A Novel Proof of Shuffle: Exponentially Secure Cut-and-Choose

Thomas Haines[1] and Johannes Müller[2][0000−0003−2134−3099]

[1] Norwegian University of Science and Technology, Norway,
firstname.lastname@ntnu.no
[2] SnT, University of Luxembourg, Luxembourg,
firstname.lastname@uni.lu

Abstract. Shuffling is one of the most important techniques for privacy-preserving protocols. Its applications are manifold, including, for example, e-voting, anonymous broadcast, or privacy-preserving machine-learning. For many applications, such as *secure* e-voting, it is crucial that the correctness of the shuffling operation be (publicly) verifiable. To this end, numerous proofs of shuffle have been proposed in the literature. Several of these proofs are actually employed in the real world.
In this work, we propose a generic compiler which can transform any "shuffle-compatible" Σ-protocol (including, among others, Σ-protocols for re-randomization, decryption, or key shifting) into a Σ-protocol for permutations of the underlying relation. The resulting proof of shuffle is black-box, easily implementable, simple to explain, and comes with an acceptable computational overhead over the state-of-the-art. Because we machine-checked our compiler in Coq, the new proof of shuffle is particularly suitable for applications that require a superior level of security assurance (e.g., high-stake elections).

1 Introduction

Proofs of shuffles are fundamental building blocks in many privacy-preserving technologies. Most prominently, they are employed in verifiable mix nets [26] that are often used for secure e-voting. Numerous proofs of shuffles have been proposed in the literature. Some of them, such as the state-of-the-art proofs of shuffles by Terelius-Wikström [42, 44] and Bayer-Groth [5], were deployed in government elections in Switzerland, Estonia, Australia, and Norway. Additionally, they have also processed millions of ballots in low-stake elections. However, these state-of-the-art proof systems have some disadvantages:

1. *Design complexity:* Implementing cryptography is notoriously difficult [20, 25, 27, 40, 41], and this is even more the case for the highly complex proofs of shuffles from [5, 14, 42, 44]. Theoretical superiority

can be useless if a protocol is not implemented correctly in practice. Indeed, the vVote project [11] in the Australian state of Victoria took this issue into account: they used a technique called random partial checking [28] since it was easier to implement even though it provides weaker security from a theoretical perspective.

2. *Cryptographic security proofs:* Due to their complexity, cryptographic security proofs tend to be error-prone. For example, the original proof of the OAEP construction [6] needed to be fixed multiple times [16, 17, 39]. On the contrary, machine-checked proofs (e.g., [1] for OAEP) in reasonable frameworks guarantee higher assurance. However, there do not exist such proofs in the literature for the state-of-the-art proofs of shuffles [5, 14], even though this would be particularly desirable due to their involved concepts.
3. *Specific design:* In order to build modular security protocols (e.g., e-voting protocols), it is advantageous if sub-protocols (e.g., mix nets) are black-box. By this, interfaces can be simplified, and sub-protocols can be replaced/updated more easily. Unfortunately, state-of-the-art proofs of shuffles [5, 14, 42, 44] have very specific protocol structures and can thus support particular data structures only.

In applications like electronic voting, mix nets are by far the most complicated pieces of cryptography implemented, and, overwhelmingly, see utterly inadequate scrutiny, even compared to the poor base level for deployed e-voting schemes, more generally. Given the long list of verifiable mix nets shown to be flawed (see, e.g., [29–32, 43]), a trusted methodology for checking the security of verifiable mix nets—both in the design and implementation—is of paramount importance. Indeed, the SwissPost e-voting system for national elections in Switzerland was withdrawn from use in 2019 in part due to an insecure mix net implementation [25]. Unlike in regularly used security protocols (e.g., key exchange), efficiency is of less concern in high-stake e-voting. Instead, for such elections, it is far more important that the e-voting system does in fact provide the security properties it is supposed to achieve; in short: "security assurance $\gg$ top-notch performance".

In this paper, we follow a novel and radically different approach in order to address the aforementioned requirements. We propose a conceptually simple and widely applicable proof of shuffle that we machine-checked in Coq to demonstrate its superior level of security assurance. More precisely, we provide the following contributions.

Contributions.

1. We introduce and formalize the notion of *shuffle-compatible* Σ-protocols (SCSP). We show that several commonly used Σ-protocols are shuffle-compatible. This includes, among others, Σ-protocols for re-encryption of arbitrary homomorphic ciphertexts, for re-randomisation of arbitrary homomorphic commitments (including lattice-based ones), for key shifting in ElGamal PKE, and for decryption in ElGamal PKE.
2. We propose a generic and conceptually simple compiler which can transform any SCSP into a Σ-protocol for permutations of the underlying relation. The resulting proofs of shuffle have particularly interesting properties in the interactive setting.
3. We provide machine-checked (and cryptographic) security proofs for the generic compiler and the SCSPs mentioned before. To this end, we used the interactive theorem prover Coq [7].[3]

Structure of the paper. We start with related work in Sec. 2. After that, in Sec. 3, we describe the main technical idea of our generic compiler as well as a number of interesting Σ-protocols to which it can be applied. In Sec. 4, we formalize the notion of SCSP and show that the interesting Σ-protocols from Sec. 4 provide this property. We describe our generic compiler in Sec. 5 and analyze its complexity in Sec. 6. We conclude in Sec. 7.

2 Related Work

2.1 Proofs of shuffle

Proofs of shuffles are often used to make mix nets verifiable. Beyond proofs of shuffles, several different techniques have been proposed for this purpose (see [26] for a recent systemization-of-knowledge). Since proofs of shuffles are the only known technique to provide an ideal verifiability level for mix nets (i.e., where manipulating at least one message is detected with overwhelming probability), we will restrict our attention to proofs of shuffles in what follows.

In practice, the most common proofs of correct shuffle are [42, 44] (which are foundation of the prominent Verificatum mix net [45]) and [5]. There are also more efficient proofs of correct shuffle which have since emerged [13–15]. These new proofs are roughly three times faster than [5, 42, 44] but require trust assumptions which are typically undesirable in practice (see [26]).

[3] The Coq code can be found at `https://www.dropbox.com/s/7q9k504hie144x7`. We will later make it available at a public git repository.

Historically, the first technique for verifiable mix nets was proposed in [35]. Their technique was a straightforward cut-and-choose zero-knowledge proof. The proof is fairly effective but was considered to be computational impractical. We show the cut-and-choose based approach introduced in this paper is not only more generic and machine-checked but it also has acceptable performance overhead compared with state-of-the-art protocols [5, 42, 44] (see Sec. 6). More specifically, in the most common case (mixing ElGamal ciphertexts) our proof of shuffle either has, depending on the batch techniques we apply, computational cost or size within a small factor of the state-of-the-art, but not both at the same time. We leave as future work the investigation of batch techniques which would reduce both size and computational cost. Nevertheless, we believe that, for applications like high-stake e-voting, the simplicity, generality and machine-checked security of our approach more than justifies the performance trade-off.

To test the practicality, we simulated a test election using commodity hardware with 1,000,000 voters. We employed the proof of shuffle produced by our transform on the ElGamal re-encryption SCSP implemented over the prime-order Ristretto subgroup of Curve25519 using `curve25519-dalek` [34] and the optimisations detailed in Sec. 6.3. The proof generation and verification time was 40 minutes. The proof size is 128 megabytes (which is denominated by sending the encrypted votes). Larger elections might require either additional cores or checking the proof overnight.

2.2 Machine-checked proofs

Interactive theorem provers are tools to encode mathematically rigorous definitions and algorithms. Desired properties can be encoded as theorems which are interactively proved (machine-checked).

The machine-checked proofs of our compiler and the SCSPs are in the interactive theorem prover Coq [8]. Coq is based upon Coquand's Calculus of Constructions and has been developed for decades. A significant body of work has already been completed on verifying cryptography in Coq, most notably, the CertiCrypt project [2]. Because the proofs we give are straight reductions without utilizing game hopping, we do not use CertiCrypt. Moreover, CertiCrypt appears to have been abandoned in favor of EasyCrypt.[4] EasyCrypt is a separate interactive proof system which is designed specifically for verifying cryptographic proofs. Early versions

[4] See `http://certicrypt.gforge.inria.fr/\#related`

of EasyCrypt were compatible with CertiCrypt but this has since been discontinued. EasyCrypt is seeing exciting developments but at present is far less mature than Coq.

Interactive theorem provers, particularly mature ones, give higher confidence in the security of the proofs. However, do they not (necessarily) increase confidence in the definitions. For this reason, we prove our transform under established definitions. We use the definition of a Σ-protocol from [24] which was subsequently refined in [23].

