{i^*} = (\mathsf{Com}(0; open_{1i}), \mathsf{Com}(0; open_{2i}))$. By the hiding property of the commitment scheme, and the fact that the commitment openings were unused as all proofs are simulated, this is indistinguishable from G_4. Furthermore, this is now the honest share simulatability game for $b = 1$, in which SimKg and $\mathsf{SimShareDec}$ are used.

As each game was either indistinguishable from or identical to the preceding one, we can therefore conclude that G_0 is indistinguishable from G_5, and thus an adversary cannot distinguish with more than negligible probability between the game for $b = 0$ and for $b = 1$. □

4.2 A cm-NIZK for partial decryption

Given our concrete choices in the threshold encryption construction above, we must also show how to construct a strongly derivation-private malleable NIZKPoK for the relation R defined in Equation 1. In fact, to use the same framework as in Section 3, we instead do something strictly stronger and show how to construct a cm-NIZK for this relation. We note that, for greater efficiency, one might instead directly construct a malleable NIZKPoK from Groth-Sahai proofs. (For example, one could use the

following approach: include a commitment in the CRS and use Groth-Sahai to prove knowledge of an opening to the commitment *or* a valid witness, in a manner similar to that of the original FLS approach [15]. Intuitively, by the CKLM result on the malleability of Groth-Sahai OR proofs, the resulting proof is malleable using the same approach as below. It is furthermore zero knowledge and strongly derivation private because a simulator can use the opening to the commitment to form all the proofs and Groth-Sahai proofs are witness indistinguishable, and a proof of knowledge by the hiding property of the commitment scheme.) On the other hand, there may also be stronger definitions of threshold decryption for which cm-NIZKs would be necessary; we leave examination of such definitions as an open problem.

To now show that we can construct a cm-NIZK for this relation, we use the same outline as we did in Section 3; first, we show that we can satisfy the properties of CM-friendliness. Then, we can directly apply the results of CKLM to construct a cm-NIZK secure under the DLIN assumption.

- $\mathsf{CRSSetup}(1^k)$: Generate and output a prime-order bilinear group (p, G, G_T, e, g).
- Representable statements. Instances are of the form (vk_c, c, s), where vk_c is a pair of commitments (A, B); as seen above, all these values (as well as the ciphertext c and decryption share s) are group elements. Witnesses, $t_i = (g^{a_i}, g^{b_i})$ and $open_i = (open_{1i}, open_{2i})$, are group elements as well.
- Representable transformations. Transformations are represented by $T = (\hat{s}, \hat{t}, \widehat{open})$. As these are of the same form as elements in instances/witnesses, we can represent them as group elements.
- Provable statements. To prove that the share s is a correct partial decryption of the ciphertext $c = (u, v, w)$ performed using the cumulative secret key corresponding to the tokens t_1 and t_2, committed to in A and B with openings $open_1$ and $open_2$ respectively, we can use the following set S (with Equations 2 and 3 expanded into the pairing product equations for verifying commitments as in previous work):

  $$(1)\ e(s,g) = e(u,t_1)\cdot e(v,t_2)\cdot e(w,g), \quad (2)\ A = \mathsf{Com}(t_1; open_1), \text{ and } \quad (3)\ B = \mathsf{Com}(t_2; open_2).$$

- Provable transformations. To prove $T_{\text{inst}}(x') = x$ for $T \in \mathcal{T}$, we must first prove that T_{inst} does not change c; if we represent $x = (vk_c = (A, B), c, s)$ and $x' = (vk'_c = (A', B'), c', s')$, this means checking that $c = c'$. We also want to check that for $T = (\hat{s}, (\hat{t_1}, \hat{t_2}), (\widehat{open_1}, \widehat{open_2}))$, it is the case that vk_c incorporates the new commitments, t incorporates the new tokens, s incorporates the new share, and $\hat{s}$ is a valid share. This means that S' consists of the following equations:

  $$(1)\ e(c,g) = e(c',g), \quad (2)\ A = A'\cdot \mathsf{Com}(\hat{t_1}; \widehat{open_1}), \quad (3)\ B = B'\cdot \mathsf{Com}(\hat{t_2}; \widehat{open_2}),$$
  $$(4)\ e(s,g) = e(s',g)\cdot e(\hat{s},g), \text{ and } \quad (5)\ e(\hat{s},g) = e(u,\hat{t_1})\cdot e(v,\hat{t_2}).$$

- Transformable statements. To change the statement $(x, w) \in R$ into $(T_{\text{inst}}(x), T_{\text{wit}}(w)) \in R$, we use the fact that CKLM already showed that multiplying in constants is a valid transformation; i.e., is allowed for Groth-Sahai proofs. For $T = (\hat{s}, (\hat{t_1}, \hat{t_2}), (\widehat{open_1}, \widehat{open_2}))$, we therefore add $e(\hat{s}, g)$ to the left-hand side of Equation 1 in S and $e(u, \hat{t_1})\cdot e(v, \hat{t_2})$ to the right-hand side. We similarly change the commitment equations by adding $\mathsf{Com}(\hat{t_1}; \widehat{open_1})$ and $\mathsf{Com}(\hat{t_2}; \widehat{open_2})$ to the right-hand sides of Equations 2 and 3 respectively.
- 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 equality check for c (Equation 1 in S') alone. For $T' = (\tilde{s}, (\tilde{t_1}, \tilde{t_2}), (\widehat{open_1}, \widehat{open_2}))$, we change the commitment equations by multiplying in $\mathsf{Com}(\tilde{t_1}; \widehat{open_1})$ and $\mathsf{Com}(\tilde{t_2}; \widehat{open_2})$ to both sides of Equations 2 and 3 respectively. Next, we multiply $e(\tilde{s}, g)$ into both

sides of Equation 4, $e(\tilde{s}, g)$ into the left-hand side of Equation 5, and $e(u, \tilde{t_1}) \cdot e(v, \tilde{t_2})$ into the right-hand side of Equation 5.

