Did you mix me? Formally Verifying Verifiable Mix Nets in Electronic Voting

Thomas Haines*, Rageev Goré†, and Bhavesh Sharma†
*Dept of Mathematical Sciences,
Norwegian University of Science and Technology
†Research School of Computer Science,
Australian National University

***Abstract*—Verifiable mix nets, and specifically, proofs of (correct) shuffle, are a fundamental building block in numerous applications: these zero-knowledge proofs allow the prover to produce a public transcript which can be perused by the verifier to confirm the purported shuffle. They are particularly vital to verifiable electronic voting, where they underpin almost all voting schemes with non-trivial tallying methods. These complicated pieces of cryptography are a prime location for critical errors which might allow undetected modification of the outcome.**

The best solution to preventing these errors is to machine-check the cryptographic properties of the design and implementation of the mix net. Particularly crucial for the integrity of the outcome is the soundness of the design and implementation of the verifier (software). Unfortunately, several different encryption schemes are used in many different slight variations which makes it infeasible to machine-check every single case individually. However, a particular optimized variant of the Terelius-Wikström mix net is, and has been, widely deployed in elections including national elections in Norway, Estonia and Switzerland, albeit with many slight variations and several different encryption schemes.

In this work, we develop the logical theory and formal methods tools to machine-check the design and implementation of all these variants of Terelius-Wikström mix nets, for all the different encryption schemes used; resulting in provably correct mix nets for all these different variations. We do this carefully to ensure that we can extract a formally verified implementation of the verifier (software) which is compatible with existing deployed implementations of the Terelius-Wikström mix net. This gives us provably correct implementations of the verifiers for more than half of the national elections which have used verifiable mix nets.

Our implementation of a proof of correct shuffle is the first to be machine-checked to be cryptographically correct and able to verify proof transcripts from national elections. We demonstrate the practicality of our implementation by verifying transcripts produced by the Verificatum mix net system and the CHVote evoting system from Switzerland.

I. Introduction

In modern e-voting schemes, extensive use is made of zero-knowledge proofs (ZKPs), first studied by Goldwasser, Micali, and Rackoff [31], in which a prover and a verifier follow a specific protocol to produce a public proof of a particular statement. The proof produced by the prover is "zero knowledge" because it leaks no information other than the truth of the statement. Of course, the statement itself may leak information: for example, a proof of correct decryption leaks no information, but the decryption itself obviously leaks the message of the ciphertext. ZKPs enable cryptographic, rather than formal, verification of the election result (statement) without revealing information which adversely affects the privacy of the election. In the context of ZKPs, we say that a party, and its output, is honest if it follows the protocol and denote a protocol run, called a transcript, as valid if the verifier accepts (the transcript). Some of these cryptographic proofs are of a particularly simple and efficient form known as a sigma protocol; a class first defined and analysed by Cramer in his PhD Thesis [18]. The other main type of ZKPs used in electronic voting are verifiable mix nets.

Mix nets were first introduced by Chaum [15] as a solution to the traffic analysis problem in which an adversary is able to extract useful information from patterns of communication, even when that communication is encrypted. The traffic analysis problem can be thought of, more generally, as the set of problems that arise by the ability to link the messages between sets of senders and receivers. Mix nets therefore consist of a finite sequence of authorities (mixers), each of which permutes (shuffles) and hides the relationship between its inputs and its outputs. Informally, a mix net has privacy if it hides the relationship between the initial input and the final output, provided one of the mixers is honest.

For example, in electronic voting, the input to a mix net is usually a sequence of encrypted ballots, ordered according to a separate list of voter-ids. Thus the first encrypted ballot is the ballot of the voter with id 1, and so on. The intention is that the encrypted ballots will be decrypted before counting. But naively decrypting the input sequence, as is, allows the election authority to learn exactly how each voter voted since we can link the i-th encrypted ballot to the i-th voter, thus breaking the privacy of the voters. By first using a mix net, as an irreversible black box, to permute the initial sequence of encrypted ballots, we can ensure that the i-th encrypted ballot in the permuted output sequence is no longer linked to the i-th voter. Decrypting the output sequence now preserves the privacy of the voters. Clearly, the mix net must maintain integrity in that it must not delete, add or change the contents of any of the encrypted ballots. Informally, we say that an output sequence of encrypted messages is a correct shuffle of the input sequence if the integrity is maintained.

Unfortunately, vanilla mix nets allow the mixers to change the content without detection. Indeed, a mix net which provides no integrity cannot even provide privacy since a dishonest mixer can substitute all but one of the honest inputs with

inputs known to the adversary: the adversary then breaks the privacy of the targeted honest input simply by looking at the output and seeing which value is different from the substituted ones.

Neff [49] and Furukawa and Sako [28] independently suggested verifiable mix nets which allow the mixer to prove the correctness (integrity) of its shuffle using complicated zero-knowledge proofs (of shuffle). These schemes are used to modify the vanilla mix net by requiring each party to produce a zero-knowledge-proof that it shuffled correctly in addition to performing the shuffle. Subsequently, many verifiable mix nets [2], [3], [8], [22]–[24], [28], [38], [40], [42], [45], [48], [49], [55], [59], [61]–[63], [66] have been proposed and many of the designs have been shown to be flawed [43], [44], [46], [47], [60].

Beyond the issue of uncaught design flaws is the further issue of implementing the design correctly. In electronic voting, mix nets are by far the most complicated pieces of cryptography implemented and overwhelmingly see utterly inadequate scrutiny[1], even compared to the poor base level for deployed e-voting schemes, more generally. Given the long list of verifiable mix nets which have been shown to be flawed, we clearly need a trusted methodology for checking the verifiability of verifiable mix nets—both in the design and implementation. Here, we give such a trusted methodology based upon the interactive proof-assistant Coq. In particular, we have exploited the significant commonalities in the mix nets used by deployed e-voting schemes to structure our encodings in a way which allows ease of reuse. Consequently, we are able to extract provably correct software for verifying transcripts from several mix net implementations.

We are primarily interested in mix nets as used in national elections, and in this setting, the Terelius-Wikström [59] and Bayer-Groth [8] verifiable mix nets are used.[2] We choose Terelius-Wikström because it is more commonly used.

When any work claims to machine-check something, the immediate question is check for what? We give an informal description here to provide some context to the rest of our introduction; In our case, we check the following properties of the (zero-knowledge) proof of shuffle:

- **Completeness:** An honestly generated proof will be accepted by an honest verifier.
- **Soundness:** A dishonest party will only be able to construct a valid proof for a false statement with negligible probability. [3]
- **Honest verifier zero-knowledge:** If the challenges are sampled independently and uniformly then the distribution of the honest transcripts can be simulated without knowing the witness.

We note that in addition to the properties we check, the act of encoding the definition in a formal language with such a rich type system has certain advantages. For example, we are required to formally define the types of our functions inside Coq. If we run the functions, the inputs are checked with respect to the claimed type, which, as we shall see, has positive implications for privacy and integrity.

A. Mix nets

Mix nets can be classified into two categories, decryption mix nets and re-encryption mix nets, which we now explain. Recall that our aim is to take a sequence $m_1, m_2, \cdots, m_k$ of messages and produce an output sequence which is a permutation of this sequence without deletions, additions or mutations of the underlying messages.

Decryption mix nets were proposed by Chaum [15]. In a decryption mix net, there are n authorities (mixers) in some fixed order $a_1, a_2, \cdots, a_n$ who each publish a public key $pk_1, pk_2, \cdots, pk_n$. A sender who wishes to submit a plaintext message m to the mix iteratively encrypts m under the public keys of the mixers in reverse order to produce $enc(m) := Enc_{pk_1}(...(Enc_{pk_n}(m))...)$. This nested ciphertext $enc(m)$ is then submitted as one element of the input sequence $enc(m_1), enc(m_2), \cdots, enc(m_k)$ to the first mixer. Each mixer in turn, decrypts each element in its input sequence using its private key and then outputs a permutation of its decrypted sequence as the input sequence for the next mixer (if any).

Unfortunately, decryption mix nets are less computationally efficient than re-encryption mix nets and it is normally harder to construct proofs of correct shuffle for them. More specifically, proofs of correct shuffle are easier if the encryption scheme obeys certain homomorphic properties [36] but the encryption schemes typically used in decryption mix nets lack these homomorphic properties.

Re-encryption mix nets were first proposed by Park et al. [52]. The basic idea is as follows. First, use a threshold public key encryption scheme that allows re-encryption of a given ciphertext without knowing the secret key. (Such encryption schemes naturally have certain homomorphic properties between the ciphertext space and the randomness space.) Then, the mixers jointly construct a public key, for the mix, for which they all hold secret key shares. A sender encrypts its plaintext message under the joint public key. Each mixer, in its turn, re-encrypts (re-randomizes) its input and permutes the output, its input in the case of the first mix is the input to the mix net otherwise it is the output of the mix before it. Once all mixers have mixed, the authorities jointly decrypt the output using the threshold decryption method.

The reader may have noticed that we, following the literature, equivocate on the meaning of the phrase mix net. Specifically, the term is sometimes used to refer to a proof of shuffle—for instance the Terelius-Wikström mix net—but also used to refer to a multi-party protocol which takes a vector of ciphertexts and produces a shuffle of them. The notion of integrity for the two meanings are effectively equivalent in our context but the privacy is not. Specifically, the information leaked by the output of the mix is not considered by the

[1]This assertion is based on personal correspondence with many of the leading experts who regularly examine e-voting schemes.

[2]There are a few other verifiable mix nets that have been used in real government elections but they are significantly less common.

[3]Formally we prove special soundness which is known to imply soundness.

definition of zero-knowledge (privacy) for the proof of shuffle, by contrast, it is considered by the privacy definition for the multi-party protocol. The following example is given to demonstrate the difference.

Example 1. *Consider a re-encryption mix net which permutes without re-encrypting. Clearly, no such mix net can provide meaningful privacy since the pertinent information about the permutation can be easily calculated by looking at the input and output. However, any proof of shuffle for the scheme might well be zero-knowledge (have privacy) since the proof transcripts could still be simulated.*

As our example demonstrates, the zero-knowledge properties of the proof of shuffle are necessary but not sufficient for the privacy of the mix net as whole. In appendix E we give a paper proof that the properties of the proof of shuffle (which we machine checked) plus the indistinguishability against chosen plaintext attack (IND-CPA) properties of the encryption scheme suffice for the privacy of the mix net. Unless clear from context, we will use mix net to refer to a proof of shuffle.

As stated above, the homomorphic properties of the encryption scheme are also useful in constructing the proof of shuffle. For this reason, the overwhelming majority of verifiable mix nets are based on re-encryption mix nets; this includes both Terelius-Wikström [59] and Bayer-Groth [8], the two verifiable mix nets most commonly used in e-voting. Both Terelius-Wikström and Bayer-Groth are, strictly speaking, zero-knowledge arguments of correct shuffle, rather than zero-knowledge proofs, since they are only sound if the commitment scheme used is binding.[4] The parameters of this commitment scheme are called a common reference string and in both of these mix nets can be securely generated without trust assumptions. The process is straightforward and we discuss in our future work how this might be verified in Coq.

The particular mix net most commonly used in verifiable voting is an optimized variant of the Terelius-Wikström mix net. There are many slight variations but essentially it involves taking Terelius-Wikström's original mix net [59], which works for any homomorphic encryption scheme for which a sigma protocol for re-encryption is known, and restricting to the case where the ciphertext space has prime order (or sufficiently close to). This allows the removal of Fujisaki-Okamoto commitments [27], based on an RSA modulus, which hampered the efficiency of the original mix net. In practice, everyone using the Terelius-Wikström proof of shuffle uses an optimized variant which avoids the use of Fujisaki-Okamoto commitments. This style of mix net is most prominently implemented in Verificatum [65] and CHVote2.0 [58]; in addition, it has been implemented by various companies and academic groups. The various implementations have been used to mix millions of votes in elections ranging from local organizations to national elections. It was proved that the original mix net worked for a wide class of encryption schemes; it was folklore that the optimized variant also worked for a significant class of schemes but no paper proof was ever published and the exact class of schemes for which the result should hold was undefined. In this work, we give a formal definition of this theorem and give a machine checked proof.