One advantage of using Coq is that we able to take advantage of it's well-established code extraction facility to produce practical implementations of the verified specifications. This has been done before by Haines et al. [23, 24] who proved the security of the underlying sigma protocol in the Terelius-Wikström [42, 44] proof of shuffle and used the extraction facility to produce a verifier to check real elections. Compared to their work ours is more general in that Terelius-Wikström was only proved for re-encryption and re-randomisation whereas we cover a much wider class of underlying relations. Moreover, they only checked the completeness and zero-knowledge of the underlying sigma protocol but not that this suffices for the completeness and zero-knowledge of the Terelius-Wikström mix net. In contrast we machine check the entire proof of shuffle.

3 Technical Overview

We first describe the main idea of our generic proof of shuffle. We then elaborate on a number of concrete interesting applications.

3.1 Main idea

We now explain the main idea of the generic compiler which takes as input an arbitrary SCSP for relation $\mathcal{R}$ and outputs a (standard) Σ-protocol for relation

$$\mathcal{R}_{\mathsf{Shuffle}} = \{((x_j), \pi), (y_j, y'_j)_{j\in[\tau]}) \colon \forall j \in [\tau] \colon ((x_j), (y_j, y_{\pi(j)})) \in \mathcal{R}\},$$

where τ denotes the size of the shuffled vector.[5] To illustrate our approach, we first describe the notion of SCSPs, i.e., how they differ from general Σ-protocols. After that, we explain the main technique of our compiler.

Shuffle-compatible Σ protocols (SCSP). In general, a Σ-protocol for relation $\mathcal{R}$ is a particular form of interactive zero-knowledge proof between

[5] We will refine $\mathcal{R}_{\mathsf{Shuffle}}$ further below.

a prover P and verifier V. The joint input for P and V is a *statement* y, and the secret input to P is a *witness* x for y, i.e., $(x, y) \in \mathcal{R}$. In the first step, the prover creates a so-called *commitment* a, sends it to the verifier, who then replies with a random *challenge* e in the second step. In the third step, the prover computes a *response* z which it sends to the verifier. Eventually, the verifier on input statement y and *transcript* (a, e, z) either outputs "accept" or "reject". Each Σ-protocol guarantees *completeness* (if the prover and the verifier run their specified programs and $(x, y) \in \mathcal{R}$, then the verifier outputs "accept"), *special soundness* (if the prover is able to output valid responses z and z' for two different challenges e and e' but for the same commitment a, then the prover knows a witness x for statement y), and *special honest verifier zero-knowledge* (the interaction between the prover and the verifier can be simulated when the challenge e is given). The formal definition of a Σ-protocol is given in Sec. 4.1.

Now, a SCSP is essentially a (standard) cut-and-choose Σ-protocol, where the statement y is of the form (prm, c_0, c_1). That is, the statement consists of some parameters prm and two elements c_0 and c_1; for example, a public key and two ciphertexts. In the commitment phase, the prover constructs an intermediary value a between c_0 and c_1. After it receives a challenge bit e, the prover responds with a message z that reveals the relationship between a and c_e. In particular, the verifier can check the correctness of the transcript (a, e, z) using only the *partial* statement (prm, c_e). We will formally define the notion of SCSPs in Sec. 4.2. As we shall see below, many interesting Σ-protocols are actually shuffle-compatible.

Compiler. Let us now explain the main concept of our generic compiler. The compiler takes as input a SCSP for relation $\mathcal{R}$, and outputs a (standard) Σ-protocol $\Sigma_{\mathsf{Shuffle}}$ for relation

$$\mathcal{R}_{\mathsf{Shuffle}} = \{(((x^j)_{j\in[\tau]}, \pi), (\mathrm{prm}, (c_0^j)_{j\in[\tau]}, (c_1^j)_{j\in[\tau]})) : \\ \forall j \in [\tau] : (x^j, (\mathrm{prm}, c_0^{\pi(j)}, c_1^j)) \in \mathcal{R}\}.$$

To this end, the compiler constructs $\Sigma_{\mathsf{Shuffle}}$ (Fig. 1) as follows:

Commit phase: The prover P first runs the commit algorithm of the underlying SCSP for each entry j to obtain a commitment a^j (plus some internal state α^j). Then, the prover shuffles $(a^j)_{j\in[\tau]}$ according to a uniformly random permutation π_a, and returns commitment $(a^{\pi_a(j)})_{j\in[\tau]}$.

Response phase: The prover first runs the response algorithm of the underlying SCSP for each entry j to obtain a response z^j. If the chal-

lenge bit e was 0, the prover responds with $(\pi \circ \pi_a, (z^{\pi_a(j)})_{j\in[\tau]})$, and otherwise with $(\pi_a, (z^{\pi_a(j)})_{j\in[\tau]})$. That is, the prover either "opens" the right links while the left ones remain hidden by π_a, or the prover "opens" the left links while the right ones remain hidden by π.

Verification phase: The verifier checks whether the opened links are correct, i.e., for each j, the verifier runs the check of the underlying SCSP for partial statement $(\mathrm{prm}, c_e^{\pi_z(j)})$ and transcript (a^j, e, z^j).

We will define $\Sigma_{\mathsf{Shuffle}}$ formally in Sec. 5. We provide both a machine-checked proof of our compiler in Coq [7] (module `ProofOfShuffle`) as well as a cryptographic one (App. D).

3.2 Applications

We list a number of concrete SCSPs that can be transformed by our generic compiler into Σ-protocols of shuffle with interesting applications. We formally define these examples in App. A. The following list is by no means exhaustive; there likely exists many further potentially interesting SCSPs.

Re-encryption of ciphertexts. In every homomorphic PKE scheme, it is possible to *re-encrypt* a given ciphertext $c = \mathsf{Enc}(\mathsf{pk}, m; r)$ into a different ciphertext $c' = \mathsf{Enc}(\mathsf{pk}, m; r')$ (which still encrypts the same message m under the same public key pk but with different randomness r') without knowledge of the secret key sk, message m, or randomness r. Prominent examples of such schemes include ElGamal PKE [19] and Paillier PKE [36]. We will show in App. A that a (common) generic Σ-protocol for proving that c' is a re-encryption of c is actually SCSP. Now, by transforming this SCSP with our generic compiler from Sec. 5, we can use the resulting Σ-protocol for making any *re-encryption mix net* verifiable [26]. These protocols are often used in secure e-voting to guarantee vote privacy (e.g., in Civitas [10]).

Re-randomization of commitments. In every homomorphic commitment scheme, it is possible to *re-randomize* a commitment $c = \mathsf{Com}(\mathsf{pk}, m; r)$ into a different commitment $c' = \mathsf{Com}(\mathsf{pk}, m; r')$ without knowledge of the opening (m, r). A prominent example of such schemes is Pedersen's one [37]. Similarly to the previous application, a (common) generic Σ-protocol for proving that c' is a re-randomized commitment of c is shuffle-compatible as well.

In the case of homomorphic commitments (unlike encryption), it is particularly interesting that our novel compiler can, in principle, not only be used to construct an efficient proof of shuffle for "traditional" commitment schemes but also for *lattice-based* ones. The reason is that a number of zero-knowledge proofs for the required underlying relations have been proposed (e.g., [4, 12]) that can be employed very efficiently in our scenario because they can be amortized by performing many of them in parallel.

Key shifting in ElGamal PKE. In several applications of the ElGamal PKE, the key is distributed across multiple authorities. For this reason, we need to do a key shifting (sometimes called partial decryption) mix [18] rather than a decryption mix. In addition, to prevent senders tracking their own ciphertext through the mix net, we want to simultaneously re-randomise and do a key shift at each point.

Decryption of ElGamal ciphertexts. A commonly used Σ-protocol for proving that an ElGamal ciphertext was decrypted correctly is shuffle-compatible as well.

4 Sigma Protocols

In this section, we first recall the general concept of Σ-protocols. After that, we describe a class of Σ-protocols which are compatible with the novel proof of shuffle that we will introduce in Sec. 5. This class includes numerous Σ-protocols for commonly used relations, such as re-randomisation of commitments or ciphertexts, decryption, and key shifting (see App. A).

4.1 General Sigma Protocols

We start with recalling the general definition of Σ-protocols.