If we use Groth-Sahai proofs for our sets S and S' of pairing product equations, then we can achieve perfect extractability. In particular, unlike in Section 3, our representations of instances, witnesses, and transformations are now *perfect*; this means the entire proof system is perfectly extractable and thus we 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 perform $(s'_i, I \cup \{k\}, \phi'_i) \xleftarrow{\$} \mathsf{ShareDec}(pk, vk, dk_k, c_i, (s_i, I, \phi_i))$ 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; formal proofs of their indistinguishability can be found in Appendix D:

- Game G_0. The honest game with $b = 0$.
- Game G_1. In the setup phase, we switch to using SimKg to generate the threshold keys, and in the tallying phase we switch to using $\mathsf{SimShareDec}$. This is indistinguishable from G_0 by share simulatability (Definition 2.4).
- Game G_2. In the tallying phase, we switch to giving $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ to $\mathsf{SimShareDec}$ instead of $m_i := \mathsf{Dec}(sk, c'_i)$ for all i, where φ is a random permutation. This is indistinguishable from G_1 by compact verifiability (Definition 2.3).
- Game G_3. In the voting phase, we switch to using simulated proofs of knowledge π_i. This is indistinguishable from G_2 by zero knowledge.

- Game G_4. In the tallying phase, we switch to using $m_i := \mathsf{Extract}(\tau_e, \varphi(\pi_i))$ for the adversarially-generated ballots (c_i, π_i); i.e., the value extracted from the permuted proof of knowledge. This is indistinguishable from G_3 by simulation-sound extractability (Definition A.2).
- Game G_5. In the tallying phase, we switch to using $m_i := \varphi(v_{i0})$, where, for the i-th ballot, the adversary queried (v_{i0}, v_{i1}) to its oracle. This is identical to G_4 by the correctness of the encryption scheme.
- Game G_6. In the tallying phase, we switch to using $m_i := \varphi(v_{i1})$, where, for the i-th ballot, the adversary queried (v_{i0}, v_{1i}) to its oracle. This is identical to G_5 because $\{v_{i0}\}_i$ and $\{v_{i1}\}_i$ are guaranteed to lead to the same election result.
- Game G_7. In the voting phase, we switch to using $\mathsf{Enc}(pk, v_1)$ instead of $\mathsf{Enc}(pk, v_0)$. This is indistinguishable from G_6 by IND-CPA security.
- Game G_8. In the tallying phase, we switch back to using $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ instead of v_{1i} for the ballots formed by query. This is again identical to G_7 by the correctness of the encryption scheme.
- Game G_9. In the tallying phase, we switch back to using $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ instead of $m_i := \mathsf{Extract}(\tau_e, \varphi(\pi_i))$ for the adversarially-formed ballots. This is again indistinguishable from G_8 by simulation-sound extractability.
- Game G_{10}. In the voting phase, we switch back to using honest proofs of knowledge; this is again indistinguishable from G_9 by zero knowledge.
- Game G_{11}. In the tallying phase, we switch back to using $m_i := \mathsf{Dec}(sk, c'_i)$ instead of $m_i := \mathsf{Dec}(sk, \varphi(c_i))$. This is again indistinguishable from G_{10} by compact verifiability.
- Game G_{12}. In the setup phase, we switch back to using EncKg and $\mathsf{ShareDec}$ instead of SimKg and $\mathsf{SimShareDec}$. This is again indistinguishable from G_{11} by share simulatability, and is now the honest game for $b = 1$.

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] M. Blum, A. de Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[6] 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.

[7] 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.

[8] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. In *Proceedings of Eurocrypt 2012*, pages 281–300, 2012.

[9] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. Cryptology ePrint Archive, Report 2012/012, 2012. `http://eprint.iacr.org/2012/012`.

[10] 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`.

[11] I. Damgård. On sigma protocols. `http://www.daimi.au.dk/~ivan/Sigma.pdf`.

[12] 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.

[13] 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.

[14] Y. Desmedt and Y. Frankel. Threshold cryptography. In *CRYPTO '89*, volume 435 of *LNCS*, pages 307–315. Springer-Verlag, 1990.

[15] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal of Computing*, 29(1):1–28, 1999.

[16] 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.

[17] 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.

[18] J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 145–160. Springer-Verlag, 2003.

[19] J. Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS*, volume 3531 of *LNCS*, pages 467–482. Springer-Verlag, 2005.

[20] 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.

[21] 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.

[22] J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero-knowledge for NP. In *Proceedings of Eurocrypt 2006*, volume 4004 of *LNCS*, pages 339–358. Springer-Verlag, 2006.

[23] 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.

[24] 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.

[25] 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.

[26] D. Sandler, K. Derr, and D. S. Wallach. Votebox: A tamper-evident, verifiable electronic voting system. In *USENIX Security Symposium*, pages 349–364, 2008.

[27] 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.

A Non-Interactive Proof Systems

Definition A.1. [8] *A set of algorithms* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$[2] *constitute a non-interactive (NI) proof system for an efficient relation* R *with associated language* L_R *if completeness and soundness below*

[2] Although we deal here with the standard definition of a non-interactive proof, using these three algorithms, we mention that the definitions can be easily extended to consider the malleable proofs $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ defined in Section 2. In fact, the definitions can simply ignore the ZKEval algorithm and are thus unmodified.

are satisfied. A NI proof system is extractable if, in addition, the extractability property below is satisfied. A NI proof system is witness-indistinguishable (NIWI) if the witness-indistinguishability property below is satisfied. An NI proof system is zero-knowledge (NIZK) if the zero-knowledge property is satisfied. A NIZK proof system that is also extractable constitutes a non-interactive zero-knowledge proof of knowledge (NIZKPoK) system. A NIWI proof system that is also extractable constitutes a non-interactive witness-indistinguishable proof of knowledge (NIWIPoK) system.