The proof of correct shuffle in verifiable mix nets, as with zero-knowledge proofs more generally, can be interactive or non-interactive. Generally the proofs are defined, and proven secure, in the interactive variant and then made non-interactive using the Fiat-Shamir transform [25]. In contexts where the statement being proved is adaptive, as in mix nets, care must be taken to use the strong variant of the transform [11]. The strong transform involves replacing the challenge from the verifier with the hash of the statement and commitment. All the verifiable mix nets proved in this work are in the interactive variant; they can be made non-interactive with a one-line change to the verifier but proving the security of the Fiat-Shamir transform inside Coq is left as future work.[5]

B. Verifiable Electronic Voting

Secure electronic voting is a difficult security problem with numerous competing constraints. The key approach to securing electronic elections is so-called "verifiable electronic voting": that is, each component must produce public evidence to allow its results to be verified. The techniques involved in producing and verifying this evidence are largely cryptographic. Rivest famously called this property of public verifiability "software independence" [54]; however, while these technique do make the correctness of the announced outcome independent of the software used to *produce* the evidence, it does *not* make it independent of the software used to *check* the evidence.

Unfortunately, implementing these techniques is an extremely error prone process. For example, the Swiss post system, intended and used for national elections in Switzerland, contained an abundance of critical errors when opened for public review, despite the "thorough" certification process it had previously undergone; interested readers should read [35]. Some of these errors were independently discovered by Haenni [32]. Additional prominent failures and issues in allegedly end-to-end verifiable systems have included the iVote system deployed in the Australian state of New South Wales [37], and the e-voting system used in national elections in Estonia [39], [57]. Many general issues have also been discovered [10], [11], [17] which need to be carefully avoided in any implementation, but most of these issues were at one time present in the Helios end-to-end verifiable e-voting system [1] used by the International Association for Cryptologic Research. Many of these issues are examples of the weakness in precision of traditional paper based proofs; while the work required to construct a machine-checked proof is much greater, the advantages for security are significant. Part of what we aim

[4]Even more strictly speaking, Terelius-Wikström is presented as a zero-knowledge proof of the disjunctive relation that either a correct shuffle occurred or the binding property of the commitment scheme was broken.

[5]Proving the security of the transform requires reasoning about rewinding in the random oracle model which to the authors' knowledge is not supported in Coq, EasyCrypt or other equivalent interactive provers.

to show in this work is that if the machine-checked proof is modular then it can cover a large range of deployed variants.

More recently, Gaudry [29] demonstrated two attacks on the Moscow Internet voting system. The first issue was that key sizes were chosen incorrectly, so that the decisional Diffie-Hellman problem was not hard. This would not be directly caught by our machine-checked proofs, nor indeed by any reduction, which assumes the underlying problem is hard. The second issue was that ElGamal, as implemented, was not semantically secure. More specifically, the values representing the candidates were not in the message space of ElGamal, thereby leaking the identity of the chosen candidate. This would be caught by our Coq formalisation; our type checking alone would catch this issue.

Voting systems are large collections of software and formally verifying everything, to avoid the errors above, is hardly a practical goal. Furthermore, even if the software was verified to be correct, external parties would be unable to tell if this software was actually being run.[6] However, as Rivest observed, we do not need to prove the correctness of all the election software for integrity; we only need to prove the soundness of the verifier used to check the public evidence. This is precisely the fact that we exploit in this work. While, we do in fact define the entire mix net and prove its cryptographic properties, we have little desire to encourage vendors to use our implementation of the mixer and the prover. The soundness property of the verifier ensures that if it accepts, then, with overwhelming probability, the mixing was done correctly. Thus it is unnecessary for the integrity of the mix net (and the election) to use our verified mixer and prover since it suffices to use our verified verifier; however, if the vendors wished to ensure that the privacy properties we proved hold, our prover would need to be used and the randomness would need to be sampled uniformly.

Verifiable electronic voting schemes can be largely grouped into two categories: those based on homomorphic tallying and those based on mix nets. Homomorphic electronic voting schemes utilise homomorphic cryptosystems to allow the encrypted ballots to be publicly computed (tallied) before decryption. This approach has a high degree of conceptual simplicity but does not presently scale beyond fairly basic electoral systems, such as first-past-the-post. The issue is that the more complicated tallying methods require a large number of both multiplications and additions to compute the tally; if this were to be done homomorphicly it would require fully homomorphic encryption and, hence, at present this is not practical.

More complicated elections, which notably includes many of the governmental elections using electronic voting such as Norway, Estonia, Switzerland and Australia, use mix nets. The reasons are myriad: in Norway, the ballot structure and counting is too complicated (see [30]); in Switzerland, the use of write-in candidates prevents homomorphic tallying; and in Australia, the single transferable vote electoral system is not readily amenable to homomorphic tallying.[7] There are several schemes and designs used in these countries but they have in common that, at some point, the encrypted ballots are shuffled using a verifiable mix net to break the link between the voters and the encrypted ballots.

End-to-end verifiability is often broken down into three sub-properties: namely, cast-as-intended, collected-as-cast and counted-as-collected. Cast-as-intended captures the idea that the encrypted ballot the voter casts contain the vote they intended. Collected-as-cast captures the idea that the cast vote was collected without tampering. Counted-as-collected captures the idea that the collected ballots are properly counted. Correct mixing falls within the counted-as-collected property. Therefore, any use of our verifier in real elections should be done alongside mechanisms which check cast-as-intended and collected-as-cast along with other properties such as voter eligibility.

Thus it should be clear that the privacy of the proof of shuffle falls within the scope of our work but the privacy of the election, as a whole, does not.

C. Outline

Having introduced mix nets and verifiable electronic voting, we have hopefully convinced the reader that this is a crucial area of security in need of additional research to secure the mix nets deployed in national elections. In the next section, background, we will discuss more of the prior work. In section III we will detail and clarify our contribution. We will then introduce various building blocks in section IV. After that, in section V we will formally define the class of encryption schemes for which we have proved the proof of shuffle. In section VI we will define the Terelius-Wikström mix net and state its security theorem, which we have proven in Coq. Following this, in section VII we will define several encryption schemes in Coq and show that they are in the class which can be mixed. Finally, we will discuss the applications of our work to verifying national elections (section VIII) before concluding in section IX.

II. Background

In this work we machine-check our results in the Coq interactive theorem prover; interactive theorem provers are tools which allow encoding of mathematically rigorous definitions and algorithms, stating desired properties as theorems to be proved, and interactively proving (machine-checking) that the definitions imply these theorems.

We used the Coq theorem prover [12] which is based upon Coquand's Calculus of Constructions and has been developed over decades. A significant body of work has already been completed on verifying cryptography in Coq, most notably, the CertiCrypt project [6]. The CertiCrypt project can be viewed as an extension of Coq to allow game-based cryptographic proofs. It has been used to give numerous machine-checked

[6]The standard techniques for remote attestation do not work in the threat model normally used for national elections.

[7]There are various approaches but they rely on trusted parties or are many orders of magnitude slower.

cryptographic proofs, some of which we cite below. The proofs we give are straight reductions without utilizing game hopping and for this reason we do not use CertiCrypt; there are interesting extensions to our work which could make use of both our results and existing results in CertiCrypt. However, to the best of our knowledge, CertiCrypt appears to have been abandoned in favor of EasyCrypt.[8] EasyCrypt is a separate interactive proof system which is designed specifically for verifying cryptographic proofs. Early versions 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 and has not itself been proved correct.

A. Verification and Code Extraction Via Coq

Note: This section original appeared in our previous work [34] and we include it here for completeness.

We now explain how to use the interactive theorem prover called Coq [12] to: encode specifications; encode functional programs; and to verify them correct against these encoded specifications to finally extract the code corresponding to the verified functional programs.

We use a running example, we describe how to specify, verify and extract code using Coq.

1) Classical Logic and Constructive Logics: We assume familiarity with classical logic, but list three of its defining features:

1) excluded middle: every statement is true or false;
2) non-contradiction: no statement is both true and false; and
3) non-empty domain of discourse: the values of variables such as x and y are drawn from a non-empty set.

The first two combine to make classical logic bivalent: every statement is either true or else false. The third means that the formula $\forall x.\varphi(x)$ cannot be true vacuously, and the formula $\exists x.\varphi(x)$ cannot be false vacuously, by the domain of x being empty. Philosophers, mathematicians and computer scientists have invented a multitude of alternatives to classical logic, usually by removing one or more of these principles, and intuitionistic logic is arguably the most prominent alternative. Briefly, intuitionistic logic elides the law of the excluded middle and demands that for an existential $\exists x.\varphi(x)$ to be true, we must find a witness a from the domain of discourse which makes $\varphi(a)$ true. Thus we cannot assume $A \vee \neg A$, and then proceed by cases on A and $\neg A$. Nor can we proceed by contradiction whereby we assume $\neg A$, show that this leads to a contradiction, and hence conclude that A must hold. Intuitionistic logic is "constructive" because to conclude $A \vee B$, we must construct a proof of A or a proof of B, and to conclude $\exists x.\varphi(x)$, we must construct a witness as explained above. Consequently, finding proofs in intuitionistic logic is usually harder than in classical logic.

[8] See http://certicrypt.gforge.inria.fr/#related

2) An overview of the Coq proof engine: At all stages of a Coq proof, the proof engine maintains a collection of labelled hypotheses or assumptions $t_1 : \alpha_1, \cdots, t_n : \alpha_n$, one current goal γ_0, and a list of further goals $\gamma_1, \cdots, \gamma_m$ as illustrated below at left.

$$\begin{array}{c} t_1 : \alpha_1 \\ t_2 : \alpha_2 \\ \vdots \\ t_n : \alpha_n \\ \hline\hline \gamma_0 \\ \gamma_1, \cdots, \gamma_m \end{array} \qquad \begin{array}{c} \alpha_2 \to \beta \\ \alpha_2 \\ \beta \\ \vdots \\ \alpha_n \\ \hline\hline \gamma_0 \\ \gamma_1, \cdots, \gamma_m \end{array} \qquad \begin{array}{c} \forall x, \beta_1(x) \to \beta_2(x) \\ \alpha_2 \\ \vdots \\ \alpha_n \\ \hline\hline \beta_1(x)\theta \\ \gamma_1, \cdots, \gamma_m \end{array}$$

Ignoring the labels t_i for now, proof construction then either proceeds in a forward manner or a backward manner using a finite collection of predefined "natural deduction" rules. For example, as shown in the centre above, if α_1 is of the form $\alpha_2 \to \beta$, then we may extend the assumptions with β by apply the rule of *modus ponens* which intuitively captures "if $\alpha_2 \to \beta$ and α_2 then β". Alternatively, as shown in the rightmost figure, if α_1 is of the form $\forall x, \beta_1(x) \to \beta_2(x)$, then we can pattern-match γ_0 with $\beta_2(x)$ to obtain a substitution θ such that $\beta_2(x)\theta = \gamma_0$, and then replace the goal γ_0 with $\beta_1(x)\theta$, to "backchain" on the implicational assumption instance $\beta_1(x)\theta \to \beta_2(x)\theta$. The pattern matching required is usually higher-order matching. Once we have proved γ_0, Coq automatically replaces it with γ_1 to keep track of sub-goals and the current proof state. There are many facilities for reordering subgoals, composing rules into tactics, and using libraries of previously proved results. But Coq will only accept a putative proof if all rules are used correctly, thereby guaranteeing overall correctness.

3) Proofs as programs and code extraction: The syntax of the basic propositions α and β is user-definable and is based upon a highly sophisticated type-theory which allows all of the logical manipulations mentioned above to be interpreted purely inside a lambda-calculus of terms with the logical formulae as types where $t_1 : \alpha_1$ is now read as "term t_1 is of type α_1". For example, the *modus ponens* rule corresponds to function application: "if f is a function from domain type α_2 to range type β, and t is of type α_2 then $f(t)$ is of type β". By using the type annotations, we can also read $t : \alpha_2$ as "t is a proof of α_2", read $f : \alpha_2 \to \beta$ as "f is a function that converts proofs of α_2 into proofs of β", and read $f(t) : \beta$ as "$f(t)$ is a proof of β". Thus a successful proof corresponds to a computable function in the underlying lambda-calculus. Coq provides "extraction" facilities to turn such computable functions into actual code in one of programming languages OCaml, Haskell or Scheme.