Definition 1 (Sigma protocol). *Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be an NP relation. A* Σ-protocol *for $\mathcal{R}$ with challenge length $t \geq 1$ is a pair of probabilistic polynomial-time (ppt) interactive Turing machines (P, V), where*

- *the* prover *P takes as input a witness-statement pair $(x, y) \in \mathcal{R}$,*
- *the* verifier *V takes as input a statement $y \in \mathcal{Y}$, and returns 0 or 1,*
- *the structure of the interaction between P and V is as follows:*
  1. *P: compute* commitment *a and send a to V*

2. *V: compute* challenge $e \xleftarrow{r} \{0,1\}^t$ *and send* e *to* P
3. *P: compute* response z *and send* z *to* V
4. *V: output either 0 or 1 (as a function of* y *and* (a, e, z)*).*

– completeness *(Def. 2),* special soundness *(Def. 3), and* special honest verifier zero-knowledge *(Def. 4) are guaranteed.*

We say that $\mathsf{trans}(\langle P(x,y), V(y)\rangle) := (a, e, z)$*, where* a, e, z *are as above, is a* transcript *of the conversation between* P *and* V*. We say that* (a, e, z) *is an* accepting transcript (for y) *if and only if* V *returns 1 in this conversation.*

Definition 2 (Completeness). *Let* (P, V) *be as in Def. 1. We say that* (P, V) *achieves* completeness *if and only if for all* $(x, y) \in \mathcal{R}$*:*

$$\Pr\left(\langle P(x,y), V(y)\rangle = 1\right) = 1.$$

Definition 3 (Special soundness). *Let* (P, V) *be as in Def. 1. We say that* (P, V) *achieves* special soundness *if and only if there exists a polynomial-time* extractor *algorithm* Ext*, where*

– Ext *takes as input statement* $y \in \mathcal{Y}$*, and two accepting transcripts* (a, e, z)*,* (a, e', z') *where* $e \neq e'$*,*
– Ext *outputs witness* x *such that* $(x, y) \in \mathcal{R}$*.*

Definition 4 (Special honest verifier zero-knowledge). *Let* (P, V) *be as in Def. 1. We say that* (P, V) *achieves* special honest verifier zero-knowledge *if and only if there exists a ppt* simulator *algorithm* Sim*, where*

– Sim *takes as input statement* $y \in \mathcal{Y}$ *and challenge* e*,*
– Sim *outputs an accepting transcript* (a, e, z) *such that*

$$\mathsf{Sim}(y, e) = \mathsf{trans}(\langle P(x,y), V^e(y)\rangle)$$

holds true (i.e., the simulator's output $\mathsf{Sim}(y, e)$ *and the transcript between* $P(x, y)$ *and* $V(y)$ *who chooses challenge* e *have same distributions).*

4.2 Shuffle-Compatible Sigma Protocols (SCSP)

We now characterize those Σ-protocols which are compatible with the proof of shuffle introduced in Sec. 5.

The main characteristic of these SCSP is that the public statement y can be expressed as $y = (\mathrm{prm}, c_0, c_1)$, where c_0 will be the "input"

element to the shuffle and c_1 will be the "output" element. Now, for a given challenge $e \in \{0,1\}$,[6] the input to the verifier's final check can be restricted to the *partial* public statement (prm, c_e) (and the transcript (a, e, z) as before). Furthermore, SCSPs require the following stronger variant of special honest verifier zero-knowledge: the simulator Sim takes as input the challenge $e \in \{0,1\}$ (as before) but only the partial public statement (prm, c_e).

Definition 5 (Shuffle-compatible sigma protocol). *Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be an NP relation where elements from $\mathcal{Y}$ are of the form (prm, c_0, c_1). A* shuffle-compatible Σ-protocol (SCSP) *(with challenge length 1) is a pair of ppt interactive Turing machines (P, V), where*

- *the prover P takes as input a witness-statement pair $(x, (prm, c_0, c_1)) \in \mathcal{R}$,*
- *the verifier V takes as input a statement $y = (prm, c_0, c_1)$, and returns 0 or 1,*
- *the structure of the interaction between P and V is as follows:*
  1. *P: compute $(a, \alpha) \leftarrow \mathsf{Com}(x, y)$ and send commitment a to V*
  2. *V: compute challenge $e \xleftarrow{r} \{0,1\}$ and send e to P*
  3. *P: compute response $z \leftarrow \mathsf{Resp}(x, y, (a, \alpha), e)$ and send z to V*
  4. *V: output $0/1 \leftarrow \mathsf{Check}((prm, c_e), (a, e, z))$*
- *completeness (Def. 2), special soundness (Def. 3), and a variant of special honest verifier zero-knowledge (Def. 6) are guaranteed.*

We say that (a, e, z) is an accepting transcript *if and only if V outputs 1, i.e., $\mathsf{Check}((prm, c_e), (a, e, z)) = 1$, in this conversation.*

Definition 6 (Shuffle-compatible special honest verifier ZK). *Let (P, V) be as in Def. 5. We say that (P, V) achieves* shuffle-compatible special honest verifier zero-knowledge *if and only if there exists a ppt* simulator *algorithm Sim, where*

- Sim *takes as input* partial *statement (prm, c_e) and challenge e,*
- Sim *outputs an accepting transcript (a, e, z) such that*

$$\mathsf{Sim}(prm, c_e, e) = \mathsf{trans}(\langle P(x, y), V^e(y) \rangle)$$

holds true.

[6] Without loss of generality, we restrict our attention to Σ-protocols with challenge length $t = 1$.

The Coq definition of a shuffle-compatible Σ-protocol is the module type `SigmaOfFunction`. This module defines a shuffle-compatible Σ-protocol as a Σ-protocol with the restrictions on the verifier and simulator.

In App. A, we included SCSPs (and accompanying proofs) for many common relationships including re-encryption of ciphertexts for arbitrary homomorphic public key encryption schemes, re-randomisation of commitments for homomorphic commitment schemes, re-randomisation of lattice-based commitments.

5 Transform

In this section, we present our main contribution: a generic Σ-protocol of correct shuffle $\Sigma_{\mathsf{Shuffle}}$ which can invoke any SCSP, as specified in Def. 5. The protocol $\Sigma_{\mathsf{Shuffle}}$ is described in Fig. 1. The formal relation to be proven is

$$\mathcal{R}_{\mathsf{Shuffle}} = \{(((x^j)_{j\in[\tau]}, \pi), (\mathrm{prm}, (c_0^j)_{j\in[\tau]}, (c_1^j)_{j\in[\tau]})) : \\ \forall j \in [\tau] : (x^j, (\mathrm{prm}, c_0^{\pi(j)}, c_1^j)) \in \mathcal{R}\},$$

where $\mathcal{R}$ is the relation of the underlying shuffle-compatible Σ-protocol. We refer to Sec. 3 for the main idea of $\Sigma_{\mathsf{Shuffle}}$.

We provide two proofs that $\Sigma_{\mathsf{Shuffle}}$ is a Σ-protocol for relation $\mathcal{R}_{\mathsf{Shuffle}}$. The first proof is a machine-checked one using Coq. The Coq encoding of the transform is given by the module `ProofOfShuffle`. The module defines the proof of shuffle as defined in Fig. 1 and then proves that it is a Σ-protocol for the relation $\Sigma_{\mathsf{Shuffle}}$. The second proof is a cryptographic proof which is provided in App. D.

6 Complexity

The basic complexity (computational performance and proof size) of the new proof of shuffle is straightforward. At the same time, the new proof of shuffle comes with some interesting properties that make its performance more multifaceted. To illustrate this, in what follows, we first elaborate on the proof's basic complexity and then we describe useful trade-offs in the interactive setting as well as some optimisations. After that, we compare the complexity of the new proof of shuffle with state-of-the-art protocols. Eventually, we provide concrete values for performance and proof size for large-scale elections.