1. *Completeness [5]. For all* $\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ *and* $(x, w) \in R$, $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ *for all proofs* $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, x, w)$.
2. *Soundness [5]. For all PPT* $\mathcal{A}$, *and for* $\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$, *the probability that* $\mathcal{A}(\mathsf{crs})$ *outputs* (x, π) *such that* $x \notin L$ *but* $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ *is negligible.* Perfect *soundness is achieved when this probability is 0.*
3. *Extractability [22]. There exists a PPT extractor* $E = (E_1, E_2)$ *such that* $E_1(1^k)$ *outputs* (σ_{ext}, τ_e), *and* $E_2(\sigma_{ext}, \tau_e, x, \pi)$ *outputs a value* w *such that (1) any PPT* $\mathcal{A}$ *given* σ *cannot distinguish between the honest CRS and one output by* E_1; *i.e.,*

$$Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k) : \mathcal{A}(\mathsf{crs}) = 1] \approx Pr[(\sigma_{ext}, \tau_e) \xleftarrow{\$} E_1(1^k) : \mathcal{A}(\sigma_{ext}) = 1], \text{ and}$$

   and (2) for all PPT $\mathcal{A}$, *the probability that* $\mathcal{A}$ *outputs* (x, π) *such that* $\mathcal{V}(\sigma_{ext}, x, \pi) = 1$ *but* $R(x, E_2(\sigma_{ext}, \tau_e, x, \pi)) = 0$ *is negligible; i.e., there exists a negligible function* $\nu(\cdot)$ *such that*

$$Pr[(\sigma_{ext}, \tau_e) \xleftarrow{\$} E_1(1^k); (x, \pi) \xleftarrow{\$} \mathcal{A}(\sigma_{ext}) : \mathcal{V}(\sigma_{ext}, x, \pi) = 1 \wedge (x, E_2(\sigma_{ext}, \tau_e, x, \pi)) \notin R] < \nu(k).$$

   Perfect *extractability is achieved if this probability is 0, and* σ_{ext} *is distributed identically to* crs.
4. *Witness indistinguishability [15]. For all* (x, w_1, w_2) *such that* $(x, w_1), (x, w_2) \in R$, *any PPT* $\mathcal{A}$ *cannot distinguish between proofs for* w_1 *and proofs for* w_2; *i.e.,*

$$\begin{aligned} &Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k); (x, w_1, w_2) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}); \pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, x, w_0) : \mathcal{A}(\pi) = 1 \wedge (x, w_0), (x, w_1) \in R] \\ \approx &Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k); (x, w_1, w_2) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}); \pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, x, w_1) : \mathcal{A}(\pi) = 1 \wedge (x, w_0), (x, w_1) \in R]. \end{aligned}$$

   Perfect *witness indistinguishability is achieved when these two distributions are identical.*
5. *Zero knowledge [15]. There exists a polynomial-time simulator algorithm* $S = (S_1, S_2)$ *such that* $S_1(1^k)$ *outputs* (σ_{sim}, τ_s), *and* $S_2(\sigma_{sim}, \tau_s, x)$ *outputs a value* π_s *such that for all* $(x, w) \in R$, *a PPT adversary* $\mathcal{A}$ *cannot distinguish between proofs produced by the prover and simulator; i.e., for all PPT adversaries* $\mathcal{A}$,

$$Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k) : \mathcal{A}^{P(\mathsf{crs}, \cdot, \cdot)}(\mathsf{crs}) = 1] \approx Pr[(\sigma_{sim}, \tau_s) \xleftarrow{\$} S_1(1^k) : \mathcal{A}^{S(\sigma_{sim}, \tau_s, \cdot, \cdot)}(\sigma_{sim}) = 1],$$

   where, on input (x, w), P *outputs* $\perp$ *if* $(x, w) \notin R$ *and* $\pi \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, x, w)$ *otherwise, and* S *also outputs* $\perp$ *if* $(x, w) \notin R$, *and returns* $\pi \xleftarrow{\$} S_2(\sigma_{sim}, \tau_s, x)$ *otherwise.* Perfect *zero knowledge is achieved if for all* $(x, w) \in R$, *these distributions are identical.*

Additionally, we recall the notion of simulation-sound extractability, which combines extractability and zero knowledge.

Definition A.2. [20] *Let* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *be a NIZKPoK system for an efficient relation* R *with a simulator* (S_1, S_2) *and an extractor* (E_1, E_2), *and let* SE_1 *be an algorithm that outputs* $(\mathsf{crs}, \tau_s, \tau_e)$ *such that these first two values are distributed identically to the* (crs, τ_s) *output by* S_1. *Then 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.* $w \leftarrow E_2(\mathsf{crs}, \tau_e, x, \pi)$.

Then the NIZKPoK satisfies simulation-sound extractability *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 $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$ but $(x, \pi) \notin Q$ (where Q is the set of S_2 queries and their responses) and $(x, w) \notin R$ is at most $\nu(k)$.*

B The Shuffle Construction of CKLM

Recall from Section 2.2 that a compactly verifiable shuffle is a tuple $(\mathsf{Setup}, \mathsf{ShuffleKg}, \mathsf{Shuffle}, \mathsf{Verify})$, defined with respect to an re-randomizable encryption scheme $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$. The generic construction given by CKLM of such a shuffle combines a hard relation with generator $\mathcal{G}$, a proof of knowledge $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$, and a cm-NIZK $(\mathsf{CRSSetup}', \mathcal{P}', \mathcal{V}')$:

- $\mathsf{Setup}(1^k)$: Generate $\mathsf{crs} \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ and $\mathsf{crs}' \xleftarrow{\$} \mathsf{CRSSetup}'(1^k)$. Output $params := (\mathsf{crs}, \mathsf{crs}')$.
- $\mathsf{ShuffleKg}(params)$: For each mix authority i, generate $(pk_i, sk_i) \xleftarrow{\$} \mathcal{G}(1^k)$. Output $(\{pk_i\}_i, \{sk_i\}_i)$.
- $\mathsf{Enc}(params, pk, \{m_i\}_{i=1}^n)$. Parse $params = (\mathsf{crs}, \mathsf{crs}')$. Each user i now picks randomness r_i and encrypts his message m_i as $c_i := \mathsf{Enc}(pk, m_i; r_i)$; he also forms a proof of knowledge $\pi_i \xleftarrow{\$} \mathcal{P}(\mathsf{crs}, (pk, c_i), (m_i, r_i))$. Aggregated across all users, this produces $\{(c_i, \pi_i)\}_{i=1}^n$.
- $\mathsf{Shuffle}(params, \{c_i, \pi_i\}_i, \{c'_i\}_i, \pi, \{pk_j\}_j)$: First check if this is the initial shuffle by checking if $\pi = \perp$ and $\{c'_i\}_i = \{pk_j\}_j = \emptyset$. If it is, check that $\mathcal{V}(\mathsf{crs}, c_i, \pi_i) = 1$ for all i, $1 \le i \le n$; if this check fails for some i, abort and output $\perp$. Otherwise, pick a random permutation $\varphi \xleftarrow{\$} S_n$ and compute $c'_i \xleftarrow{\$} \mathsf{ReRand}(pk, \varphi(c_i))$ for all i. Next, form a proof π for the shuffle performed by the user in possession of the secret key corresponding to pk_1 (i.e., the initial mix server). Output $(\{c_i, \pi_i\}_i, \{c'_i\}_i, \pi, \{pk_1\})$.
  Otherwise, if this is not the initial mix server, check that $\mathcal{V}'(\mathsf{crs}', (pk, (\{c_i\}_i, \{c'_i\}_i, \{pk_j\}_j), \pi) = 1$; if this check fails abort and output $\perp$. Otherwise, continue by choosing a random permutation $\varphi \xleftarrow{\$} S_n$ and randomness $\{R_i\}_i$ for the encryption scheme, and computing $c''_i \xleftarrow{\$} \mathsf{ReRand}(pk, \varphi(c'_i); R_i)$ for all i. If the public key for the current mix server is pk_k define $T := T_{(\varphi, \{R_i\}_i, \{sk_k, pk_k\}, \emptyset)}$ and compute $\pi' \xleftarrow{\$} \mathsf{ZKEval}(\mathsf{crs}', T, (pk, (\{c_i\}_i, \{c'_i\}_i, \{pk_j\}_j), \pi)$. Output $(\{c_i\}_i, \{c''_i\}_i, \pi', \{pk_j\}_j \cup \{pk_k\})$.
- $\mathsf{Verify}(params, \{c_i, \pi_i\}, \{c'_i\}, \pi, \{pk_j\})$: Check that $\mathcal{V}(\mathsf{crs}, c_i, \pi_i) = 1$ for all i, $1 \le i \le n$; if this fails for any i abort and return 0. Otherwise, check that $\mathcal{V}'(\mathsf{crs}', (\{c_i\}, \{c'_i\}, \{pk_j\}), \pi) = 1$; again, if this fails output 0 and otherwise output 1.

As proved by CKLM, this construction constitutes a secure compactly verifiable shuffle (as defined in Definition 2.3).

C Constructing a cm-NIZK Using CM-Friendliness

In Section 3, we saw our modified definition for CM-friendliness that weakened the CKLM definition to allow for computationally sound proofs in addition to ones that are perfectly sound. As we would like to argue that, for a relation R and transformation class $\mathcal{T}$, any instantiation $(S, S', \mathsf{CRSSetup})$

meeting our definition still allows us to construct a cm-NIZK, we prove here a modified version of the corresponding theorem from the full CKLM paper [9, Theorem 4.1]:

Theorem C.1. *Given a derivation-private NIWIPoK* $(\mathsf{CRSSetup}_{\mathsf{pp}}, \mathcal{P}_{\mathsf{pp}}, \mathcal{V}_{\mathsf{pp}}, \mathsf{ZKEval}_{\mathsf{pp}})$ *that is malleable for the set of all valid transformations and a structure-preserving signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*, if* $(S, S', \mathsf{CRSSetup})$ *is a CM-friendly instantiation for some relation and transformation class* $(R, \mathcal{T})$ *then we can construct a cm-NIZK for* $(R, \mathcal{T})$*.*

Proof. Let R' be the relation

$$\{((x, vk), (w, x', T, \sigma)) \mid (x, w) \in R \ \vee \ (\mathsf{Verify}(vk, \sigma, x') = 1 \wedge x = T_{\text{inst}}(x') \wedge T \in \mathcal{T})\}$$

needed by the generic construction of CKLM; our goal is to embed this relation into a set of pairing product equations using the underlying sets S and S'. By the definition of S and S' and CM-friendliness, we know that $(x, w) \in R$ implies that $F_s(params, x)$ and $F_w(params, w)$ satisfy S (for $params \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$), and if values X and W satisfy S then with overwhelming probability $(F_s^{-1}(params, X), F_w^{-1}(params, W)) \in R$. We similarly have that if $x = T_{\text{inst}}(x')$ for $T \in \mathcal{T}$ then $F_s(params, x)$, $F_s(params, x')$, and $F_t(params, T)$ satisfy S', and that the reverse direction is also true with overwhelming probability. Finally, by the definition of a structure preserving signature, there exists a set of pairing product equations S_σ such that $\mathsf{Verify}(vk, \sigma, x') = 1$ if and only if vk, $F_s(params, x')$, and σ satisfy S_σ.