4) Program Verification via Coq: Coq also provides a vast array of pre-defined constructs from functional programming such as natural numbers, lists, pattern matching and explicit function definitions.

Below, we explain one way to produce verified programs via Coq using addition of two natural numbers as an example.

As in the sequel, we first give a natural language definition as might be found in a mathematics text, then its encoding into Coq, followed by an explanation of the encoding. Doing so is important as it helps to ensure that the encoding really does do the job we intend it to do.

Definition 1. *The set $mynat$ is the smallest set formed using the following clauses:*

1) *the term O is in $mynat$;*
2) *if the term n is in $mynat$ then so is the term $S\ n$;*
3) *nothing else is in $mynat$.*

```
Inductive mynat : Set :=
  | O : mynat              (* O is a mynat *)
  | S : mynat -> mynat.    (* S of a mynat is a mynat *)
```

Here, the first line encodes that $mynat$ is of type Set and the vertical bar separates the two subclauses of the encoding. The terms O and S are known as constructors and anything in between "(*" and "*)" are comments. The first subclause illustrates that the colon can also be read as set membership $\in$ while the second clause illustrates that the constructor S is actually a function that accepts a member from $mynat$ and constructs another member of $mynat$ by prefixing the given member with S. Thus the explicit mention of n in the natural language definition is elided. Clause (3) of the natural language definition is encoded by the declaration `Inductive`. Intuitively, the natural numbers are the terms $O, (S\ O), (S\ (S\ O)), \cdots$ corresponding intuitively to $0, 1, 2, \cdots$.

Definition 2 (Specification of addition)**.** *Adding O to any natural number m gives m, and for all natural numbers n, m, and r, if adding n to m gives r then adding $(S\ n)$ to m gives $(S\ r)$.*

```
Inductive add: mynat -> mynat -> mynat -> Prop :=
  | addO: forall m, (add O m m)
  | addS: forall n m r, add n m r -> add (S n) m (S r).
```

Here, the notation $mynat \rightarrow mynat \rightarrow mynat \rightarrow Prop$ encodes that add is ternary and that it is a "Proposition" which returns either true or else false, but in intuitionistic logic rather than classical logic. Our specification of addition is encoded as a ternary predicate $add\ n\ m\ r$ that is true iff "adding n to m gives r", based purely on the only two ways in which we can construct the first argument: either it is O, or it is of the form $(S\ \cdot)$. There are now two ways to proceed to "extract" the code for an implementation "myplus" of the predicate add. The first is to write our own function $myplus$ inside Coq and to prove that the function implements the specification of addition. The second is to prove a theorem inside Coq such that the proof encodes the function implicitly. In both cases, the "extraction" facilities of Coq allow us to produce actual code in OCaml, Haskell, or Scheme. The encoding below is our hand-crafted function $myplus$ in which the "where" keyword allows an infix symbol $+$ for $myplus$ and $\Rightarrow$ (not $\rightarrow$) indicates the return value of the function:

```
Fixpoint myplus (n m: mynat) : mynat :=
  match n with
  | O   => m
  | S p => S (p + m)
  end
  where "p + m" := (myplus p m).
```

Our function is correct if it implies the specification, as expressed and encoded below.

Theorem 1. *For all natural numbers n, m, r, if $r = myplus\ n\ m$ then $add\ n\ m\ r$ is true.*

```
Theorem myplus_correct:
  forall n m r: mynat, (r = myplus n m) -> (add n m r).
Proof.
   induction n. intro m. intro r. intro H. simpl in H.
   subst r. apply addO. intros m r H. rewrite H.
   simpl myplus1. apply addS. apply IHn. reflexivity.
Qed.
```

The text shown between the words `Proof` and `Qed` consists of commands typed in by the user to guide Coq to the proof of the theorem. That is, the user interacts with Coq to obtain the proof, with Coq checking each step to ensure that it is acceptable. The Coq extraction mechanism turns our function "myplus" into Ocaml, Haskell or Scheme code giving us a program which is provably correct with respect to our specification of addition.

We can also reason about our specification itself inside Coq. For example, the theorem below encodes that our definition of addition is commutative:

Theorem 2. *For all natural numbers n, m, r, if $add\ n\ m\ r$ then $add\ m\ n\ r$*

```
Theorem add_commutative:
  forall n m r: mynat, (add n m r) -> (add m n r).
Proof. ... Qed.
```

In the sequel, we give all of our theorems in both plain text and in Coq to enable the reader to confirm that our encodings do indeed capture our intentions.

B. Related Work

There has been significant focus on using automated tools for proof creation or checking in e-voting. Starting with the work in ProVerif of Delaune et al. [21] to reason formally about privacy; ProVerif was subsequently used to reason formally about cryptographic verifiability by Smyth et al. [56]. EasyCrypt and Tamarin have been used more recently to formally verify various e-voting schemes [13], [41] with the Cortier et al. [16] work on (cryptographic) verifiability and privacy of Belenios [14] being one of the best examples. However, all of these works are interested in the privacy or integrity of the (theoretical) scheme, not the security of the implementation. This makes the works complementary with ours, since they show the e-voting scheme to be secure if the underlying cryptography is correctly implemented and we prove the security of the implementation of the underlying cryptography. Moreover, these tools are not themselves formally verified, are comparatively immature compared to Coq, and have been shown to contain bugs themselves. We

stress that the above list of works straddles two different techniques: model checking and theorem proving. The model checkers, such as Tamarin, only establish symbolic proofs which are qualitatively different from the computational proofs of interactive theorem proving.

Previous work related to formally verifying sigma protocols includes Barthe et al. [7]. Almeida et al. [4] developed a compiler which accepts an abstract description of the statement to be proved and produces an implementation of a sigma protocol for that statement along with an Isabelle/HOL proof that the sigma protocol is correct. Both of these works were combined and expanded upon by Almeida et al. [5]. These works could be used to produce verifiable electronic voting systems; they could also be use to produce verifiable mixnets when combined with our work here. However, the automatically produced implementation will very likely not be compatible with the deployed schemes since the implementation produced, while sound for the same statement, will differ in implementation aspects.

Haines et al. [34] demonstrated how interactive theorem provers and code extraction can be used to gain much higher confidence in the outcome of elections; they achieved this by using the interactive theorem prover Coq and its code extraction facility to produce verifiers, for verifiable voting schemes, with the verifiers proven to be cryptographically correct. They also showed that it was possible to verify the correctness (completeness, soundness and zero-knowledge) of a proof of correct shuffle. However, the primary focus of their work was on formally verified verifiers for elections schemes using sigma protocols and their work on mix nets only supported one encryption scheme (ElGamal) with a maximum of two inputs; hence it cannot be used for any real election. To the best of our knowledge, the work of Haines et al is the only work to do machine-checked proofs of shuffle. Here, we have removed both limitations: not only do we allow an unbounded number of inputs, which Haines et al. speculated should follow, but we generalise the proof to hold for a significant class of cryptosystems (encryption schemes). This is not only technically interesting but crucial to practice since most of the e-voting schemes used in national elections do not use standard ElGamal.

The work of Haines et al. [34], and hence ours as well, differs from most of the related work by avoiding (direct) probabilistic reasoning. It is relatively straightforward to turn most honest probabilistic algorithms into functions which take the random coin as input. Avoiding probabilistic reasoning for security properties is more difficult but it is possible because the properties of sigma protocols (see Sec. IV-D), and related protocols with more rounds, can be expressed without resorting to probabilities. Two of the three properties (completeness and special soundness) are normally stated without any probabilities but honest-verifier zero-knowledge is stated using probabilities. We can circumvent this issue, by first observing that we can describe both a set and a distribution over the set as a multiset. We can then prove honest verifier zero-knowledge by showing that the multiset of transcripts produced by the honest runs is equal to the multiset of transcripts produced by the simulator. Haines et al. [34] had an imperfect definition of honest verifier zero-knowledge inside Coq which required an additional property to ensure the probabilities are equal. We have updated this definition to remove the problem and it is now sufficient without any additional external properties. (We also proved that their results hold under the updated definition.)

III. Contribution

We will now describe our contributions in their logical order. The first three contributions are necessary for the main contributions but are less significant.

- We extend the work of Haines et al. [34] on the Terelius-Wikström mix net to allow an unbounded number of ciphertexts.
- We formalise a generic class of encryption schemes (Definition 5) which captures most/all of the encryption schemes commonly used in e-voting. Defining this class required some care beyond distilling the salient properties because we wanted the resulting proof of shuffle to be consistent with existing implementations.
- We provide a pen and paper proof showing that the optimized Terelius-Wikström mix net is a proof of correct shuffle; we prove this for all encryption schemes in the generic class and hence most encryption schemes commonly used in e-voting. The proof can found in Appendix B. As we have already noted, something akin to this result was widely believed to be true but no precise characterisation of the claim, much less a proof, has ever been published.
- We encode the above class of encryption schemes into Coq and machine-check the proofs of security for the optimized Terelius-Wikström mix net in Coq. This means we have machine checked both the design and implementation of the mix net to be complete, sound, and zero-knowledge.[9] The Coq formalisations can be found in Sections V and VI. The Coq source is provided in the link in Appendix A
- It then suffices to prove that a given encryption scheme falls into this class in order to get a formally verified cryptographically verifiable mix-net for that encryption scheme. We prove, in Coq, that both basic ElGamal and parallel ElGamal fall into this class (Section VII-A). We have also proved (inside Coq) that anything in this class is preserved under composition, both for the same encryption scheme in parallel and different encryption schemes in parallel. The results can be found in section V-A.
- We demonstrate the practicality and applicability of our formally verified implementation of the optimized Terelius-Wikström mix net by showing that it is able to check (verify) proof transcripts produced by two existing

[9]For completeness and zero-knowledge we prove sufficient conditions [59] but formalising these properties (for protocols with more than three rounds) in Coq is left as future work.

systems used (or planned for use in the case of CHVote) in national elections. The first is the Verificatum [65] mix net which is and has been used in Norway, Estonia, and Switzerland. The second is the CHVote 2.0 [58] electronic voting system developed for the State of Geneva in Switzerland.

A. Clarifications and Limitations

What we verified: We have defined a functor from an encryption scheme into a proof of shuffle for that encryption scheme. We proved all the proofs of shuffle (mix nets) produced by this functor to be complete, to enjoy special soundness and perfect honest-verifier zero-knowledge, provided the encryption scheme satisfies certain properties. Our proofs are either that the property holds perfectly (completeness, and zero-knowledge) or in the form of reductions which reduce attacks against the mix net to some underlying hard problem. Of course, if the parameters are chosen incorrectly, and this underlying problem is not hard in practice, there is no security. For our CHVote verifier, we instantiated the system with a Schnorr group[10] modulo a prime p of 2048bits. We also proved that both ElGamal and Parallel ElGamal have the required properties for the functor to apply. The privacy of the mix net (as a whole) rather than just the zero-knowledge proof additionally depends on the security of the encryption scheme against chosen plaintext attack (IND-CPA) see E; we did not prove that the encryption schemes satisfy IND-CPA.

Side channel attacks: The primary piece of verified software that we wish to use is the verifier. The verifier is a public algorithm running only on public data and hence is incapable of revealing private information; for this reason, we did not verify our implementations to be free of side channel attacks (specifically timing attacks). The prover in the proof of shuffle does run on private information and hence side channel attacks are relevant here. The nature of how these proofs are used in practice for national election, (on air gapped machines), and the nature of the batch proofs themselves make timing attacks more difficult to carry out and, to our knowledge, no timing attack (practical or otherwise) has ever been proposed for a verifiable mix net used in electronic voting. If, for some reason, timing attacks were of greater concern, then it should be easily possible to prove that the algorithms have constant runtime when using a constant time mathematics library since they contain no branching.