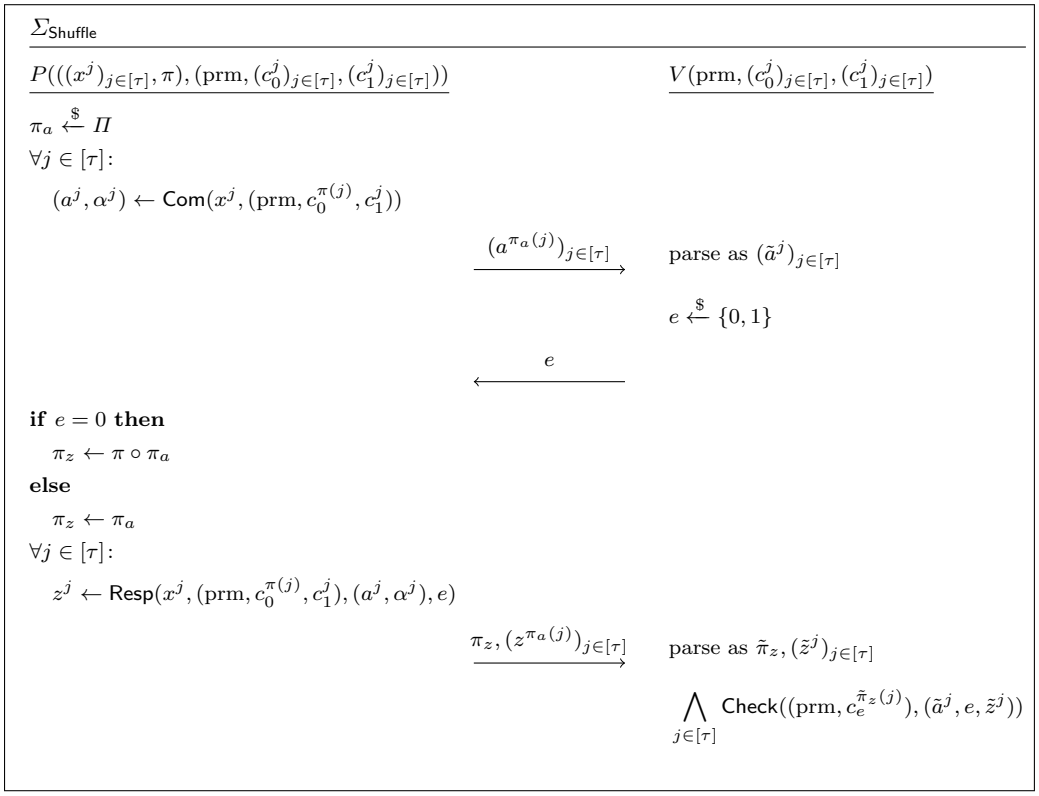

Fig. 1: Σ-protocol of correct shuffle $\Sigma_{\mathsf{Shuffle}}$. The protocol $\Sigma_{\mathsf{Shuffle}}$ invokes the abstract SCSP Σ specified in Def. 5. In particular, the algorithms Com, Resp, and Check employed in $\Sigma_{\mathsf{Shuffle}}$ are the ones of the underlying shuffle-compatible Σ-protocol.

6.1 Basic complexity

Let s be the proof size of the underlying Σ-protocol and let c_P and c_V be the prover and verifier complexity in Σ, respectively. Then, the proof size of $\Sigma_{\mathsf{Shuffle}}$ is $\tau \cdot s$ group and field elements and the permutation. The prover's complexity is $\tau \cdot c_P$, and the verifier's complexity is $\tau \cdot c_V$, where τ is the number of items being shuffled. Consequently, if λ is the number of times we repeat $\Sigma_{\mathsf{Shuffle}}$ to improve the soundness level down to $2^{-\lambda}$, then the proof size is $\tau \cdot s \cdot \lambda$, the prover's complexity is $\tau \cdot c_p \cdot \lambda$, and the verifier's complexity is $\tau \cdot c_V \cdot \lambda$.[7]

[7] We have assumed that both Com and Resp contain at least one expensive operation (such as exponentiation) which will dwarf the cost of handling the permutations.

6.2 Interactive vs non-interactive

Existing protocols in the literature do not gain much performance advantage by reducing the security parameter λ. On the contrary, the new proof of shuffle has linear complexity for both the prover and verifier in λ. This means that the interactive variant provides the following interesting trade-off. For example, if we use our protocol for secure e-voting, then we can first run an interactive variant with low security parameter, say $\lambda = 10$, which will allow the generation of a proof hundreds of times faster (in the online phase) than any other proof of shuffle. This will give 99.9% immediate confidence in the integrity of the election outcome while a higher confidence can be produced in the coming hours.

6.3 Optimisations

We describe two options to optimize the new proof of shuffle: amortising and pre-computation.

Amortising. In all examples considered in App. A, it is possible to significantly reduce the computational cost by amortising the underlying operations. In the case of the lattice example, this is explicit. The ElGamal example presents two possible ways to amortise:

Speed Since all exponentations are to a fixed base, we can use Gordon's [21] Radix-R method. More precisely, if we are handling x ciphertexts in a group order y, then we set R as 2^d where d is chosen to minimise $2^d \log_{2^d} 2^y + x \log 2^d 2^y$. Consequently, in practice, when handling millions of ciphertexts on a group of order roughly 2^{256}, this reduces the cost of exponentiation to less than $\frac{1}{10}$ of the cost of a normal exponentiation.

Space Batch techniques could be applied to reduce the size of the proof, for instance [38]. It is straightforward to apply such techniques (in the ElGamal examples) to get a proof with a number of group and field elements independent of the number of messages. However, the suggested computational optimisations would no longer be effective.

Pre-Computation. In prominent use cases (like electronic voting), there is a significant period of time (vote casting period), followed by a shorter period in which the proof must be computed (tally phase). It is therefore advantageous when proof of shuffles have the option to do significant amounts of their computation before knowing the exact ciphertexts to be

mixed. This is similar to the application scenario of modern secure multi-party computation (MPC) protocols which split their computational cost into a rather slow offline phase (which can be computed ahead of time without knowing the inputs) and a very fast online phase (which depends on the specific inputs). Similarly, as explained next, our new proof of shuffle offers a particularly useful offline/online split since the offline computation does not depend on the exact number of items to be shuffled (in contrast to all other state-of-the-art proofs).[8]

Observe that in the re-randomisation examples considered in App. A, the prover's computationally most expensive parts are independent of the statement to proven. More precisely, in the ElGamal-based examples, only the commitment algorithm Com contains exponentations and these can be executed without knowledge of the statement; all other calculations by the prover are computationally minor. Similarly, in the lattice-based examples, computationally expensive Gaussian sampling appears only in the commitment algorithm Com and can be executed independently of the statement. Therefore, all expensive steps in these examples can be pre-computed in an offline phase so that the online phase is blazingly fast.

6.4 Comparison

	Size	Offline c_P	Online c_P	Complexity c_V
Our protocol (speed)	$\tau(4G + 128(2G + F + \log(\tau)))$	24τ	0	24τ
Our protocol (space)	$\tau 4G + 128(2G + F + \tau\log(\tau))$	0	256τ	256τ
[5]	$\tau 4G + 11mG + 5nF$	0	$2\log(m)\tau$	4τ
[42, 44]	$\tau(11G + 2F)$	0	5τ	8τ

Table 1: Comparison of $\Sigma_{\mathsf{Shuffle}}$ applied to re-encryption of ElGamal ciphertexts with state-of-the-art protocols [5] and [42, 44]. The security parameter is $\lambda = 128$. We denote the prover's computational complexity by c_P and the verifier's one by c_V. We denote by F the size of a field element and by G the size of a group element.

In practice, proofs of shuffles are most commonly used for re-encryption of ElGamal. Therefore, we compare the efficiency of our compiler (using the computational optimisations) in this case with the state-of-the-art protocols [5, 42, 44] for the concrete choice of security parameter $\lambda = 128$.

[8] There are approaches to overcome this for the state-of-the-art proofs (e.g. appending dummy ciphertexts to reach some limit) but they complicate the protocols.

We measure size as the combination of the proof size and the statement since both must be sent in practice. We denote by F the size of a field element and by G the size of a group element. We measure computational complexity in the number of exponentiations.[9] We have chosen to present the other proofs in their most efficent variants which do not seperate the online and offline phases. While, it is possible to do some pre-computation in both [5] and [42, 44] there is still $O(\tau)$ exponentiations in the online phase.

As can seen from Table 1, our protocol is either:

- much faster in the online phase and only a small factor different overall with a proof size roughly 40 times larger than the state-of-the-art,
- or of comparable size with with a proof roughly 40 times larger.

6.5 Concrete efficiency

Note that the motivation of using our work with ElGamal is not higher performance than state-of-the-art protocols but simpler and thus less error-prone implementation as well as superior security assurance due to machine-checked proofs. We have seen that this unique property comes at a cost; the performance of our new proof of shuffle is significantly worse than the state-of-the-art. Nevertheless, if we enter concrete values, we shall see that the space optimised variant still offers acceptable performance even for large scale elections.

Consider an election with 1,000,000 voters using the proof of shuffle produced by our transform on the ElGamal re-encryption SCSP implemented over the prime-order Ristretto subgroup of Curve25519 using `curve25519-dalek` [34]. Depending on the optimisations used, we either have a proof generation and verification time of 4 minutes and a proof size of 8.3 gigabytes, or a proof generation and verification time of 40 minutes and a proof size of 128 megabytes. In practice, we expect the latter would be preferred. Larger elections might require either additional cores or checking the proof overnight.