Looking at these three equations, we observe that S_σ and S' share the variables in $F_s(params, x')$; without loss of generality, we assume that these are the first n variables in each. We then set

$$x_{\mathsf{pp}} := \mathsf{Or}(S(F_s(params, x)), \mathsf{And}(S_\sigma(vk), S'(F_s(params, x)); n)) \tag{2}$$

and

$$\begin{aligned} w_{\mathsf{pp}} := \mathsf{Or}_w(S(F_s(params, x)), \mathsf{And}(S_\sigma(vk), S'(F_s(params, x)); n), F_w(params, w), \\ \mathsf{And}_w((F_s(params, x'), \sigma), (F_s(params, x'), F_t(params, T), \sigma), n)). \end{aligned} \tag{3}$$

Using such an instance and witness, it is straightforward to check that if $((x, vk), (w, x', T, \sigma)) \in R'$ then $(x_{\mathsf{pp}}, w_{\mathsf{pp}}) \in R_{\mathsf{pp}}$, where R_{pp} consists of all statements than can be proved using pairing product equations. We can therefore implement a NIWIPoK for R' as follows:

- $\mathsf{CRSSetup}(1^k)$: Compute $\mathsf{crs}_{\mathsf{pp}} \xleftarrow{\$} \mathsf{CRSSetup}_{\mathsf{pp}}(1^k)$ and $params \xleftarrow{\$} \mathsf{CRSSetup}(1^k)$ and output $\mathsf{crs} := (\mathsf{crs}_{\mathsf{pp}}, params)$.
- $\mathcal{P}(\mathsf{crs}, x, (w, x', T, \sigma))$: Let x_{pp} and w_{pp} be as defined in Equations 2 and 3 respectively, and return $\pi \xleftarrow{\$} \mathcal{P}_{\mathsf{pp}}(\mathsf{crs}_{\mathsf{pp}}, x_{\mathsf{pp}}, w_{\mathsf{pp}})$.
- $\mathcal{V}(\mathsf{crs}, x, \pi)$: Again, let x_{pp} be as defined in Equation 2, and output $\mathcal{V}_{\mathsf{pp}}(\mathsf{crs}_{\mathsf{pp}}, x_{\mathsf{pp}}, \pi)$.
- $\mathsf{ZKEval}(\mathsf{crs}, T, x, \pi)$: Let $T_s := s(T)$ and $T_t := \mathsf{Lift}(\mathrm{id}, t(T))$, where $s(T)$ and $t(T)$ are the respective transformations on statements and transformations defined by CM-friendliness. Output $\mathsf{ZKEval}_{\mathsf{pp}}(\mathsf{crs}, \mathsf{LR}(T_s, T_t), \pi)$.

In their Theorems B.1 and B.3 [9], CKLM show that $\mathsf{LR}(T_s, T_t)$ is a valid transformation that turns an instance x_{pp} into an instance

$$x'_{\mathsf{pp}} := \mathsf{Or}(s(T)(S(F_s(params, x))), \mathsf{And}(S_\sigma(vk), t(T)(S'(F_s(params, x)); n))).$$

The properties of Or and And further guarantee that this $x'_{\mathsf{pp}} \in L_{\mathsf{pp}}$ if there exists a tuple (w, x', T', σ) such that $((T_{\text{inst}}(x), vk), (w, x', T', \sigma)) \in R'$; in other words, for every $T := (T_{\text{inst}}, T_{\text{wit}}) \in \mathcal{T}$, $\mathsf{LR}(T_s, T_t)$ realizes a transformation $T' \in \mathcal{T}'$.

Putting everything together, we see that the proof system described above is indeed a derivation-private NIWIPoK for the relation R', malleable with respect to the class $\mathcal{T}'$. The theorem then follows by the proofs of security (Theorems 3.2, 3.3, and 3.4) that CKLM provide for their generic cm-NIZK construction. □

D A Proof of Voting Security (Theorem 5.1)

We follow the proof outline described in Section 5, which we replicate here for convenience:

- Game G_0. The honest game with $b = 0$.
- Game G_1. In the setup phase, we switch to using SimKg to generate the threshold keys, and in the tallying phase we switch to using $\mathsf{SimShareDec}$. This is indistinguishable from G_0 by share simulatability (Definition 2.4).
- Game G_2. In the tallying phase, we switch to giving $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ to $\mathsf{SimShareDec}$ instead of $m_i := \mathsf{Dec}(sk, c'_i)$ for all i, where φ is a random permutation. This is indistinguishable from G_1 by compact verifiability (Definition 2.3).
- Game G_3. In the voting phase, we switch to using simulated proofs of knowledge π_i. This is indistinguishable from G_2 by zero knowledge.
- Game G_4. In the tallying phase, we switch to using $m_i := \mathsf{Extract}(\tau_e, \varphi(\pi_i))$ for the adversarially-generated ballots (c_i, π_i); i.e., the value extracted from the permuted proof of knowledge. This is indistinguishable from G_3 by simulation-sound extractability (Definition A.2).
- Game G_5. In the tallying phase, we switch to using $m_i := \varphi(v_{i0})$, where, for the i-th ballot, the adversary queried (v_{i0}, v_{i1}) to its oracle. This is identical to G_4 by the correctness of the encryption scheme.
- Game G_6. In the tallying phase, we switch to using $m_i := \varphi(v_{i1})$, where, for the i-th ballot, the adversary queried (v_{i0}, v_{1i}) to its oracle. This is identical to G_5 because $\{v_{i0}\}_i$ and $\{v_{i1}\}_i$ are guaranteed to lead to the same election result.
- Game G_7. In the voting phase, we switch to using $\mathsf{Enc}(pk, v_1)$ instead of $\mathsf{Enc}(pk, v_0)$. This is indistinguishable from G_6 by IND-CPA security.
- Game G_8. In the tallying phase, we switch back to using $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ instead of v_{1i} for the ballots formed by query. This is again identical to G_7 by the correctness of the encryption scheme.
- Game G_9. In the tallying phase, we switch back to using $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ instead of $m_i := \mathsf{Extract}(\tau_e, \varphi(\pi_i))$ for the adversarially-formed ballots. This is again indistinguishable from G_8 by simulation-sound extractability.
- Game G_{10}. In the voting phase, we switch back to using honest proofs of knowledge; this is again indistinguishable from G_9 by zero knowledge.
- Game G_{11}. In the tallying phase, we switch back to using $m_i := \mathsf{Dec}(sk, c'_i)$ instead of $m_i := \mathsf{Dec}(sk, \varphi(c_i))$. This is again indistinguishable from G_{10} by compact verifiability.
- Game G_{12}. In the setup phase, we switch back to using EncKg and $\mathsf{ShareDec}$ instead of SimKg and $\mathsf{SimShareDec}$. This is again indistinguishable from G_{11} by share simulatability, and is now the honest game for $b = 1$.