Fiat-Shamir transform: We proved the interactive variants of the proof of shuffle; however, in practice, the non-interactive variant is invariably used. The non-interactive variant is obtained from the interactive variant by using the Fiat-Shamir transform [25]. The transform involves replacing the challenge from the verifier with the hash of the statement and commitment. All the verifiable mix nets proved in this work are in the interactive variant; they can be made non-interactive with a one-line change to the verifier but proving the security of the Fiat-Shamir transform inside Coq is left as future work. [11]

Code extraction: We machine-checked the proof of shuffle in Coq but, for efficiency reasons, we extract the Coq code into OCaml code before using it. This process, while fairly mature, could introduce errors, The commonalties between Coq and OCaml mean that the code seems almost identical to human eyes.[12] Alas, we cannot do better as a formally verified extraction facility for Coq is still under development. We stress that this in no way detracts from the value of formally proving the correctness of the specification of the mix net at a level which is computable; it does suggest that the current practice of developing multiple independent verifiers still has value.

IV. Building Blocks

We first present our notation and then the building blocks in Coq upon which will sit the results in the next two sections.

A. Notation

Let M denote a square matrix of order N from $\mathbb{Z}_q^{N*N}$, let $\bar{m}_i$ denote the ith row of M. Let $\mathbf{v}$ be a vector of length N from $\mathbb{Z}_q^N$. Let $\langle \mathbf{v}, \mathbf{v}' \rangle = \sum_{i=1}^{N} \mathbf{v}_i \mathbf{v}'_i$ denote the inner product. Given a finite set S, we write $s \leftarrow_r S$ for a uniformly random assignment of an element in S to the variable s. Given two sets X and Y, a binary relation $\mathcal{R}(X)(Y)$ is a subset of the Cartesian product $X \times Y$. Then, given two binary relations $\mathcal{R}_1$ and $\mathcal{R}_2$, we write $\mathcal{R}_1 \vee \mathcal{R}_2$ for the pairs $((x_1, x_2), w)$ s.t. $(x_1, w) \in \mathcal{R}_1$ or $(x_2, w) \in \mathcal{R}_2$ and write $\mathcal{R}_1 \wedge \mathcal{R}_2$ for the pairs $((x_1, x_2), w)$ s.t. $(x_1, w) \in \mathcal{R}_1$ and $(x_2, w) \in \mathcal{R}_2$.

B. Algebraic Structures

The basic algebraic structures are all standard and encoded into Coq as module types. For instance, a ring is:

```
Module Type RingSig.
  Parameter F        : Set.
  Parameter Fadd     : F -> F -> F.
  Parameter Fzero    : F.
  Parameter Fbool_eq : F-> F-> bool.
  Parameter Fsub     : F -> F -> F.
  Parameter Finv     : F -> F.
  Parameter Fmul     : F -> F -> F.
  Parameter Fone     : F.

  Axiom module_ring : ring_theory Fzero Fone Fadd Fmul
         Fsub Finv (@eq F).

  Axiom F_bool_eq_corr: forall a b : F,
        Fbool_eq a b = true <-> a=b.
  Axiom  F_bool_neq_corr: forall a b : F,
        Fbool_eq a b = false <-> a <> b.

  Add Ring module_ring : module_ring.
End RingSig.
```

[10] A Schnorr group is a prime order subgroup of the integers modulo a larger prime.

[11] As already noted, proving the security of the transform requires reasoning about rewinding in the random oracle model which to the authors' knowledge is not supported in Coq, EasyCrypt or other equivalent interactive provers.

[12] The extracted OCaml code has a slightly less rich type system.

When the module type is instantiated, the parameters must be supplied and the instantiator must prove that the axioms hold. This formalisation is useful since it allows us to build generic structures without specifying which particular instantiation of the module will be used. For instance, the Coq module which encodes what it means to be an algebraic module can take any group and ring which satisfy the axioms.

```
Module Type ModuleSig (Group: GroupSig) (Ring: RingSig).
  Import Group. Export Group.
  Import Ring.  Export Ring.

  Parameter op : G -> F -> G.

  Axiom mod_dist_Gdot: forall (r: F) (x y: G),
        op (Gdot x y) r = Gdot (op x r) (op y r).
  Axiom mod_dist_Fadd: forall (r s: F) (x: G),
        op x (Fadd r s) = Gdot (op x r) (op x s).
  Axiom mod_dist_Fmul: forall (r s: F) (x: G),
        op x (Fmul r s) = op (op x s) r.
  Axiom mod_id: forall (x: G), op x Fone = x.
  Axiom mod_ann: forall (x: G), op x Fzero = Gone.

  Infix "+" := Fadd.            Infix "*" := Fmul.
  Notation "0" := Fzero.        Notation "1" := Fone.
  Notation "- x" :=  (Finv x).
  Notation "x - y" := (x + (- y)).
  Infix "o" := Gdot (at level 50).
  Notation "- x" := (Ginv x).
  Infix "^" := op.
End ModuleSig.
```

We also introduce a number of strange but convenient structures which capture certain combinations of algebraic structures which occur often in the kind of encryption schemes used in e-voting. For instance it is common to set the space of challenges in the proof of shuffle to be the same as the randomness space in the encryption scheme. In the basic case of standard ElGamal, as normally used in a Terelius-Wikström mix net, this is indeed what happens in all implementations. However, when one considers shuffling vectors of ElGamal ciphertexts in parallel, things become more complicated. It would, in theory, still be possible to set the randomness space of these vectors of ElGamal ciphertexts as the challenge space but since it is no longer a field, this creates technical issues. It is much simpler to set the challenge space to the original field and let the field act as a scalar operation on the vectors of ElGamal ciphertexts.

When generalising the properties discussed above slightly further, one ends up with the notation of a group which is a module with respect to a given ring and a given vectorspace with respect to a field. For the proof of the mix net to still work, we need to be able to describe the scalar action of the field on the ring and apply some constraints. This is done in the Coq module below. We have included the details here for completeness but the reader may wish to simply believe us that the proof of the mix net goes through if these properties are satisfied, and further believe that these properties are satisfied in all/most encryption schemes used in e-voting—we have proved both assertions in Coq.

```
Module Type VectorSpaceModuleSameGroup
 (Group: GroupSig) (Ring: RingSig) (Field: FieldSig)
 (M:ModuleSig Group Ring)(VS:VectorSpaceSig Group Field).

 Export Group.  Export M.  Export VS.

 Parameter op3 : Ring.F -> Field.F -> Ring.F.

 Axiom RopInv: forall a, op3 a (Field.Finv Fone) =
       Ring.Finv a.
 Axiom RopFZero: forall x, op3 x Fzero = Ring.Fzero.
 Axiom RopRZero: forall x, op3 Ring.Fzero x = Ring.Fzero.
 Axiom RopDistRadd: forall x y z, op3 (Ring.Fadd x y) z =
       Ring.Fadd (op3 x z) (op3 y z).
 Axiom RopDistFadd: forall (r s : F) (x : Ring.F),
       op3 x (Fadd r s) = Ring.Fadd (op3 x r) (op3 x s).
 Axiom RopDistFmul: forall x y z, op3 (op3 x y) z =
       op3 x (Fmul y z).
 Axiom RaddInv: forall (a : Ring.F)(b : F),
       (Ring.Fadd (op3 a b) (op3 a (Finv b))) =
              Ring.Fzero.

End VectorSpaceModuleSameGroup.
```

C. Pedersen commitments

A Pedersen commitment [53] is an information-theoretic hiding and computationally binding commitment scheme. We will make use of both basic and extended Pedersen commitments; extended Pedersen commitments commit to a vector of messages rather than a single message.

The extended Pedersen commitment scheme Π is the triple of PPT algorithms $(\Pi.\mathsf{Setup}, \Pi.\mathsf{Com}, \Pi.\mathsf{Open})$, such that:

- $CK \leftarrow \Pi.\mathsf{Setup}(\mathbb{G}, n)$ s.t. $N > 1, CK = \{\mathbb{G}, g, h_1, ..., h_N\}$. Given a group $\mathbb{G}$ of prime order q, let g be any generator of $\mathbb{G}$ and choose $h_1, ..., h_N \leftarrow_r \mathbb{G}$.
- The $\Pi.\mathsf{Com}_{CK}$ algorithm takes $\mathbf{m} = (m_1, ..., m_N) \in \mathbb{Z}_q^N, r \in \mathbb{Z}_q$ and sets $c = g^r h_1^{m_1} \dots h_N^{m_N}$ and $d = (\mathbf{m}, r)$.
- The $\Pi.\mathsf{Open}_{CK}$ algorithm takes a commitment $\mathbf{c} \in \mathbb{G}$ and opening $d = (\mathbf{m}, r) \in \mathbb{Z}_q^N \times \mathbb{Z}_q$. If $c = g^r h_1^{m_1} ... h_N^{m_N}$ return $\mathbf{m}$ else return $\perp$.

Commitment schemes which can commit to arbitrarily many values in a constant size commitment, such as extended Pedersen commitments, are used in numerous proofs of correct shuffle to reduce the size of the proof from quadratic to linear; they are inherently information-theoretic hiding and computationally binding.

D. Sigma Protocols

Zero-knowledge proofs are protocols that allow a prover to prove to a verifier that a given statement s belongs to a certain language L. The prover normally uses a witness w to allow it to efficiently compute its part of the protocol. We use $\mathcal{R}$ for the relationship of statements and witness which denotes that the witness w evidences that the statement s belongs to the language L.

Sigma protocols are a class of particularly simple and efficient zero-knowledge proofs that were first defined and analysed by Cramer in his PhD Thesis [18]. Haines et al. [34] provided the logical machinery in Coq to produce provable secure implementations of the sigma protocols commonly used in e-voting; which is to say, prove they satisfy special soundness, honest-verifier zero-knowledge and completeness.

The definition of sigma protocols makes no reference to probabilities in its definition of soundness. Special soundness says that if any adversary can produce two accepting transcripts for different challenges then it is possible to

extract a witness w from those transcripts efficiently such that $(s, w) \in \mathcal{R}$. Bellare and Goldreich give the standard definition of proofs of knowledge in their work "On Defining Proofs of Knowledge" [9]. They define knowledge error, which intuitively denotes the probability that the verifier accepts even when the prover does not know a witness. It has been shown [20] that a sigma protocol satisfying special soundness is a proof of knowledge with negligible knowledge error in the length of the challenge, as stated next.

Theorem 3. *Let $\mathcal{P}$ be a sigma protocol for relation $\mathcal{R}$ with challenge length t. Then $\mathcal{P}$ is a proof of knowledge with knowledge error 2^{-t}.*

We briefly give the formal definition of a sigma protocol here for completeness:

Definition 3. *Sigma Protocol: A protocol $\mathcal{P}$ is said to be a sigma protocol for relation $\mathcal{R}$ if:*

Form: *$\mathcal{P}$ is of the appropriate 3-move form, that is the prover P sends a message, then the verifier V sends a challenge, P sends a reply, and finally V decides to accept on rejected based on the statement and the three messages.*

Completeness: *If P and V follow the protocol on a statement x and private input w where $(x, w) \in \mathcal{R}$, the verifier always accepts.*

Special soundness: *For any statement x and any pair of accepting conversations on x, $(a, e, z), (a, e', z')$ where $e \neq e'$, one can efficiently compute w such that $(x, w) \in \mathcal{R}$.*

Honest-verifier zero-knowledge: *There exists a polynomial-time simulator M, which on statement x and random e outputs an accepting conversation of the form (a, e, z), with the same probability distribution as conversations between the honest P and V on input x.*

We ignore most of the details of how this is transcribed into Coq, for details see [34]. We will, however, give the definition of honest-verifier zero-knowledge since we have updated this property. We stress again that their results hold under the updated definition and we have included the updated Coq proofs in our code repository. The idea of the definition is to show that the multiset of transcripts produced by the honest runs and the simulator are equivalent by describing the first multiset as a function of the honest parties on the set of randomness (`Sig.R`) and the second multiset as a function of the simulator on the set of responses (`Sig.T`); it then suffices to show a bijection (`Sig.simMap`) between the set of randomness and set of responses such that the output of the respective functions are equal. The old definition (below) clearly captures that the output of the respective functions are equal. but the second condition in the definition is only sufficient if the set of response and the set of randomness have the same cardinality. Note that it also conditioned on the relationship (`Sig.Rel`) being true where the definition should be conditioned on the transcript accepting.

```
Class SigmaProtocol (Sig: Sigma.form E) := {
  ...
  honest_verifier_ZK:
    forall (s: Sig.S) (w: Sig.W) (r: Sig.R) (e: E),
      Sig.Rel s w = true ->
      (Sig.P1(Sig.V0 (Sig.P0 s r w) e) r w) =
      Sig.simulator s (Sig.simMap s r e w) e  /\
      forall (t: Sig.T),
       exists r: Sig.R, t = (Sig.simMap s r e w);
        ...
```