7 Conclusion

We have presented a novel proof of shuffle with black box applicability, simple design and machine-checked proofs. Our technique converts any

[9] Protocol [5] has an additional parameters m and n such that $\tau = m \cdot n$ where m is often set to 8 in practice. We have drawn on [22] for the analysis of [42].

shuffle-compatible Σ-protocol into a proof of shuffle for the underlying relation. We have shown that the computational cost of our technique (when used on a re-encryption ElGamal shuffle) is within a small factor of the state-of-the-art. Interactive versions of the techniques can provide highly efficient and small proofs with a small error term. We have, also, shown that our technique is applicable to verifiably shuffling lattice-based commitments.

Future work

Mixing post-quantum encryption schemes. It is relatively straightforward to construct a shuffle-compatible Σ-protocol for lattice-based encryption schemes (e.g., [9]) using zero-knowledge proofs of linear relations (e.g., [3]). However, to do so, would be computationally expensive and result in large proofs. An interesting area of future work is to develop amortised proofs of linear relations to allow efficient shuffling of these encryption schemes following the idea of our novel proof of shuffle.

Batch techniques. We have mentioned techniques which give either significant speed ups or reduced size. However, the techniques appear to be mutually exclusive. We leave as future work the investigation of combing the techniques to gain both increased computational efficiency and reduced size.

Acknowledgements

Thomas Haines was supported by Research Council of Norway and the Luxembourg National Research Fund (FNR), under the joint INTER project SURCVS (INTER/RCN/17/11747298/SURCVS/Ryan). Johannes Mueller was supported by the Luxembourg National Research Fund (FNR), under the CORE Junior project FP2 (C20/IS/14698166/FP2/Mueller).

Bibliography

[1] G. Barthe, B. Grégoire, Y. Lakhnech, and S. Z. Béguelin. Beyond Provable Security Verifiable IND-CCA Security of OAEP. In *Topics in Cryptology - CT-RSA 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196. Springer, 2011.

[2] G. Barthe, B. Grégoire, and S. Zanella Béguelin. Formal Certification of Code-Based Cryptographic Proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2009)*, pages 90–101. ACM, 2009.

[3] C. Baum, I. Damgård, V. Lyubashevsky, S. Oechsner, and C. Peikert. More Efficient Commitments from Structured Lattice Assumptions. In *SCN*, volume 11035 of *Lecture Notes in Computer Science*, pages 368–385. Springer, 2018.

[4] C. Baum and V. Lyubashevsky. Simple Amortized Proofs of Shortness for Linear Relations over Polynomial Rings. *IACR Cryptol. ePrint Arch.*, 2017:759, 2017.

[5] S. Bayer and J. Groth. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *Advances in Cryptology - EUROCRYPT 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.

[6] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption. In *Advances in Cryptology - EUROCRYPT '94, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.

[7] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[8] Y. Bertot, P. Castéran, G. Huet, and C. Paulin-Mohring. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Texts in theoretical computer science. Springer, 2004.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, 2014.

[10] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (S&P 2008)*, pages 354–368. IEEE Computer Society, 2008.

[11] C. Culnane, P. Y. A. Ryan, S. A. Schneider, and V. Teague. vVote: A Verifiable Voting System. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30, 2015.

[12] R. del Pino, V. Lyubashevsky, G. Neven, and G. Seiler. Practical Quantum-Safe Voting from Lattices. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1565–1581. ACM, 2017.

[13] P. Fauzi and H. Lipmaa. Efficient Culpably Sound NIZK Shuffle Argument Without Random Oracles. In *Topics in Cryptology - CT-RSA 2016, Proceedings*, pages 200–216, 2016.

[14] P. Fauzi, H. Lipmaa, J. Siim, and M. Zajac. An Efficient Pairing-Based Shuffle Argument. In *Advances in Cryptology - ASIACRYPT 2017, Proceedings, Part II*, pages 97–127, 2017.

[15] P. Fauzi, H. Lipmaa, and M. Zajac. A Shuffle Argument Secure in the Generic Model. In *Advances in Cryptology - ASIACRYPT 2016, Proceedings, Part II*, pages 841–872, 2016.

[16] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP Is Secure under the RSA Assumption. In *Advances in Cryptology - CRYPTO 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2001.

[17] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP Is Secure under the RSA Assumption. *J. Cryptol.*, 17(2):81–104, 2004.

[18] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An Implementation of a Universally Verifiable Electronic Voting Scheme based on Shuffling. In *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2002.

[19] T. E. Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[20] P. Gaudry and A. Golovnev. Breaking the Encryption Scheme of the Moscow Internet Voting System. In *Financial Cryptography and Data Security - 24th International Conference, FC 2020*, volume 12059 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2020.

[21] D. M. Gordon. A Survey of Fast Exponentiation Methods. *J. Algorithms*, 27(1):129–146, 1998.

[22] R. Haenni and P. Locher. Performance of Shuffling: Taking It to the Limits. In *Financial Cryptography Workshops*, volume 12063 of *Lecture Notes in Computer Science*, pages 369–385. Springer, 2020.

[23] T. Haines, R. Goré, and B. Sharma. Did you mix me? Formally verifying verifiable mix nets in voting. In *2021 IEEE Symposium on Security and Privacy, SP 2021*. IEEE, 2021.

[24] T. Haines, R. Goré, and M. Tiwari. Verified Verifiers for Verifying Elections. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*, pages 685–702. ACM, 2019.

[25] T. Haines, S. J. Lewis, O. Pereira, and V. Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, pages 644–660. IEEE, 2020.

[26] T. Haines and J. Müller. SoK: Techniques for Verifiable Mix Nets. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020*, pages 49–64. IEEE, 2020.

[27] J. A. Halderman and V. Teague. The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election. In *VoteID 2015, Proceedings*, volume 9269 of *Lecture Notes in Computer Science*, pages 35–53. Springer, 2015.

[28] M. Jakobsson, A. Juels, and R. L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *Proceedings of the 11th USENIX Security Symposium, 2002*, pages 339–353. USENIX, 2002.

[29] S. Khazaei, B. Terelius, and D. Wikström. Cryptanalysis of a universally verifiable efficient re-encryption mixnet. In *EVT/WOTE*. USENIX Association, 2012.

[30] S. Khazaei and D. Wikström. Randomized partial checking revisited. In *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2013.

[31] R. Küsters and T. Truderung. Security Analysis of Re-Encryption RPC Mix Nets. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 227–242, 2016.

[32] R. Küsters, T. Truderung, and A. Vogt. Formal Analysis of Chaumian Mix Nets with Randomized Partial Checking. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 343–358, 2014.

[33] A. Langlois and D. Stehlé. Worst-Case to Average-Case Reductions for Module Lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.

[34] I. A. Lovecruft and H. De Valence. curve25519_dalek. `https://doc.dalek.rs/curve25519_dalek/`.

[35] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *ICICS*, volume 1334 of *Lecture Notes in Computer Science*, pages 440–444. Springer, 1997.

[36] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - EUROCRYPT '99, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[37] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO '91, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[38] K. Peng, C. Boyd, and E. Dawson. Batch zero-knowledge proof and verification and its applications. *ACM Trans. Inf. Syst. Secur.*, 10(2):6, 2007.
[39] V. Shoup. OAEP Reconsidered. *J. Cryptol.*, 15(4):223–249, 2002.
[40] M. A. Specter, J. Koppel, and D. J. Weitzner. The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in U.S. Federal Elections. In *29th USENIX Security Symposium, USENIX Security 2020*, pages 1535–1553. USENIX Association, 2020.
[41] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman. Security Analysis of the Estonian Internet Voting System. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 703–715. ACM, 2014.
[42] B. Terelius and D. Wikström. Proofs of Restricted Shuffles. In *Progress in Cryptology - AFRICACRYPT 2010. Proceedings*, volume 6055 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2010.
[43] D. Wikström. Five Practical Attacks for "Optimistic Mixing for Exit-Polls". In *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2003.
[44] D. Wikström. A Commitment-Consistent Proof of a Shuffle. In *Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Proceedings*, volume 5594 of *Lecture Notes in Computer Science*, pages 407–421. Springer, 2009.
[45] D. Wikström. Verificatum, 2018.

A Examples of Shuffle-Compatible Sigma Protocols

We provide several examples of shuffle-compatible Σ-protocols: re-encryption of arbitrary homomorphic ciphertexts, re-randomisation of arbitrary homomorphic commitments, key shifting in ElGamal PKE, and decryption in ElGamal PKE. They can all be transformed into proofs of shuffles of the respective underlying relation using the generic compiler introduced in Sec. 5.