So, we first switch in Game G_1 to using SimKg and $\mathsf{SimShareDec}$ instead of EncKg and $\mathsf{ShareDec}$, and argue that this is indistinguishable by the security of the threshold decryption.

Lemma D.1. *If the threshold encryption scheme* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify})$ *satisfies share simulatability, as defined in Definition 2.4, then Game* G_1 *is indistinguishable from* G_0.

Proof. To prove this, we show that if there exists an adversary $\mathcal{A}$ that distinguishes between G_0 and G_1 with some non-negligible advantage ϵ then we can use it to construct an adversary $\mathcal{B}$ that wins at the threshold security game with the same advantage.

To start, $\mathcal{A}$ specifies the sets S and T of mix and decryption authorities it wants to corrupt. $\mathcal{B}$ then specifies the same set S in its own game and gets $(pk, vk, \{sk_i\}_{i \in S})$ in return. It then generates all the parameters for the shuffle itself, and gives all the appropriate values to $\mathcal{A}$. During the voting and ballot processing phases, $\mathcal{B}$ behaves completely honestly (by encrypting v_0 and performing the honest shuffle). During the tallying phase, if $\mathcal{A}$ queries the $\mathsf{ShareDec}$ oracle on input (j, k, s_k, I, ϕ_k), $\mathcal{B}$ sends the query (j, c_k, s_k, I, ϕ_k) to its SD oracle and publishes the resulting $(s', I \cup \{j\}, \pi')$ to the bulletin board. Finally, if $\mathcal{A}$ guesses G_0 then $\mathcal{B}$ guesses $b = 0$, and if $\mathcal{B}$ guesses G_1 then $\mathcal{B}$ guesses $b = 1$.

To see that interactions with $\mathcal{B}$ are indistinguishable from those that $\mathcal{A}$ expects, we observe that the only phase in which $\mathcal{B}$ is not behaving completely honestly is the tallying phase. Here, if the keys $\mathcal{B}$ gave to $\mathcal{A}$ were generated by EncKg and its SD oracle is using $\mathsf{ShareDec}$ then $\mathcal{B}$ is executing the exact code of G_0, while if the keys were generated by SimKg and the SD oracle is using $\mathsf{SimShareDec}$, it is executing the exact code of G_1. $\mathcal{A}$ will therefore have the same advantage ϵ when interacting with $\mathcal{B}$; as $\mathcal{B}$ furthermore guesses correctly whenever $\mathcal{A}$ does, $\mathcal{B}$ will succeed at the threshold security game with the same advantage. □

Next, we switch in Game G_2 to using $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ in the share decryption oracle instead of $m_i := \mathsf{Dec}(sk, c_i')$, where φ is a random permutation.

Lemma D.2. *If the shuffle* $(\mathsf{Setup}, \mathsf{ShuffleKg}, \mathsf{Shuffle}, \mathsf{Verify})$ *is compactly verifiable, as defined in Definition 2.3, then Game* G_2 *is indistinguishable from* G_1*.*

Proof. To prove this, we show that if there exists an adversary $\mathcal{A}$ that distinguishes between G_1 and G_2 with some non-negligible advantage ϵ then we can use it to construct an adversary $\mathcal{B}$ that wins at the compact verifiability game with the same advantage.

To start, $\mathcal{B}$ gets as input *params*, the encryption key pk, and $T := \{pk_i\}$. It now generates all the threshold parameters using $\mathsf{SimKg}(pk, N, S)$, and gives the parameters and the appropriate keys for the corrupted parties to $\mathcal{A}$. **MC: Our threshold enc definition currently gives vk' to SimKg. Can we just remove that? It sort of makes things tricky here because we'd have to come up with some correctly distributed vk' to pass in.** In the voting phase, $\mathcal{B}$ behaves completely honestly. In the ballot processing phase, when $\mathcal{A}$ queries its initial shuffle oracle $\mathcal{B}$ will query its own initial shuffle oracle, and when $\mathcal{A}$ queries the regular shuffle oracle $\mathcal{B}$ will query its own shuffle oracle. When the final shuffle $(\{c_i'\}, \pi, \{pk_j\})$ gets posted, $\mathcal{B}$ outputs $(\{c_i, \pi_i\}, \{c_i'\}, \pi, \{pk_j\})$. If the game continues (i.e., this tuple is valid), then $\mathcal{B}$ gets back some collection $\{m_i\}$. In the tallying phase, when $\mathcal{B}$ is queried on a tuple (j, k, s_k, I, ϕ_k), it computes the code of $\mathsf{SimShareDec}$ honestly, using m_k as input. At the end, if $\mathcal{A}$ guesses G_1 then $\mathcal{B}$ guesses $b = 0$, and if $\mathcal{A}$ guesses G_2 then $\mathcal{B}$ guesses $b = 1$.

To see that interactions with $\mathcal{B}$ are indistinguishable from the ones that $\mathcal{A}$ expects, we observe that the parameters $\mathcal{B}$ gives to $\mathcal{A}$ are identical to what they would be in either game, as is its behavior in the voting phase. As for the shuffle, $\mathcal{B}$'s behavior is again identical to both games. As for the tallying phase, if $b = 0$ then $\mathcal{B}$ gets back $\{m_i := \mathsf{Dec}(sk, c_i')\}$, which means it is executing the exact code of G_1. Similarly, if $b = 1$ then $\mathcal{B}$ gets back $\{m_i := \mathsf{Dec}(sk, \varphi(c_i))$ for a random permutation, which means it is executing the exact code of G_2. As $\mathcal{A}$ will therefore have the same advantage ϵ when interacting with $\mathcal{B}$, and furthermore $\mathcal{B}$ guesses correctly whenever $\mathcal{A}$ does, $\mathcal{B}$ will succeed with advantage ϵ. □

Next, we switch in Game G_3 to using simulated proofs of knowledge in the voting phase.