In the new definite (below) we fix both issues. We make use of the result that function f from X to Y is bijective iff there exists a function g such that forall x in X, g(f(x)) = x and forall y in Y, f(g(y)) = y. Recall that our aim is show that the function `Sig.simMap` from `Sig.R` to `Sig.T`[13] is bijective and we do this by introducing its inverse `Sig.simMapInv` and requiring the aforementioned properties hold.

```
Class SigmaProtocol (Sig : Sigma.form E) := {
  ...
 honest_verifier_ZK :
  forall (s : Sig.S)(w : Sig.W)(e : E)(r : Sig.R)(t : Sig.T),
    (Sig.V1 (Sig.P1(Sig.V0 (Sig.P0 s r w) e) r w) = true ->
    (Sig.P1(Sig.V0 (Sig.P0 s r w) e) r w) =
    Sig.simulator s (Sig.simMap s w e r) e) /\
    Sig.simMapInv s w e (Sig.simMap s w e r) = r /\
    Sig.simMap s w e (Sig.simMapInv s w e t) = t;
      ...
}.
```

Haines et al. [34] also made the observation that the flow of the proof for the Terelius-Wikström mix net [59], [63], is particularly amenable to machine-checking once sigma protocols are well handled. The structure of the proof is in two stages. First, we prove that the accepting transcripts allow us to extract witnesses satisfying some sub-statements; this follows nearly immediately from the special soundness of the underlying sigma protocol. Then we prove that, given witnesses to these sub-statements, we can produce a witness for the correctness of the shuffle. The definition of soundness used for the Terelius-Wikström mix net is a generalisation of special soundness for sigma protocols; as with special soundness it make no reference to probabilities. In Section VI we comment upon the soundness knowledge implicit in the definition.

The limitation of Haines et al.'s work on sigma protocols is that it only provides a formal proof for the interactive variant; in practice, however, the non-interactive variant is what is used. To make an interactive sigma protocol non-interactive the most common approach is to use the Fiat-Shamir transform [25]. They claim that "*While the Fiat-Shamir transform is out of scope, our explicit formalisation for sigma protocols makes clear what information needs to go into the transform. If the transform is instantiated using the full transcript up to the point of the challenge in our scheme then these issues are avoided.*" Our work on mix nets inherits this limitation since it is based on an underlying sigma protocol.

V. Generic Class of Encryption Schemes

It is relatively clear to anyone familiar with the proof of the Terelius-Wikström mix net that it will work for a variety of cryptosystems, though exactly which cryptosystems it will

[13]Formally `Sig.simMap` is a family of such functions paramertised on s, w, and e.

work for is not always clear. Furthermore, for the optimised variant of the mix net which is used in practice, no one has ever provided—to our knowledge—a concrete classification of the cryptosystems for which it works. We provide the first such classification which, while not exhaustive, does capture most/all of the encryption schemes used with the mix net in practice. The only suggested encryption schemes, of which the authors are aware, for the Terelius-Wikström mix net which are not covered by our generic class are variants of the Paillier encryption scheme [51]. It would seem most of the variants of Paillier can be included in a generalisation of our class which requires only that the overwhelming majority of the elements in (what was previously a) field are invertible.

Our description is fairly verbose; we will first give it in somewhat standard cryptographic notation and then give the Coq version. The reader will note that we don't describe any privacy property of the encryption scheme but only algebraic properties. No privacy properties of the scheme are required for the privacy of the proof of shuffle; however, the scheme should satisfy indistinguishably under chosen plaintext attack (IND-CPA) to provide privacy to the overall mix net.

Definition 4. *A (Terelius-Wikström compatible) encryption scheme Σ is a tuple of PPT algorithms* $(\Sigma.\mathsf{KeyGen}, \Sigma.\mathsf{Enc}, \Sigma.\mathsf{Dec}, \Sigma.\mathsf{KeyMatch})$*, such that:*

- *the ciphertext space $\mathcal{C}$ is a group under some operation, the randomness space R is a ring (usually the integers modulo n, where n is either prime or semi-prime) and the ciphertext space is a module with respect to the randomness space;*
- *the message space $\mathcal{M}$ forms an Abelian group;*
- *the* KeyGen *algorithm defines a set of public and secret key pairs $(\mathcal{PK}, \mathcal{SK})$ from which one is uniformly selected: $(PK \in \mathcal{PK}, SK \in \mathcal{SK}) \leftarrow_r \Sigma.\mathsf{KeyGen}()$;*
- *The* Enc *algorithm takes a public key PK, message m and randomness r and returns a ciphertext CT from $\mathcal{C}$: that is, $\forall PK \in \mathcal{PK}, \forall m \in \mathcal{M}, \forall r \in \mathcal{R},\ CT \in \mathcal{C} \leftarrow \Sigma.\mathsf{Enc}_{PK}(m, r)$*
- *The* Dec *algorithm takes a ciphertext $CT \in \mathcal{C}$ and $SK \in \mathcal{SK}$ and returns either a message $m \in \mathcal{M}$ or null ($\perp$): that is, $\forall CT \in \mathcal{C}, \Sigma.\mathsf{Dec}_{SK}(c) \rightarrow m \in \mathcal{M}$ or $m = \perp$;*
- *Correctness: $\forall PK, SK$, if $(PK \in \mathcal{PK}, SK \in \mathcal{SK}) \leftarrow_r \Sigma.\mathsf{KeyGen}()$, then $\forall m \in \mathcal{M}, r \in R, \Sigma.\mathsf{Dec}_{SK}(\Sigma.\mathsf{Enc}_{PK}(m, r)) = m$;*
- *Homomorphic: $\forall PK \in \mathcal{PK}, \forall m, m' \in \mathcal{M}, \forall r, r' \in \mathcal{R}$, $\Sigma.\mathsf{Enc}_{PK}(m', r') * \Sigma.\mathsf{Enc}_{PK}(m, r) = \Sigma.\mathsf{Enc}_{PK}(m \cdot m', r + r')$;*[14]
- *there exists a field which forms a vector space with the ciphertext space. Further the field and ring satisfy, for some operator, the axioms in VectorSpaceModuleSameGroup.*

The Coq definition captures the cryptographic definition with several additional technicalities which we will skip over. The module that captures the encryption schemes takes as input a group, ring, and field. In addition it requires a proof that the group and ring form a module and the group and field form a vectorspace.

```
Module Type EncryptionScheme (Group: GroupSig)
  (Ring: RingSig) (Field: FieldSig) (M: ModuleSig Group
    Ring)  (VS: VectorSpaceSig Group Field)
  (MVS: VectorSpaceModuleSameGroup Group Ring Field M VS)
   <: Mixable Message Ciphertext Ring Field VS MVS.
Import MVS.
  Parameter KGR      : Type. (* randomness for keygen *)
  Parameter PK       : Type.   (* public key space *)
  Parameter SK       : Type.   (* secret key space *)
  Parameter M        : Set.      (* message space *)
  Parameter Mop      : M -> M -> M.
  Parameter Mzero    : M.
  Parameter Minv     : M -> M.
  Parameter Mbool_eq : M -> M -> bool.

  Parameter keygen  : KGR -> (PK*SK).
  Parameter enc     : PK -> M -> Ring.F -> G.
  Parameter dec     : SK -> G -> M.
  Parameter keymatch: PK -> SK -> bool.

  Axiom correct: forall (kgr: KGR) (m: M) (r: Ring.F),
        let (pk, sk) := keygen kgr in
        dec sk (enc pk m r) = m.

  Axiom M_abgrp: AbeGroup M Mop Mzero Mbool_eq Minv.

  Axiom homomorphism: forall (pk : PK) (m m' : M)
                (r r' : Ring.F),
        Group.Gdot (enc pk m' r') (enc pk m r) =
            enc pk (Mop m m') (Ring.Fadd r r').

  Axiom encOfOnePrec: forall (pk : PK)(a : Ring.F)(b :F),
         VS.op (enc pk zero a) b = enc pk zero (op3 a b).

End EncryptionScheme.
```

Having defined the encryption scheme, it is fairly straightforward to define both the computational and decisional variants of re-encryption, where the re-encryption is evidenced by knowledge of the randomness used to re-encrypt.

```
 Definition reenc (pk: PK) (c: G) (r: Ring.F): G :=
   Group.Gdot (enc pk Mzero r) c.

 Definition IsReEnc (pk : PK)(c1 c2 : G)(r : F) : Prop :=
   c2 = reenc pk c1 r.
```

A. Class preserved under Composition

Before we discuss the composability of encryption schemes, we will first take an aside. Ciphertexts are not the only thing which one may want to mix securely; it is also common to mix commitments. To deal with this, we define a module called Mixable which suffices for mixing but has less structure than an encryption scheme. This module capture commitments and other less structured objects which we may wish to be mixed. The definition of the encryption scheme enforces that all encryption schemes are Mixable but the inverse is not necessarily true.

```
Module Type Mixable (Message Ciphertext : GroupSig)
  (Ring: RingSig)(Field : FieldSig)(VS : VectorSpaceSig
  Ciphertext Field)(MVS :  VectorSpaceModuleSameGroup
  Ciphertext Ring Field VS).
  Import MVS.
  Parameter KGR : Type.
  Parameter PK : Type.

  Definition M := Message.G.
  Definition Mop := Message.Gdot.
  Definition Mzero := Message.Gone.
  Definition Minv := Message.Ginv.
```

[14] The exact operations represented by $*, \cdot$ and $+$ are fairly flexible.

```
  Definition Mbool_eq := Message.Gbool_eq.

  Parameter keygenMix : KGR -> PK. (* key generation   *)
  Parameter enc : PK -> M -> Ring.F -> G. (* or commit *)

  Axiom M_abgrp : AbeGroup M Mop Mzero Mbool_eq Minv.

  Axiom homomorphism : forall (pk : PK)(m m' : M)
        (r r' : Ring.F),
                Ciphertext.Gdot (enc pk m' r')(enc pk m r) =
                  enc pk (Mop m m') (Ring.Fadd r r').

  Axiom encOfOnePrec : forall (pk : PK)(a : Ring.F)(b :F),
          VS.op (enc pk Mzero a) b = enc pk Mzero (op3 a b).
End Mixable.
```

We have proved in Coq that given any Mixable, and any natural number n, that if you do n lots of the mixable pairwise, that is you take the product groups of all the spaces, the result is still within the class mixable. This is important because often in practice in e-voting, and other applications, the senders submit more than one ciphertext which needs to be shuffled in parallel. That is, the input for m senders is n ciphertexts each, these are shuffled so that the output is m packets of n ciphertexts, the mix net hides which sender corresponds to which output packet but the verifiability ensures that not only are the underlying messages the same but that the ciphertext groupings are preserved. We have also proved that given any two Mixables if you do all operation pairwise this is still a Mixable. Again, this is important because in practice it is common to mix different schemes together; everlasting privacy e-voting schemes in particular mix ciphertexts and commitments together. Both of these results are not particularly surprising, though the exact characterisation might be, but a machine checked proof of them is very useful in increasing the applicability of the work since it allows more complicated "mixables", and hence mix nets to be automatically generated from simpler mixables. The exact description of how this works in Coq is included in Appendix D.

VI. Optimized Terelius-Wikström mix net

Recall that the homomorphic properties of the encryption systems in question ensure that if each of the mixers perform a shuffle then the output is guaranteed to be a shuffle; it then suffices for each mix server to prove that it behaved correctly. A proof of shuffle provides exactly that functionality.

The optimized Terelius-Wikström mix net is a relatively straightforward, if verbose, zero-knowledge proof for the correctness of a shuffle of the initial input. More properly, it could be called a zero-knowledge argument for the correctness of a shuffle since an adversary is able to efficiently make proofs even if the shuffle is not correct, provided that the adversary can break the binding property of the commitment scheme used. Of course, the commitment scheme should be chosen such that the binding property is hard and then the proof works as expected.