A.1 Abstract Re-Encryption

We define a shuffle-compatible Σ-protocol for re-encryption of ciphertexts in arbitrary homomorphic PKE schemes. In particular, this Σ-protocol

can be applied in popular PKE schemes such as the ones by ElGamal [19] and Paillier [36]. We have defined and analyzed an instance of this Σ-protocol in Coq (see module `AbstractReEncryptionSigma`).

Before we describe the shuffle-compatible Σ-protocol for re-encryption, we first recall the notion of homomorphic PKE schemes.

Definition 7 (Homomorphic PKE). *Let* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE. We say that* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* homomorphic *if and only if for all* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$*, the following properties are guaranteed:*

1. *The message space* $(\mathcal{M}_{\mathsf{pk}}, \circ)$ *and the ciphertext space* $(\mathcal{C}_{\mathsf{pk}}, \cdot)$ *are monoids.*
2. *The randomness space* $(\mathcal{F}_{\mathsf{pk}}, \odot)$ *is a group.*
3. *For all messages* $m, m' \in \mathcal{M}_{\mathsf{pk}}$ *and all randomness elements* $r, r' \in \mathcal{F}_{\mathsf{pk}}$*, we have that*

$$\mathsf{Enc}(\mathsf{pk}, m; r) \cdot \mathsf{Enc}(\mathsf{pk}, m'; r') = \mathsf{Enc}(\mathsf{pk}, m \circ m', r \odot r')$$

holds true.

In what follows, we write $\mathcal{M} = \mathcal{M}_{\mathsf{pk}}$*,* $\mathcal{C} = \mathcal{C}_{\mathsf{pk}}$*, and* $\mathcal{F} = \mathcal{F}_{\mathsf{pk}}$ *if the public key* pk *is obvious from the context. We also write* $1 = 1_{\mathcal{M}}$ *to denote the unity in* $\mathcal{M}$*.*

Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a homomorphic PKE. Then, a given ciphertext c_0 can be *re-encrypted* as follows (without knowledge of the secret key sk):

1. $r \xleftarrow{\$} \mathcal{F}$
2. $c_1 \leftarrow c_0 \cdot \mathsf{Enc}(\mathsf{pk}, 1; r)$

Observe that the resulting ciphertext c_1 decrypts to the same message as c_0 under sk. The respective formal relation between c_0 and c_1 is:

$$\mathcal{R}_{\mathsf{ReEnc}} = \{(r, (\mathsf{pk}, c_0, c_1)) \colon r \in \mathcal{F} \ \wedge \ c_1 = c_0 \cdot \mathsf{Enc}(\mathsf{pk}, 1; r)\}.$$

The following protocol is a SCSP for the re-encryption relation $\mathcal{R}_{\mathsf{ReEnc}}$ (we refer to Def. 5 for the structure of shuffle-compatible Σ-protocols):

- Com takes as input statement (pk, c_0, c_1). It then samples $\alpha \xleftarrow{\$} \mathcal{F}$ and computes commitment $a \leftarrow c_0 \cdot \mathsf{Enc}(\mathsf{pk}, 1; \alpha)$.
- Resp takes as input witness r, statement (pk, c_0, c_1), commitment a, state α, and challenge bit e. It computes response $z \leftarrow \alpha$ if $e = 0$ and $z \leftarrow r^{-1} \odot \alpha$ if $e = 1$.

– Check takes as input partial statement (pk, c_e) and the transcript (a, e, z). It outputs accept if and only if $z \in \mathcal{F}$ and $a = c_e \cdot \mathsf{Enc}(\mathsf{pk}, 1; z)$ hold true.

We now show that this protocol is a SCSP for relation $\mathcal{R}_{\mathsf{ReEnc}}$:

– *Completeness*: Follows from observation.
– *Special soundness*: The extractor Ext takes as input two accepting transcripts $(a, 0, z)$ and $(a, 1, z')$ with the same commitment a and returns $\tilde{r} \leftarrow (z')^{-1} \odot z$.
– *Shuffle-compatible special honest verifier zero-knowledge*: The simulator Sim takes as input (pk, c_e, e), chooses $z \xleftarrow{\$} \mathcal{F}$ and sets $a \leftarrow c_e \cdot \mathsf{Enc}(\mathsf{pk}, 1; z)$.

A.2 Abstract Re-Randomisation of Commitments

It is possible to define a shuffle-compatible Σ-protocol for re-randomisation of commitments in arbitrary homomorphic commitment schemes (e.g., Pedersen commitments [37]). The Coq definition and proof are given in `AbstactReRandomSigma`. Since the construction of this abstract Σ-protocol is analogous to the one of abstract re-encryption (see above), we decided not to present here. Instead, we will demonstrate in the next example that by modifying this generic Σ-protocol, we can also capture re-randomisation of homomorphic lattice-based commitments.

A.3 Re-Randomisation of Lattice-Based Commitments

We now define a SCSP for re-randomisation of lattice-based commitments. To this end, we need to relax the notion of "shuffle-compatible" (as specified below) because the randomisation space of lattice-based commitments does not allow uniformly distributed samples. This prevents us from directly revealing random values as in the abstract Σ-protocol (see above). In order to solve this issue, we will use an underlying efficient Σ-protocol with relaxed security properties to prove knowledge of the randomness. More precisely, our construction works as follows.

For the sake of concreteness, we consider the lattice-based commitment scheme proposed in [12]. We recall its details in App. C. This commitment scheme is additively homomorphic[10] and we consequently show

[10] The additively homomorphic property is restricted by the norm of the randomness but the parameters chosen in [12] allow for even up to 80 additions. This suffices for our purposes where we need only two homomorphic additions in the scheme and in the proof.

that $\boldsymbol{c}_1$ is a re-randomisation of $\boldsymbol{c}_0$ by showing that $\boldsymbol{c}_1 - \boldsymbol{c}_0$ is a commitment to 0. The respective formal relation between $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$ is:

$$\mathcal{R}_{\mathsf{ReRand}} = \{(\boldsymbol{r}, (\boldsymbol{C}, \boldsymbol{c}_0, \boldsymbol{c}_1)) : \boldsymbol{c}_1 - \boldsymbol{c}_0 = \boldsymbol{C}\boldsymbol{r} \wedge \|\boldsymbol{r}\| \leq B_r\}.$$

As mentioned above, the randomisation space $\mathcal{D}_\sigma^{n\cdot(2d+1)}$ does not allow uniformly distributed samples. We will thus employ an underlying non-interactive zero-knowledge proof $(\mathsf{Prover}, \mathsf{Verifier})$ from [12] for the relationship

$$\mathcal{R}_{\mathsf{Open}} = \{(\boldsymbol{r}, (\boldsymbol{C}, \boldsymbol{c})) \colon \boldsymbol{c} = \boldsymbol{C}\boldsymbol{r} \wedge \|\boldsymbol{r}\| \leq B_r\}.$$

This non-interactive zero-knowledge proof is particularly useful for application because it can be amortized by performing many of them in parallel. Technically, this proof is based on a Σ-protocol which provides *computational* completeness and the following *relaxed* notion of special soundness. Let $\mathcal{R}_{\mathsf{Open}}$ be as defined above, and let

$$\mathcal{R}'_{\mathsf{Open}} = \{(\boldsymbol{r}, (\boldsymbol{C}, \boldsymbol{c})) \colon \boldsymbol{c} = \boldsymbol{C}\boldsymbol{r} \wedge \|\boldsymbol{r}\| \leq B'_r\},$$

where B'_r is a certain less restrictive bound, i.e., $\mathcal{R}_{\mathsf{Open}} \subset \mathcal{R}'_{\mathsf{Open}}$ (see [12] for details). Now, the notion of special soundness that is provided by this Σ-protocol allows the knowledge extractor algorithm to also return witnesses from the larger relation $\mathcal{R}'_{\mathsf{Open}}$. This generalisation of special soundness is commonly used in the literature and results into more efficient lattice-based Σ-protocols.