Lemma D.3. *If the proof* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *is zero knowledge, then Game* G_3 *is indistinguishable from* G_2*.*

Proof. To prove this, we show that if there exists an adversary $\mathcal{A}$ that distinguishes between G_2 and G_3 with some non-negligible advantage ϵ then we can use it to construct an adversary $\mathcal{B}$ that breaks zero knowledge with the same advantage.

To start, $\mathcal{B}$ gets as input a CRS crs. It then forms all the other parameters as in G_2, and gives the parameters and the appropriate keys for the corrupted parties to $\mathcal{A}$. In the voting phase, if $\mathcal{A}$ outputs a query of the form (v_0, v_1), $\mathcal{B}$ will pick some randomness r for the encryption scheme and form $c \xleftarrow{\$} \mathsf{Enc}(pk, v_0)$. It then queries its own oracle on $((pk, c), (v_0, r))$ to get back a proof π, and returns (c, π) to $\mathcal{A}$. In both the ballot processing and tallying phases, $\mathcal{B}$ behaves exactly as in both G_2 and in G_3. At the end of the game, if $\mathcal{A}$ guesses G_2 then $\mathcal{B}$ guesses it is interacting with the prover on an honest CRS, and if $\mathcal{A}$ guesses G_3 then $\mathcal{B}$ guesses it is interacting with the simulator on a simulated CRS.

To see that interactions with $\mathcal{B}$ are identical to the ones that $\mathcal{A}$ expects, we observe that the value of the CRS and the proofs π are the only way in which $\mathcal{B}$ is not identical to both games. Furthermore, if the CRS is an honest CRS and the proofs π are formed by the prover then $\mathcal{B}$ is executing the exact code of G_2, while if the CRS and proofs are both simulated then $\mathcal{B}$ is executing the exact code of G_3. $\mathcal{A}$ will therefore have the same advantage ϵ when interacting with $\mathcal{B}$; furthermore, as $\mathcal{B}$ guesses correctly whenever $\mathcal{A}$ does, it guesses correctly with the same advantage. □

Next, we switch in Game G_4 to extracting from the proofs rather than decrypting the ciphertexts; i.e., for adversarially-generated ballots (c_i, π_i), we use $(\mathsf{crs}, \tau_s, \tau_e) \xleftarrow{\$} SE_1(1^k)$ and $m_i := \mathsf{Extract}(\tau_e, \varphi(\pi_i))$ instead of $m_i := \mathsf{Dec}(sk, \varphi(c_i))$.

Lemma D.4. *If the proof system* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *satisfies simulation sound extractability, as defined in Definition A.2, then Game* G_4 *is indistinguishable from* G_3*.*

Proof. To prove this, we show that if there exists an adversary $\mathcal{A}$ that distinguishes between G_3 and G_4 with some non-negligible advantage ϵ then we can use it to construct an adversary $\mathcal{B}$ that breaks simulation-sound extractability with some related non-negligible probability ϵ'. We first observe that, in order to distinguish between the games, there must exist some index i such that $\mathsf{Extract}(\tau_e, \varphi(\pi_i)) \neq \mathsf{Dec}(sk, \varphi(c_i))$, as otherwise the two games are identical.

To start, $\mathcal{B}$ will get as input (crs, τ_e). It now computes all the other parameters completely honestly, and gives the parameters and the appropriate keys for the corrupted parties to $\mathcal{A}$; note that this means it will also know the decryption key sk. In the voting phase, if $\mathcal{A}$ outputs a query of the form (v_0, v_1), $\mathcal{B}$ will form $c \xleftarrow{\$} \mathsf{Enc}(pk, v_0)$ and query its S_2 oracle on (pk, c) to get back a proof π; it then returns (c, π) to $\mathcal{A}$ and posts it to the bulletin board. If $\mathcal{A}$ instead posts its own ballot (c, π), $\mathcal{B}$ checks that $\mathsf{Dec}(sk, c) = \mathsf{Extract}(\tau_e, \pi)$. If this check fails, then $\mathcal{B}$ outputs $((pk, c), \pi)$, and otherwise it continues the game. If it reaches the end of the voting phase without observing such a ballot, $\mathcal{B}$ aborts and outputs $\perp$.

To see that interactions with $\mathcal{B}$ are indistinguishable from those that $\mathcal{A}$ expects, we observe that if $\mathsf{Dec}(sk, c) = \mathsf{Extract}(\tau_e, \pi)$, then $\mathcal{B}$ is in fact executing the code of both games completely honestly. If $\mathsf{Dec}(sk, c) \neq \mathsf{Extract}(\tau_e, \pi)$, then $\mathcal{B}$ has output a tuple $((pk, c), \pi)$ such that $w := \mathsf{Extract}(\tau_e, \pi)$ but $w \neq \mathsf{Dec}(sk, c)$, meaning $((pk, c), w) \notin R$. In any case in which $\mathcal{A}$ could distinguish between the games, $\mathcal{B}$ therefore wins at its game, meaning that if $\mathcal{A}$ distinguishes with some non-negligible advantage ϵ, $\mathcal{B}$ will successfully output such a tuple with probability ϵ. □

Next, in Game G_5, we switch away from decrypting for the ballots produced by query as well; as the proofs here are simulated and thus we cannot extract from them, this means that, if the

adversary queried on (v_{i0}, v_{i1}) and got back a ballot (c_i, π_i), then in the share decryption oracle we use $m_i := \varphi(v_{i0})$ in place of $m_i := \mathsf{Dec}(sk, \varphi(c_i))$.

Lemma D.5. *If the threshold encryption scheme* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify})$ *is complete, then Game* G_5 *is identical to* G_4*.*

Proof. Completeness tells us that, if $c \xleftarrow{\$} \mathsf{Enc}(pk, m)$, then $\mathsf{Dec}(sk, c) = m$. On a query of the form (v_{i0}, v_{i1}), the vote v_{i0} is encrypted honestly to form c_i. By correctness then, we must have that $\mathsf{Dec}(sk, c_i) = v_{i0}$, and thus we can use $\mathsf{Dec}(sk, c_i)$ and v_{i0} interchangeably. □

Next, in Game G_6, we switch to using $\varphi(v_{i1})$ in place of $\varphi(v_{i0})$ in the share decryption oracle.