The (common) mistake of making the binding property weak and hence breaking the mix net has occurred in the SwissPost system for national elections in Switzerland and many less significant systems. It is easier to prove that the commitments are binding in Coq and we have done so. However, it is more challenging to prove that the commitment parameters are generated correctly; we comment on the feasibility of this in the future work section (IX-A).

A. Intuition

The Terelius-Wikström mix net is the combination of Wikström's work [63] and Terelius-Wikström's work [59]. In the earlier work, Wikström shows how to split the effort of constructing a proof of shuffle into an offline and online phase—for simplicity of presentation, we have chosen in this work to fold the two phases back together. In the offline phase, the prover first commits to a permutation matrix and then proves that it did so; in the online phase, the prover then shuffles the ciphertexts and proves that it did so consistently with the matrix committed to in the offline phase. In the latter work, the authors showed that a square matrix matrix $M = (m_{i,j})$ over $\mathbb{Z}_q$ is a permutation matrix if and only if for $\bar{x} = (x_1, ..., x_N)$ a vector of indepedendent variables:

$$\prod_{i=1}^{N} \langle \bar{m}_i, \bar{x} \rangle = \prod_{i=1}^{N} x_i \text{ and } M\bar{1} = \bar{1}.$$

This result is important because if the matrix is committed to column-wise in extended Pedersen commitments—$(c_1, ..., c_N) = (g^{r_1} \prod_{i=1}^{N} h_i^{m_{i,1}}, ..., g^{r_N} \prod_{i=1}^{N} h_i^{m_{i,N}})$—it is easy to publicly compute the commitment to the inner product $\langle \bar{m}_i, \bar{x} \rangle$ as $\prod_{j=1}^{N} c_j^{x_j}$. The first part of the necessary and sufficient condition given above can be efficiently tested by Schwarz-Zippel's lemma, which states that if $f(x_1, ..., x_N)$ is a non-zero polynomial of degree d and we pick a point $\bar{e}$ from $\mathbb{Z}_q^N$ randomly, then the probability that $f(\bar{e}) = 0$ is at most d/q.

The result of these insights is that is suffices to check certain discrete log relationships to establish a proof of correctness. It is then fairly straight-forward to construct a sigma protocol which proves these discrete log problems relationships hold.

B. Formally

We will now formally define the relationship which is proved, starting by defining the subrelationships.

We define $\mathcal{R}_{com}$ to be the relation consisting of pairs of tuples; the first tuple is of the form commitment key CK, commitment c. The second tuple is of the form, two distinct message vectors $\mathbf{m}, \mathbf{m}'$ and two associated randomness values r and r' s.t. $c = \Pi.\mathsf{Com}_{CK}(\mathbf{m}, r) = \Pi.\mathsf{Com}_{CK}(\mathbf{m}', r')$. If an adversary can find a witness to this relationship they have broken the binding property of the commitment scheme which in this case also means they have broken the discrete log problem. We encode this definition in Coq as follows where EPC denotes the extended Pedersen commit.

```
Definition relComEPC (h: G) (hs: VG (1+N)) (c: G)
             (m1 m2: VF (1+N))(r1 r2 : F) :=
  m1 <> m2 /\
  c = (EPC (1+N) h hs m1 r1) /\
      c = (EPC (1+N) h hs m2 r2).
```

We define $\mathcal{R}_{\pi}$ to be the relation consisting of pairs of tuples of the form commitment key CK, vector of commitments $\mathbf{c}$,

message M and associated randomness vector $\mathbf{r}$ s.t. M is a permutation matrix and $\mathbf{c}_i = \Pi.\mathsf{Com}_{CK}(M_i, \mathbf{r}_i)$.

```
Definition relPi (h: G) (hs: VG (1+N)) (c: VG (1+N))
  (m: MF (1+N)) (r: VF (1+N)) :=
    MFisPermutation m
     /\ c = (com (1+N) h hs m r).
```

We define $\mathcal{R}^{shuf}_{\Sigma_{PK}}$ to be the relation consisting of pairs of tuples of the form public key PK, two vectors of ciphertexts $\mathbf{CT} = (ct_1, \cdots, ct_n)$ and $\mathbf{CT}' = (ct'_1, \cdots, ct'_n)$ and a permutation π and randomness vector $\mathbf{r} = (r_1, \cdots, r_n)$ such that $ct'_i = ct_{\pi(i)}\Sigma.Enc_{PK}(1, r_{\pi(i)})$ for all $i \in [1, N]$. We define this in Coq by first calculating $ct_{\pi(i)}$ and $r_{\pi(i)}$ as `e''` and `r''`, respectively, and then requiring that the re-encryption relationship holds for all ciphertexts in the vectors.

```
(* Definition of shuffling *) (*e2_i = e1_p_i * r_p_i*)
Definition relReEnc(pk: enc.PK) (e e': vector G1.G (1+N))
       (m : MoC.MF (1+N)) (r: MoC_M.VF (1+N)) :=
  let e'' := MoC.PexpMatrix e' m in
  let r'' := RF_CVmult m r in
  let partial := Vmap2 (fun e e' => IsReEnc pk e e')
      e e'' in
  let partial2 := Vmap2 (fun x y => x y) partial r'' in
    Vforall (fun x => x ) partial2.
```

We are now at last ready to define the relationship proved by the mix net. The relationship is $\mathcal{R}_{com} \vee (\mathcal{R}_{\pi} \wedge \mathcal{R}^{shuf}_{\Sigma_{PK}})$ which says that either the ciphertexts were correctly mixed or the adversary has broken the commitment scheme. This is formalised as follows,

```
Definition WikRel (pk: enc.PK) (e e': vector G1.G (1+N))
                  (h: G) (hs: VG (1+N)) (c: VG (1+N)) :=
 (exists (r: MoC_M.VF (1+N)) (r': VF (1+N)) (m: MF (1+N)),
     (relReEnc pk e e' m r /\ relPi h hs c m r'))
    \/  ((exists (c: G) (m1 m2: VF (1+N)) (r1 r2: F),
          relComEPC h hs c m1 m2 r1 r2).
```

The protocol that the prover and verifier follow for the proof of shuffle is shown in Algorithm 1; normally, the proof is made non-interactive using the Fiat-Shamir transform. So in practice, we assume that the Fiat-Shamir transform was correctly implemented. For convenience of presentation, we present Algorithm 1 for the case of basic ElGamal where the randomness and challenge space is $\mathbb{Z}_q$. The algorithm describes a 4 round zero-knowledge proof between a prover and verifier which asserts that the prover knows private inputs which satisfy the public inputs. In the first round the verifier challenges the prover to which the prover responds with several commitments in round two. In the third round the verifier challenges the prover again and prover respond in round four. In step five of the algorithm the verifier checks if the responses provided by the prover in round four satisfy certain relationships with the statement and commitments.

The standard paper proof adopts a form of special soundness as a means to prove soundness. See Wikström's recent work [64] for information on the knowledge error achieved by this proof.

Our Coq theorem for the security (special soundness) of the mix net is given below. Essentially it universally quantifies over all possible statements and proof transcripts (lines 1 to 21) and then says if enough of the proof transcripts accept

Algorithm 1: Terelius-Wikström proof of shuffle

Common Input: Commitment parameters $g, h, h_1, ..., h_N \in \mathbb{G}$, two vectors of ciphertexts $\mathbf{e} = (e_1, ..., e_N) \in \mathcal{C}$ and $\mathbf{e}' = (e'_1, ..., e'_N) \in \mathcal{C}$, and a permutation matrix commitment $\mathbf{c} = (c_1, ..., c_N)$.

Private Input : Permutation matrix $M = (m_{i,j}) \in \mathbb{Z}_q^{N \times N}$, randomness $\mathbf{r} = (r_1, ..., r_N) \in \mathbb{Z}_q^N$ s.t. $c_j = g^{r_j} \prod_{i=1}^N h_i^{m_{i,j}}$, and randomness $\mathbf{r}' = (r'_1, ..., r'_N) \in \mathcal{R}$ s.t. $e'_i = e_{\pi(i)}\Sigma.\mathsf{Enc}_{PK}(1, r'_{\pi(i)})$, for $i, j \in [1, N]$.

1 $\mathcal{V}$ chooses $\mathbf{u} = (u_1, ..., u_N) \in \mathbb{Z}_q^N$ randomly and hands $\mathbf{u}$ to $\mathcal{P}$.

2 $\mathcal{P}$ defines $\mathbf{u}' = (u'_1, ..., u'_N) = M\mathbf{u}$ and then chooses $\hat{\mathbf{r}} = (\hat{r}_1, ..., \hat{r}_N), \hat{\mathbf{w}} = (\hat{w}_1, ..., \hat{w}_N), \mathbf{w}' = (w'_1, ..., w'_N) \in \mathbb{Z}_q^N$, and $w_1, w_2, w_3, \in \mathbb{Z}_q$ and $w_4 \in \mathcal{R}$. $\mathcal{P}$ defines $\bar{r} = \langle \bar{1}, \mathbf{r} \rangle$, $\tilde{r} = \langle \mathbf{r}, \mathbf{u} \rangle$, $\hat{r} = \sum_{i=1}^N \hat{r}_i \prod_{j=i+1}^N u'_j$ and $r' = (\sum_{i=1}^N r'_{i,0} u_i, \prod_{i=1}^N r'^{u_i}_{i,1}, \prod_{i=1}^N r'^{u_i}_{i,2})$. $\mathcal{P}$ hands to $\mathcal{V}$, where we set $\hat{c}_0 = h$ and $i \in [1, N]$,

$$\hat{c}_i = g^{\hat{r}_i} \hat{c}_{i-1}^{u'_i} \quad t_1 = g^{w_1} \quad t_2 = g^{w_2} \quad t_3 = g^{w_3} \prod_{i=1}^N h_i^{w'_i}$$

$$t_4 = \Sigma.\mathsf{Enc}_{PK}(0, w_4) \prod_{i=1}^N e_i'^{w'_i} \qquad \hat{t}_i = g^{\hat{w}_i} \hat{c}_{i-1}^{w'_i}$$

3 $\mathcal{V}$ chooses a challenge $\xi \in \mathbb{Z}_q$ at random and sends it to $\mathcal{P}$.

4 $\mathcal{P}$ then responds with:

$$s_1 = w_1 + \xi \cdot \bar{r} \qquad s_2 = w_2 + \xi \cdot \hat{r} \qquad s_3 = w_3 + \xi \cdot \tilde{r}$$
$$\hat{s}_i = \hat{w}_i + \xi \cdot \hat{r}_i \qquad s'_i = w'_i + \xi \cdot u'_i \qquad s_4 = w_4 - \xi \cdot r'$$

5 $\mathcal{V}$ accepts if and only if, for $i \in [1, N]$,

$$t_1 = (\textstyle\prod_{i=1}^N c_i / \prod_{i=1}^N h_i)^{-\xi} g^{s_1} \quad t_2 = (\hat{c}_N / h^{\prod_{i=1}^N u_i})^{-\xi} g^{s_2}$$

$$t_3 = (\textstyle\prod_{i=1}^N c_i^{u_i})^{-\xi} g^{s_3} \prod_{i=1}^N h_i^{s'_i} \qquad \hat{t}_i = \hat{c}_i^{-\xi} g^{\hat{s}_i} \hat{c}_{i-1}^{s'_i}$$

$$t_4 = (\prod_{i=1}^N (e_i)^{u_i})^{-\xi} \Sigma.\mathsf{Enc}_{PK}(0, s_4) \prod_{i=1}^N (e'_i)^{s'_i}$$