In what follows, we define the Σ-protocol for the re-randomisation relation $\mathcal{R}_{\mathsf{ReRand}}$:

- Com takes as input statement $(\boldsymbol{C}, \boldsymbol{c}_0, \boldsymbol{c}_1)$, picks $\boldsymbol{\alpha} \leftarrow \mathcal{D}_\sigma^{n\cdot(2d+1)}$, and returns $\boldsymbol{a} \leftarrow \boldsymbol{C}(\boldsymbol{\alpha} - \boldsymbol{c}_0)$.
- Resp takes as input witness $\boldsymbol{r}$, statement $(\boldsymbol{C}, \boldsymbol{c}_0, \boldsymbol{c}_1)$, commitment $\boldsymbol{a}$, state $\boldsymbol{\alpha}$, challenge bit e, and returns response $\boldsymbol{z} \leftarrow \mathsf{Prover}(\boldsymbol{\alpha} + e \cdot \boldsymbol{r}, \boldsymbol{a} + \boldsymbol{c}_e)$.
- Check takes as input partial statement $(\boldsymbol{C}, \boldsymbol{c}_e)$, transcript $(\boldsymbol{a}, e, \boldsymbol{z})$, and returns $\mathsf{Verify}(\boldsymbol{z}, \boldsymbol{a} + \boldsymbol{c}_e)$.

If we generalize our definition of SCSPs (Def. 5) such that completeness can be computational and knowledge soundness can be extended to larger relations $\mathcal{R}'$, then the Σ-protocol for re-randomisation of lattice-based commitments described above is shuffle-compatible as well. Similarly, we would also need to adapt our generic compiler (Sec. 5) to being able to transform these Σ-protocols.

Since these modifications would result into even more complex machine-checked proofs, a full formal treatment is clearly beyond the scope of a single paper. As mentioned in Sec. 3, we leave this challenge as interesting future work.

A.4 ElGamal Key Shifting

We define a SCSP for re-encrypting and key-shifting a ciphertext in the ElGamal PKE (see App. B for its definition).

A given ElGamal ciphertext $c_0 = (c_0', c_0'')$ can be *re-encrypted and key-shifted* from the public key (g_0, pk) to public key (g_1, pk), where $g_1 = g_0^{\mathsf{sk}}$ and sk is the current mixers share of the key, as follows:

1. $r \xleftarrow{\$} \mathbb{Z}_q$
2. $c_1 \leftarrow ((c_0' \cdot g_0^r)^{\mathsf{sk}}, c_0'' \cdot \mathsf{pk}^r)$

The resulting ciphertext c_1 then decrypts to the same message as c_0 under the new key pair. The respective formal relation between c_0 and c_1 is:

$$\mathcal{R}_{\mathsf{ReEnc+KeyShift}} = \{((\mathsf{sk}, r), (((g_0, \mathsf{pk}), (g_1, \mathsf{pk})), c_0, c_1)) : \\ \mathsf{sk} \in \mathbb{Z}_q \ \wedge \ c_1 = ((c_0' \cdot g_0^r)^{\mathsf{sk}}, c_0'' \cdot \mathsf{pk}^r) \ \wedge \ g_1 = g_0^{\mathsf{sk}}\}.$$

The following protocol is a SCSP for the key-shifting and re-encryption relation $\mathcal{R}_{\mathsf{ReEnc+KeyShift}}$ (we refer to Def. 5 for the structure of SCSPs):

- Com takes as input witness (sk, r) and statement $(((g_0, \mathsf{pk}), (g_1, \mathsf{pk})), c_0, c_1)$. It then samples (α_0, α_1) uniformly at random from $\mathbb{Z}_q$ and computes commitment $a \leftarrow ((c_0' \cdot g_0^{\alpha_0})^{\alpha_1}, c_0'' \cdot \mathsf{pk}^{\alpha_0}, g_0^{\alpha_1})$.
- Resp takes as input witness (sk, r), statement $(((g_0, \mathsf{pk}), (g_1, \mathsf{pk})), c_0, c_1)$, commitment a, state α, and challenge bit e. It computes response $z \leftarrow (\alpha_0 - r * e, \alpha_1 / \mathsf{sk}^e)$.
- Check takes as input partial statement $(((g_0, \mathsf{pk}), (g_1, \mathsf{pk})), c_e)$ and the transcript (a, e, z). For $e = 1$ it outputs accept if and only if $({c_1'}^{z''} \cdot a'''^{z'}, c_1'' \cdot \mathsf{pk}^{z'}, g_1^{z''}) = a$ hold true. For $e = 0$ it outputs accept if and only if $((c_0' \cdot g_0^{z'})^{z''}, c_0'' \cdot \mathsf{pk}^{z'}, g_0^{z''}) = a$ hold true.

The Coq definition and proof is given in `ElGamalSigmaKeyShift`.

A.5 ElGamal Decryption.

We define a SCSP for decrypting a ciphertext in the ElGamal PKE (see App. B for its definition).

A given ElGamal ciphertext $c = (c', c'')$ can be decrypted as follows:

1. $ds \leftarrow (c')^{\mathsf{sk}}$
2. $m \leftarrow ds^{-1} \cdot c''$

We call the element ds the *decryption share*. Notice that given ds the message can be computed without knowledge of sk. The respective formal relation between c and ds is:

$$\mathcal{R}_{\mathsf{Dec}} = \{(\mathsf{sk}, (\mathsf{pk}, c, ds)) \colon \mathsf{sk} \in \mathbb{Z}_q \ \wedge \ \mathsf{pk} = g^{\mathsf{sk}} \ \wedge \ ds = (c')^{\mathsf{sk}} \ \}.$$

The following protocol is a SCSP for the decryption relation $\mathcal{R}_{\mathsf{Dec}}$ (we refer to Def. 5 for the structure of shuffle-compatible Σ-protocols):

- Com takes as input statement (pk, c, ds). It then samples α uniformly at random from $\mathbb{Z}_q$ and computes commitment $a \leftarrow ((c')^{\alpha}, g^{\alpha})$.
- Resp takes as input witness sk, statement (pk, c, ds), commitment a, state α, and challenge bit e. It computes response $z \leftarrow \alpha \cdot (\mathsf{sk})^{-e}$.
- Check takes as input partial statement (pk, c_e)[11] and the transcript (a, e, z). For $e = 0$, it outputs accept if and only if $(c')^z = a'$ and $g^z = a''$ hold true. For $e = 1$, it outputs accept if and only if $ds^z = a'$ and $\mathsf{pk}^z = a''$ hold true.

  We now show that this protocol is a SCSP for relation $\mathcal{R}_{\mathsf{Dec}}$:

- *Completeness*: Follows from observation.
- *Special soundness*: The extractor Ext takes as input two accepting transcripts $(a, 0, z)$ and $(a, 1, z')$ with the same commitment a and returns $\tilde{s} \leftarrow z \cdot (z')^{-1}$.
- *Shuffle-compatible special honest verifier zero-knowledge*: The simulator Sim takes as input (pk, c_e, e) (which equals $(\mathsf{pk}, c, 0)$ for $e = 0$ and $(\mathsf{pk}, ds, 1)$ for $e = 1$), chooses $z \xleftarrow{\$} \mathbb{Z}_q$ and sets $a \leftarrow (c_e^z, g^z)$ if $e = 0$ and $a \leftarrow (c_e^z, \mathsf{pk}^z)$ if $e = 1$.

  The Coq definition and proof is given in `ElGamalSigmaDec`.

B ElGamal Public-Key Encryption Scheme

We recall the definition of the ElGamal public-key encryption scheme [19].

Definition 8 (ElGamal PKE). *Let $\mathbb{G}$ be a group of prime order q, and let g be a generator of $\mathbb{G}$. Key generation, encryption, and decryption work as follows:*

[11] Note that $c_0 = c$ and $c_1 = ds$.

– *A public/private key pair is of the form* $(\mathsf{pk}, \mathsf{sk}) = (g^s, s)$*, where* $s \in \mathbb{Z}_q$ *is chosen uniformly at random.*
– *A message* $m \in \mathbb{G}$ *is encrypted as* $c = (c', c'') = (g^r, m \cdot \mathsf{pk}^r)$*, where* $r \in \mathbb{Z}_q$ *is chosen uniformly at random.*
– *A ciphertext* $(c', c'') \in \mathbb{G}^2$ *is decrypted as* $\tilde{m} = (c')^{-\mathsf{sk}} \cdot c''$*.*

C Lattice-Based Commitment Scheme by Del Pino et al.

We recall the lattice-based commitment scheme proposed in [3] with the parameters and notation from [12].

C.1 Definition

Let q and n be integers. Let $\mathcal{R}$ be the polynomial ring $\mathbb{Z}[X]/(X^n + 1)$, and $\mathcal{R}_q$ be the quotient ring $\mathcal{R}/q\mathcal{R}$. Let $d \in \mathbb{N}$, let $\sigma \in \mathbb{R}$, and let B_r be a positive real bound. Then, the key space is $\mathcal{R}_q^{(d+1)\times(2d+1)}$, the message space is $\mathcal{R}_q$, the opening space is $\{\boldsymbol{r} \in \mathcal{R}_q^{2d+1} \colon \|\boldsymbol{r}\| \leq B_r\}$, and the commitment space is $\mathcal{R}_q^{d+1}$.