Lemma D.6. *If* $\{v_{i0}\}_i$ *and* $\{v_{i1}\}_i$ *result in the same election outcome, then Game* G_6 *is identical to* G_5*.*

Proof. If $\{v_{i0}\}_i$ and $\{v_{i1}\}_i$ result in the same outcome, then we know that, as sets, they must be equal. This means that for any $\varphi \in S_L$, there exists another $\varphi' \in S_L$ such that when we consider the effect of the permutations just on the queried votes, we get $\varphi(\{v_{i0}\}_i) = \varphi'(\{v_{i1}\}_i)$. As the permutation in our game is chosen uniformly at random from S_L, picking φ as above is equally likely as picking φ', meaning the distributions $\mathcal{D}_0 := \{\varphi \xleftarrow{\$} S_L : \varphi(\{v_{i0}\}_i)\}$ and $\mathcal{D}_1 := \{\varphi \xleftarrow{\$} S_L : \varphi(\{v_{i1}\}_i)\}$ are identical. □

Next, now that we are not using the secret key anywhere, and only the ciphertext returned to $\mathcal{A}$ reveals information about the vote, we switch in Game G_7 to encrypting the right-hand votes; i.e., using $c \xleftarrow{\$} \mathsf{Enc}(pk, v_1)$ rather than $c \xleftarrow{\$} \mathsf{Enc}(pk, v_0)$.

Lemma D.7. *If the threshold encryption scheme* $(\mathsf{EncKg}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{ShareVerify})$ *is IND-CPA secure, then Game* G_7 *is indistinguishable from* G_6*.*

Proof. To prove this, we show that if there exists an adversary $\mathcal{A}$ that distinguishes between G_6 and G_7 with some non-negligible advantage ϵ then we can use it to construct an adversary $\mathcal{B}$ that breaks IND-CPA security with advantage ϵ/q, where q is the number of queries $\mathcal{A}$ makes in the voting phase. To do this, we proceed through a series of hybrid games H_0 through H_q. In each game H_i, the first i queries are answered using $b = 0$, while the next $q - i$ are answered using $b = 1$. We therefore have that $H_q = G_6$ and $H_0 = G_7$, so to show that G_7 is indistinguishable from G_6 it suffices to show that H_{i+1} is indistinguishable from H_i for all i, $0 \leq i < q$. We then show that if there exists an adversary $\mathcal{A}'$ that distinguishes between H_i and H_{i+1} with some non-negligible advantage ϵ then we can use it to construct an adversary $\mathcal{B}'$ that breaks IND-CPA security with the same advantage.

To start, $\mathcal{B}'$ receives as input a public key pk. It then forms all the other parameters completely honestly, and gives the parameters and the appropriate keys for the corrupted parties to $\mathcal{A}'$. In the voting phase, if $\mathcal{A}'$ outputs a query of the form (v_0, v_1), for the first $i - 1$ queries $\mathcal{B}'$ will form $c \xleftarrow{\$} \mathsf{Enc}(pk, v_0)$ and $\pi \xleftarrow{\$} S_2(\mathsf{crs}, \tau_s, (pk, c))$ and return (c, π) to $\mathcal{A}'$ (and writes it to the bulletin board). For the i-th query, $\mathcal{B}'$ queries its own oracle on (v_0, v_1) to get back a ciphertext c; it then forms $\pi \xleftarrow{\$} S_2(\mathsf{crs}, \tau_s, (pk, c))$ and returns (c, π) to $\mathcal{A}'$. For the rest of the queries, $\mathcal{B}'$ forms $c \xleftarrow{\$} \mathsf{Enc}(pk, v_1)$ and $\pi \xleftarrow{\$} S_2(\mathsf{crs}, \tau_s, (pk, c))$ and returns (c, π) to $\mathcal{A}'$, and in all the other phases, $\mathcal{B}'$ behaves completely honestly. At the end of the game, if $\mathcal{A}'$ guesses H_i then $\mathcal{B}'$ guesses $b = 1$, and if $\mathcal{A}'$ guesses H_{i+1} then $\mathcal{B}'$ guesses $b = 0$.

To see that interactions with $\mathcal{B}'$ are indistinguishable from those that $\mathcal{A}'$ expects, we observe that if the oracle uses $b = 0$ then $\mathcal{B}'$ answers the first $i + 1$ queries using $b = 0$ and thus is executing the exact code of H_{i+1}, while if it uses $b = 1$ it is executing the exact code of H_i. As $\mathcal{B}'$ furthermore

behaves identically to both games the rest of the time, $\mathcal{A}'$ will have the same advantage with $\mathcal{B}'$ as it does in distinguishing games 6 and 7. As $\mathcal{B}'$ succeeds in guessing whenever $\mathcal{A}'$ does, $\mathcal{B}'$ will also succeed with non-negligible advantage ϵ and thus $\mathcal{B}$ will succeed with advantage ϵ/q. □

Now that we are using $b = 1$ (i.e., encrypting right-hand votes), we can switch everything else back. In Game G_8, we switch back to using $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ for query-generated ballots; this is identical to G_7 by the completeness of the encryption scheme, and the proof is analogous to the proof of Lemma D.5. In Game G_9, we switch back to using $m_i := \mathsf{Dec}(sk, \varphi(c_i))$ for adversarially-generated ballots; this is indistinguishable from G_8 by simulation-sound extractability, and the proof is analogous to the proof of Lemma D.4. Next, in Game G_{10}, we switch back to using honest proofs of knowledge; this is indistinguishable from G_9 by zero knowledge, and the proof is analogous to the proof of Lemma D.3. Next, in Game G_{11}, we switch back to using $\mathsf{Dec}(sk, c_i')$; this is indistinguishable from G_{10} by compact verifiability, and the proof is analogous to the proof of Lemma D.2. Finally, we switch back to using EncKg and $\mathsf{ShareDec}$ in the tallying phase; this is indistinguishable from G_{11} by share simulatability, and the proof is analogous to the proof of Lemma D.1.