(lines 23 to 28) with distinct challenges (lines 30 and 31) then we can extract a witness to the relationship (line 49). The requirement that the matrix of challenges has an inverse (line 40 and 41) could be removed by slightly modifying the proof but this would render the result incompatible with the deployed implementations.

```
1  Theorem TheMixnetIsSecure :
2   (* For all statements *)
3   (* For all keys and ciphertexts *)
4   forall (pk: enc.PK) (e e': (vector G1.G (1+N))),
5   (* Commitment parameters and matrix commitments *)
6   forall (h: G) (hs: VG (1+N)) (c: VG (1+N)),
7   (* For primary challenges *)
```

```
8   forall(U: MF (1+N)),
9   forall (cHat: vector (VG (1+N)) (1+N)),
10  let Sig := WikstromSigma in
11  let s := Vmap2 (fun cHat col =>
12    WikstromStatment pk h hs c cHat col e e')
13          cHat U  in
14  (* Sigma protocols accept *)
15  forall (com: vector (Sigma.C F Sig) (1+N))
16       (chal: vector (F*F) (1+N))
17       (t: vector((Sigma.T F Sig)*(Sigma.T F Sig))(1+N)),
18
19  let transcript := Vmap2 ( fun x y => (x,y))
20         (Vmap2 (fun x y => (x,y))
21         (Vmap2 ( fun x y => (x,y)) s com) chal) t in
22
23  Vforall
24   (fun t => Sigma.V1 F Sig (t.1.1.1,t.1.1.2,t.1.2.1,
25    t.2.1) = true
26    /\ Sigma.V1 F Sig (t.1.1.1,t.1.1.2,t.1.2.2,t.2.2)
27           = true)
28   transcript ->
29
30  Vforall (fun e => Sigma.disjoint F Sig e.1 e.2 = true)
31   chal ->
32
33  let w := Vmap (fun t => Sigma.extractor F Sig t.2.1
34   t.2.2 t.1.2.1 t.1.2.2) transcript in
35  let U' := Vmap (fun w => w.2.1.1.1) w in
36
37  let A := MF_inv U in
38  let B := (MF_mult A U') in
39
40  MF_mult U' (MF_inv U') = (MF_id (1+N)) ->
41  MF_mult U (MF_inv U) = (MF_id (1+N)) ->
42
43  (MFisPermutation B = false ->
44     (VF_beq (MF_VCmult (VF_one (1+N)) B) (VF_one (1+N))
45      && Fbool_eq (VF_prod (MF_VCmult (Vnth U index0) B)
46      - VF_prod (Vnth U index0)) 0)
47     = false) ->
48
49   WikRel pk e e' h hs c.
```

Vforall is a function which takes a predicate and a vector and returns true if and only if the predicate holds true for all elements of the vector.

The proof of the theorem can be found in the Coq source, essentially it works by exploiting the structure of the proof of shuffle which is essentially a sigma protocol with an extra challenge on the front. For each initial challenge, we first extract the witness whose existence is guaranteed by the underlying sigma protocol. Having gathered all these witnesses, we compute from them, using fairly straightforward linear algebra, either two distinct openings to the same commitment or the permutation and randomness used to shuffle. The only part of the proof which is not linear algebra is the use of the Schwartz-Zippel lemma to check the equality of polynomials. Since the lemma implies the polynomials are equal except with negligible probability we have added this as an assumption to the theorem (lines 43 to 47) and leave a general treatment of the lemma as future work.

VII. Encryption schemes in class

Having now proven the security of the mix net for all cryptosystems in the class defined in Section V, we are now in a position to prove that the cryptosystems commonly used in e-voting fall into this class.

A. ElGamal in Class

The most common encryption scheme used in e-voting is ElGamal. (In addition to the many variants of ElGamal the PPATC scheme from [19] also belongs to this class.) We briefly recall the definition of ElGamal here.

Definition 5. *ElGamal encryption scheme (in a Schnorr group)* Σ *is a tuple* $(\Sigma.\mathsf{KeyGen}, \Sigma.\mathsf{Enc}, \Sigma.\mathsf{Dec}, \Sigma.\mathsf{KeyMatch})$ *of PPT algorithms such that:*

- *Let* $\mathbb{G}$ *be the group of* k*th residues in* $\mathbb{Z}_p$ *of prime order* q *where* $p = kq + 1$ *for some* p, k*, and* q*. Let* g *denote some generator of* $\mathbb{G}$*.*
- *the ciphertext space* $\mathcal{C}$ *is the cartesian product of* $\mathbb{G}$ *with itself, the randomness space* R *is the field* $\mathbb{Z}_q$ *and the ciphertext space is a vector space with respect to the randomness space;*
- *the message space* $\mathcal{M}$ *is* $\mathbb{G}$*;*
- *the* KeyGen *algorithm defines a set of public and secret key pairs* $(\mathcal{PK} = \mathbb{G}, \mathcal{SK} = \mathbb{Z}_q)$ *from which one is uniformly selected:* $(PK \in \mathcal{PK}, SK \in \mathcal{SK}) \leftarrow_r \Sigma.\mathsf{KeyGen}() = (g^x, x \leftarrow_r \mathbb{Z}_q)$*;*
- *The* Enc *algorithm takes a public key* PK*, message* M *and* R *and returns a ciphertext* CT *from* $\mathcal{C}$*: that is,* $\forall PK \in \mathcal{PK}, \forall m \in \mathcal{M}, \forall r \in \mathcal{R},\ CT \in \mathcal{C} \leftarrow \Sigma.\mathsf{Enc}_{PK}(m, r) = (g^r, PK^r m)$
- *The* Dec *algorithm takes a ciphertext* $CT = (c_1, c_2) \in \mathcal{C}$ *and* $SK \in \mathcal{SK}$ *and returns either a message* $m \in \mathcal{M}$ *or null* $\perp$*: that is,* $\forall CT \in \mathcal{C}, \Sigma.\mathsf{Dec}_{SK}(CT) \rightarrow c_2/(c_1^{SK})$.[15]

Basic ElGamal fits the definition of our (Terelius-Wikström compatible) encryption scheme. The statement of this theorem, see below, is fairly straightforward but because of the work we have already done it immediately allows us to get a verified implementation of the optimised Terelius-Wikström mix net which is compatible with existing voting systems used in national elections; this is a first and hugely significant in gaining better confidence in the correctness of e-voting systems deployed in national elections.

DualGroupSig for any given group gives a group whose base set is the Cartesian product of the initial group's and where operations are performed pairwise. DualVectorSpaceSig is a vector space of a dual group with a field which holds automatically provided the original group was a vectorspace with the same field. See Appendix C for more details.

```
Module BasicElGamal (Group: GroupSig) (Field: FieldSig)
  (VS: VectorSpaceSig Group Field)
  (DualGroup: DualGroupSig Group)
  (DVS: DualVectorSpaceSig Group DualGroup Field VS)
  (MVS: VectorSpaceModuleSameGroupIns DualGroup Field DVS)
    <: EncryptionScheme DualGroup Field Field DVS DVS MVS.
  Module AddVSLemmas := VectorSpaceAddationalLemmas Group
        Field VS.
  Import AddVSLemmas.
  Module AddDVSLemmas :=
    VectorSpaceAddationalLemmas DualGroup Field DVS.

  Import MVS.
  Import Field.

  Definition KGR      := prod Group.G F.
  Definition PK       := DualGroup.G.
  Definition SK       := F.
  Definition M        := Group.G.
```

[15] For ElGamal—provided the input is in the right set—always returns a message and never $\perp$

```
  Definition Mop      := Group.Gdot.
  Definition Mzero    := Group.Gone.
  Definition Minv     := Group.Ginv.
  Definition Mbool_eq := Group.Gbool_eq.

  Definition keygen (kgr: KGR): (PK*SK) :=
      ((kgr.1,VS.op kgr.1 kgr.2),kgr.2).

  Definition enc (pk: PK) (m: Group.G) (r: F) : G :=
      (VS.op pk.1 r, Group.Gdot (VS.op pk.2 r) m).

  Definition dec (sk: F) (C: G): M :=
      Group.Gdot C.2 (Group.Ginv (VS.op C.1 sk)).

  Definition keymatch (pk: PK) (sk: SK): bool :=
      Group.Gbool_eq (VS.op pk.1 sk) pk.2.

  Lemma M_abgrp : AbeGroup M Mop Mzero Mbool_eq Minv.
  (* We have redacted the proofs *)

  Lemma correct : forall (kgr : KGR)(m : M)(r : F),
     let (pk,sk) := keygen kgr in
                 dec sk (enc pk m r) = m.

  Lemma homomorphism : forall (pk : PK)(m m':M)(r r':F),
     Gdot (enc pk m' r')(enc pk m r) =
                   enc pk (Mop m m') (Fadd r r').

  Lemma encOfOnePrec : forall (pk : PK)(a b : F),
     (DVS.op (enc pk Mzero a) b) =enc pk Mzero (Fmul a b).

End BasicElGamal.
```

However, as we noted earlier, most of the national e-voting systems do not use basic ElGamal. Instead they use a variety of variants of ElGamal. For instance parallel ElGamal where multiple ElGamal ciphertexts are shuffled in parallel is fairly common, as is Gjøsteen ElGamal which achieves short ciphertexts for longer messages in exchange for longer keys. All of these variants are (provably Terelius-Wikström compatible) encryption schemes.

We have proven in Coq that parallel ElGamal is a (Terelius-Wikström compatible) encryption scheme. The module which shows this takes as input the number of ciphertexts N to be shuffled in parallel. It then constructs the basic ElGamal ciphertext space as DVS which it then expands into the parallel ElGamal ciphertext space as NthGroup. NthRing is the randomness space. There is a bit more detail in the definition but essentially it reduces to doing everything on the underlying basic ElGamal pairwise on each ciphertext.

```
Module Type Nat.
  Parameter N : nat.
End Nat.

Module ExtendedElGamal (Hack: Nat) (Group : GroupSig)
  (Field: FieldSig) (VS: VectorSpaceSig Group Field)
  (DualGroup: DualGroupSig Group)
  (DVS: DualVectorSpaceSig Group DualGroup Field VS)
  (NthGroup: GroupNthSig DualGroup Hack)
  (NthRing: NthRingSig Hack Field)
  (NthM: NthModuleSig Hack DualGroup Field NthGroup
        NthRing DVS)
  (NthVS: NthVectorSpaceSig Hack DualGroup Field NthGroup
        DVS)
  (MVS: VectorSpaceModuleSameGroupNthSig Hack DualGroup
        Field DVS NthGroup NthRing NthM NthVS)
    <: EncryptionScheme NthGroup NthRing Field NthM NthVS
        MVS.
  Import Hack.
  Module MoM := Matrices Group Field VS.
  Module AddVSLemmas := VectorSpaceAddationalLemmas Group
        Field VS.
    Import AddVSLemmas.
    Import MVS.
    Import Field.

    (* randomness for keygen *)
    Definition KGR := prod (MoM.VG N)(MoM.VF N).
    Definition PK := NthGroup.G. (* public key space *)
    Definition SK := MoM.VF N.   (* secret key space *)
    Definition M := MoM.VG N.    (* message space    *)
   (* message space is an ablelian group *)
    Definition Mop (a b : M) := MoM.VG_mult a b.
    Definition Mzero := MoM.VG_id N.
    Definition Minv (a : M) := MoM.VG_inv a.
    Definition Mbool_eq (a b : M) := MoM.VG_eq a b.

  Definition keygen (kgr: KGR): (PK*SK) :=
   (Vmap2 (fun x y => (x, VS.op x y)) kgr.1 kgr.2, kgr.2).

  Definition enc (Pk: PK)(m : M) (r: MoM.VF N):
        NthGroup.G :=
    let mr := Vmap2 (fun x y => (x,y)) m r  in
    Vmap2 (fun (pk :DualGroup.G)(mr : (Group.G*F)) =>
    (VS.op pk.1 mr.2, Group.Gdot (VS.op pk.2 mr.2) mr.1))
        Pk mr.

  Definition dec (Sk : SK)(C : NthGroup.G) : M :=
   Vmap2 (fun sk c => Group.Gdot c.2 (Group.Ginv
        (VS.op c.1 sk))) Sk C.

    Definition keymatch (Pk : PK)(Sk : SK) : bool :=
      MoM.VG_eq (Vmap2 (fun pk sk => VS.op pk.1 sk) Pk Sk)
                (Vmap (fun x => x.1) Pk).

    (* We have redacted the proofs *)

End ExtendedElGamal.
```

VIII. Applications to Verifying National Elections

We will now discuss the application of our work to verify national elections. Currently, we have proven that both ElGamal and parallel ElGamal fall into this class. We can now extract the mix net with the case of basic ElGamal into OCaml and use this to verify evidence produced by an election scheme built for binding government elections.