The algorithms of the commitment scheme are defined as follows:

$\mathsf{KeyGen}(1^\ell)$
1. $\boldsymbol{A}' \xleftarrow{\$} \mathcal{R}_q^{d\times(d+1)}$
2. $\boldsymbol{A} \leftarrow [\boldsymbol{A}' | \boldsymbol{I}_d] \in \mathcal{R}_q^{d\times(2d+1)}$
3. $\boldsymbol{B} \xleftarrow{\$} \mathcal{R}_q^{1\times(2d+1)}$
4. $\boldsymbol{C} \leftarrow \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{B} \end{bmatrix} \in \mathcal{R}_q^{(d+1)\times(2d+1)}$
5. Return $\boldsymbol{C}$

$\mathsf{Com}(\boldsymbol{C} \in \mathcal{R}_q^{(d+1)\times(2d+1)}, m \in \mathcal{R}_q)$
1. $\boldsymbol{r} \leftarrow \mathcal{D}_\sigma^{n\cdot(2d+1)}$
2. $\boldsymbol{c} \leftarrow \boldsymbol{C}\boldsymbol{r} + \begin{bmatrix} \boldsymbol{0} \\ m \end{bmatrix} \in \mathcal{R}_q^{(d+1)}$
3. Return $(\boldsymbol{c}, \boldsymbol{r})$

$\mathsf{Open}(\boldsymbol{C} \in \mathcal{R}_q^{(d+1)\times(2d+1)}, m \in \mathcal{R}_q, \boldsymbol{c} \in \mathcal{R}_q^{d+1}, \boldsymbol{r} \in \mathcal{R}_q^{2d+1})$
1. If $\boldsymbol{c} - \boldsymbol{C}\boldsymbol{r} = \begin{bmatrix} \boldsymbol{0} \\ m \end{bmatrix}$ and $\|\boldsymbol{r}\| \leq B_r$, then return 1.
2. Else return 0.

C.2 Security

Del Pino et al. [12] proved that the commitment scheme is computationally binding under the Module Short Integer Solution (M-SIS) problem and computationally hiding under the Module Learning With Errors (M-LWE) problem [33].

D Transform: Cryptographic Proof

In addition to our machine-checked proof in Coq (see module `ProofOf-Shuffle`), we provide a cryptographic proof that the protocol $\Sigma_{\mathsf{Shuffle}}$ (Fig. 1) is a Σ-protocol for relation $\mathcal{R}_{\mathsf{Shuffle}}$ (see Sec. 5).

Lemma 1 (Completeness). *If the protocol Σ (Def. 5) guarantees completeness, then the protocol $\Sigma_{\mathsf{Shuffle}}$ (Fig. 1) guarantees completeness.*

Proof. Let

$$((\tilde{a}^j)_{j\in[\tau]}, e, (\pi_z, ((\tilde{z}^j)_{j\in[\tau]})))$$

be the transcript of an arbitrary run of $\Sigma_{\mathsf{Shuffle}}$ with input

$$(x, y) = (((x^j)_{j\in[\tau]}, \pi), (\mathrm{prm}, (c_0^j)_{j\in[\tau]}, (c_1^j)_{j\in[\tau]})) \in \mathcal{R}_{\mathsf{Shuffle}}.$$

We have that the output of V is 1 if and only if for all $j \in [\tau]$

$$1 = \mathsf{Check}((\mathrm{prm}, c_e^{\tilde{\pi}_z(j)}), (\tilde{a}^j, e, \tilde{z}^j))$$

holds true. Since the prover runs its honest program P, we have that

$$\begin{aligned}
&\mathsf{Check}((\mathrm{prm}, c_e^{\tilde{\pi}_z(j)}), (\tilde{a}^j, e, \tilde{z}^j)) \\
=&\begin{cases} \mathsf{Check}((\mathrm{prm}, c_0^{\pi_a(\pi(j))}), (a^{\pi_a(j)}, 0, z^{\pi_a(j)})) & , e = 0 \\ \mathsf{Check}((\mathrm{prm}, c_1^{\pi_a(j)}), (a^{\pi_a(j)}, 1, z^{\pi_a(j)})) & , e = 1 \end{cases} \\
=&\begin{cases} \mathsf{Check}((\mathrm{prm}, c_0^{\pi(j)}), (a^j, 0, z^j)) & , e = 0 \\ \mathsf{Check}((\mathrm{prm}, c_1^j), (a^j, 1, z^j)) & , e = 1 \end{cases}
\end{aligned}$$

holds true for all $j \in [\tau]$, where the second equality follows from the bijective property of π_a. By the definition of $\mathcal{R}$, we have that

$$(x^j, (\mathrm{prm}, c_0^{\pi(j)}, c_1^j)) \in \mathcal{R}$$

holds true for all $j \in [\tau]$. Therefore, completeness of $\Sigma_{\mathsf{Shuffle}}$ follows from completeness of Σ.

Lemma 2 (Special soundness). *If the protocol Σ (Def. 5) guarantees special soundness, then the protocol $\Sigma_{\mathsf{Shuffle}}$ (Fig. 1) guarantees special soundness.*

Proof. Let Ext be the extractor of the underlying protocol Σ. We now construct an extractor $\mathsf{Ext}_{\mathsf{Shuffle}}$ for $\Sigma_{\mathsf{Shuffle}}$ as follows. The extractor $\mathsf{Ext}_{\mathsf{Shuffle}}$ takes as input a public statement $(\text{prm}, (c_0^j)_{j\in[\tau]}, (c_1^j)_{j\in[\tau]})$ as well as two accepting transcripts $((\tilde{a}^j)_{j\in[\tau]}, 0, (\pi_z^0, ((\tilde{z}_0^j)_{j\in[\tau]})))$ and $((\tilde{a}^j)_{j\in[\tau]}, 1, (\pi_z^1, ((\tilde{z}_1^j)_{j\in[\tau]})))$ for $(\text{prm}, (c_0^j)_{j\in[\tau]}, (c_1^j)_{j\in[\tau]})$. For each $j \in [\tau]$, the extractor runs:

1. $x^j \leftarrow \mathsf{Ext}((\text{prm}, c_0^{\tilde{\pi}_z^0(j)}, c_1^{\tilde{\pi}_z^1(j)}), (\tilde{a}^j, 0, \tilde{z}_0^j), (\tilde{a}^j, 1, \tilde{z}_1^j))$
2. $\pi(\tilde{\pi}_z^1(j)) \leftarrow \tilde{\pi}_z^0(j)$

Then, we have that $((x^j)_{j\in[\tau]}, \pi)$ is a witness for $(\text{prm}, (c_0^j)_{j\in[\tau]}, (c_1^j)_{j\in[\tau]})$.

Lemma 3 (Special honest-verifier zero-knowledge). *If the protocol Σ (Def. 5) guarantees shuffle-compatible special honest-verifier zero-knowledge, then the protocol $\Sigma_{\mathsf{Shuffle}}$ (Fig. 1) guarantees special honest-verifier zero-knowledge.*

Proof. Let Sim be the "strong" simulator of Σ (Def. 5). We now construct a "standard" simulator $\mathsf{Sim}_{\mathsf{Shuffle}}$ for $\Sigma_{\mathsf{Shuffle}}$ (Def. 4). The simulator $\mathsf{Sim}_{\mathsf{Shuffle}}$ takes as input public statement $(\text{prm}, (c_0^j)_{j\in[\tau]}, (c_1^j)_{j\in[\tau]})$ and challenge e, and then performs the following steps:

1. $\pi_s \xleftarrow{\$} \Pi$
2. $\forall j \in [\tau]\colon (a^j, e, z^j) \leftarrow \mathsf{Sim}(\text{prm}, c_e, e)$
3. return $((a^{\pi_s(j)})_{j\in[\tau]}, e, (z^{\pi_s(j)})_{j\in[\tau]})$

Since π_s is permutation, we have that $((a^{\pi_s(j)})_{j\in[\tau]}, e, (z^{\pi_s(j)})_{j\in[\tau]})$ is an accepting transcript for $(\text{prm}, (c_0^j)_{j\in[\tau]}, (c_1^j)_{j\in[\tau]})$. Furthermore, since π_s is chosen uniformly at random, it has the same distribution as π_z in the real conversation. Thus, the output of $\Sigma_{\mathsf{Shuffle}}$ has the same distribution as the real transcript.