Our Coq formalisation contains a Coq function to mix the inputs to produce the appropriate outputs. These functions are rather trivial and consist of only a few lines of Coq. Our Coq proofs verify that these functions obey their Coq specifications, meaning they produce correctly mixed outputs. Moreover, for each such mixnet (function), our Coq formalisation contains a Coq function for the required verifier, and a Coq proof of its correctness, meaning the software to produce the ZKP proof that the mix was done correctly, the software to check this ZKP proof, and Coq proofs that the verifier accepts on the output of the prover running on valid inputs. That is, we also have formally verified verifiers for these various mix nets. However, as we have already stated we are primarily interested in using our verifier to check existing systems rather than convincing countries and vendors to switch to our implementation of the mix net and prover. Since, the soundness property of the verifier ensures that if it accepts then, with overwhelming probability, the mixing was done correctly; it is unnecessary for the integrity of the mix net to use our verified mixer and prover since it suffices to use our verified verifier instead.

A. Extracted Verifier

In this section will discuss our extracted verifier, specifically its delta from the verified objects, its efficiency and the existing

e-voting system with which we have tested it.

a) Delta between the verified objects and the implementation: There are two gaps between our verified objects and the extracted verifier.

The first is that the Coq extraction facility is not itself verified to be correct; the required verified extraction facility CertiCoq has been under development for some time but until it is complete this gap remains. For existing elections it seems astronomically unlikely that the flaws, if any, in the system line up with the flaws in the Coq extraction facility. However, once the extracted verifier is public, an adversary may try to determine flaws and design a system which exploits them. For this reason, the current practice of multiple independent verifier should be continued. The Coq proof of the verifier (before extraction) is still a machine checked proof that a correct verifier exists for the mix net in question, which is still a massive improvement over the current state of the art.

The second delta is the Fiat-Shamir transform. As we have already noted, since the statement and commitment are both formally defined in our implementation, this is a one line modification to the implementation. Proving the correctness of this transform is out of scope as it requires formalising highly non-trivial cryptographic reasoning; specifically it requires reasoning about rewinding in the random oracle model which to our knowledge has not been done by any prior work in Coq, EasyCrypt or other comparable interactive theorem provers. We have also checked that the verifier in our implementation is compatible with the proof transcripts produced by (other) implementations used in national elections. One must check that the Fiat-Shamir transform is done correctly when checking their transcripts, but as noted, the security of this one line modification is not overly hard to check. When verifying transcripts produced by other implementations, this modification is dependent on that implementation.

b) Efficiency: Efficiency is not as large an issue with mix net verifiers as in some other areas of cryptography since the verifier is run irregularly, as compared to key exchange for example. Our extracted verifier is lightweight with the exception of group exponentiations. However, since our verifier uses a fairly standard implementation of big integers, it is not much slower than a directly written verifier. For example, when using a 2048bit Schnorr group, an election consisting of 10,000 ballots took 200 seconds to verify. The verifiers we tested on both Verificatum and CHVote have nearly identical efficiency. We note that the bottleneck operation is parallelisable and further optimisations, such as fixed base exponentiation, could also be applied. At present, the largest elections using these techniques are in Estonia and our results suggest that our verifier would check all the proofs in under an hour on commodity hardware, even without further optimisations.

c) Sanity Checks: We have also tested our extracted verifier on a wide variety of invalid transcripts; as expected it rejected all of them. This is not a surprising result because, by the soundness property we proved about the verifier, it must reject on the overwhelming majority of inputs.

B. Verificatum

The Verificatum mix net was the first complete and fully distributed verifiable mix net. It is considered the gold standard for implementations of verifiable mix nets and has withstood easily the security of its more than a decade of use in national elections. The mix net has been used in national elections in Norway, Estonia and Switzerland.

We installed Verificatum from the github repository[16] on an Ubuntu virtual machine and generated various test proof transcripts using its demo functionality; this process took some time due to configuration issues. We converted these transcripts to JSON files using the tool provided by Verificatum. There are several differences in notation and structure between Verificatum and our verifier but the parser is not overly complicated. Specifically, we deliberately wrote our verifier to generalise both Verificatum and CHVote and be compatible with either depending on the choice of parameters. We wrote a parser to convert the JSON files into the data structures expected by our verifier; finally, we successfully verified the transcripts from Verificatum using our verifier. This code is included in our repository.

C. CHVote 2.0

The CHVote 2.0 system [58] was developed by the state of Geneva in Switzerland.

Switzerland's elections are run at a canton (state) level. Different cantons use different methods with different vendors. We have already referred to the SwissPost electronic voting system which was used in several cantons. The canton of Geneva decided to develop its own system which was called CHVote2.0. The system is fairly similar to the SwissPost system in user experience. The voter first receives a code sheet in the mail; they then log into the online system and cast their vote. After casting, they receive a confirmation code which should match the sheet they were sent. The system then uses a verifiable mix net as part of the tally process.

The mix net used in the CHVote2.0 system is a fairly direct implementation of the optimized Terelius-Wikström mix net. Importantly, it is compatible with the verifier we proved in Coq. We installed the CHVote2.0 system locally from its git repository[17] and produced several demo election transcripts by slightly extending the test routines included; in this case we copied the transcripts directly from the console output and posted them into the OCaml file which calls the verifier. Again, there are several differences in notation and structure between CHVote2.0 and our verifier but the parser, and parameters, are not overly complicated. Finally, we successfully verified the transcripts from CHVote2.0 in our verifier.

D. Deploying the verifier on real elections

We have not deployed the verifier on transcripts from real elections even though we have checked the verifier is compatible with the software used to generate those transcripts. This

[16] https://github.com/verificatum

[17] https://github.com/republique-et-canton-de-geneve/chvote-protocol-poc

is because the countries using these systems do not make the transcripts publicly available but only release them to trusted auditors.

IX. Conclusion

Verifiable mix nets are a crucial component in numerous applications because they are deployed widely in electronic voting systems for governmental elections where they are by far the most complicated component. Given the history of critical errors in electronic voting, it is necessary to develop tools to give greater confidence.

Our work on using Coq to prove the correctness of the mix net most commonly used in secure e-voting and then using the extracted verifier on evidence produced by an e-voting system for real governmental elections will contribute to this greater confidence.

We have demonstrated our verifier for two e-voting systems which have collectively handled millions of votes and been used in several national elections; the verifier is also clearly applicable to a wide variety of additional electronic voting systems and to other applications that rely upon verifiable mix nets based on proofs of shuffles.

A. Future work

There are several relevant directions to expand either the security guarantees provided by this work or its applicability.

As already mentioned, we inherit the limitation of Haines et al. [34] that we only prove in Coq the interactive version and consequently we must assume that the Fiat-Shamir transform is done correctly when the scheme is made non-interactive; implementing this transform is fairly straight forward, despite how often errors have occurred in implementations. Nevertheless, it would be highly interesting to prove the transform (as used in e-voting) in Coq and hence remove this limitation; at present, a proof of the transform seems impractical but it may be possible to do a hybrid paper/machine proof where we machine-check that our use of the transform works if the transform works.

The Terelius-Wikström mix net only works as a proof of shuffle if the commitment scheme is binding; consequently, it only works if the commitment parameters are correctly generated. Generating these parameters is fairly straightforward and there are standards such as Algorithm A.2.3 Verifiable Canonical Generation of the Generator g from NIST fips186-4 [26]. It would still be interesting to prove that the commitment parameters were generated correctly. However, the task seems trivial under a basic security notation; it is easy to show in Coq that the parameters were mapped into the group from a random seed such that no information about the discrete log is leaked. No better security notation is known to the authors, informally the adversary can choose a polynomial number of discrete log problem instances and its goal is to solve one of these but it is unclear exactly how to formalize everything in Coq without axiomatizing (assuming) over important points. Furthermore, each election system tends to generate these parameters differently, meaning this would need to be done on a case by case basis.

Another interesting area of future work would be to use an interactive theorem prover which provides formal guarantees for the extracted code which reach down to the machine code level, such as CakeML.

References

[1] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[2] Ben Adida and Douglas Wikström. How to Shuffle in Public. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 555–574, 2007.

[3] Ben Adida and Douglas Wikström. Offline/Online Mixing. In *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, pages 484–495, 2007.

[4] José Bacelar Almeida, Endre Bangerter, Manuel Barbosa, Stephan Krenn, Ahmad-Reza Sadeghi, and Thomas Schneider. A certifying compiler for zero-knowledge proofs of knowledge based on sigma-protocols. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, pages 151–167, 2010.

[5] José Bacelar Almeida, Manuel Barbosa, Endre Bangerter, Gilles Barthe, Stephan Krenn, and Santiago Zanella Béguelin. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 488–500, 2012.

[6] 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.

[7] Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin, Benjamin Grégoire, and Sylvain Heraud. A machine-checked formalization of sigma-protocols. In *CSF*, pages 246–260. IEEE Computer Society, 2010.

[8] Stephanie Bayer and Jens Groth. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.

[9] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 1992.

[10] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting Helios for provable ballot privacy. In Vijay Atluri and Claudia Díaz, editors, *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, volume 6879 of *Lecture Notes in Computer Science*, pages 335–354. Springer, 2011.

[11] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[12] Yves Bertot, Pierre Castéran, Gérard Huet, and Christine Paulin-Mohring. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Texts in theoretical computer science. Springer, 2004.

[13] Alessandro Bruni, Eva Drewsen, and Carsten Schürmann. Towards a mechanized proof of Selene receipt-freeness and vote-privacy. In *E-VOTE-ID*, volume 10615 of *Lecture Notes in Computer Science*, pages 110–126. Springer, 2017.

[14] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1614–1625, 2016.

[15] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[16] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. Machine-checked proofs for electronic voting: Privacy and verifiability for Belenios. In *CSF*, pages 298–312. IEEE Computer Society, 2018.

[17] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

[18] Ronald Cramer. Modular design of secure yet practical cryptographic protocols. 1997.

[19] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election Verifiability or Ballot Privacy: Do We Need to Choose? In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 481–498, 2013.

[20] Ivan Damgård. On Σ-protocols. 2002.

[21] Stéphanie Delaune, Mark Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied Pi calculus. In *IFIPTM*, volume 263 of *IFIP Advances in Information and Communication Technology*, pages 263–278. Springer, 2008.

[22] Prastudy Fauzi and Helger Lipmaa. Efficient Culpably Sound NIZK Shuffle Argument Without Random Oracles. In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, pages 200–216, 2016.

[23] Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. An Efficient Pairing-Based Shuffle Argument. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, pages 97–127, 2017.

[24] Prastudy Fauzi, Helger Lipmaa, and Michal Zajac. A Shuffle Argument Secure in the Generic Model. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, pages 841–872, 2016.

[25] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[26] PUB FIPS. 186-4: Federal information processing standards publication. Digital Signature Standard (DSS). *Information Technology Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD*, pages 20899–8900, 2013.

[27] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 1997.

[28] Jun Furukawa and Kazue Sako. An Efficient Scheme for Proving a Shuffle. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2001.

[29] Pierrick Gaudry. Breaking the encryption scheme of the Moscow internet voting system. *CoRR*, abs/1908.05127, 2019.

[30] Kristian Gjøsteen. The Norwegian internet voting protocol. In *VOTE-ID*, volume 7187 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[31] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.

[32] R Haenni. Swiss post public intrusion test: undetectable attack against vote integrity and secrecy (2019), 2019.

[33] Thomas Haines. A description and proof of a generalised and optimised variant of Wikström's mixnet. *CoRR*, abs/1901.08371, 2019.

[34] Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 685–702. ACM, 2019.

[35] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In Alina Oprea and Hovav Shacham, editors, *2020 IEEE Symposium on Security and Privacy, SP 2020, San Jose, CA, USA, May 17-21, 2020*, pages 784–800. IEEE, 2020.

[36] Thomas Haines and Johannes Mueller. Sok: Techniques for verifiable mix nets. Cryptology ePrint Archive, Report 2020/490, 2020. https://eprint.iacr.org/2020/490.

[37] J Alex Halderman and Vanessa Teague. The New South Wales ivote system: Security failures and verification flaws in a live online election. In *International Conference on E-Voting and Identity*, pages 35–53. Springer, 2015.

[38] Chloé Hébant, Duong Hieu Phan, and David Pointcheval. Linearly-Homomorphic Signatures and Scalable Mix-Nets. *IACR Cryptology ePrint Archive*, 2019:547, 2019.

[39] Sven Heiberg and Jan Willemson. Verifiable internet voting in Estonia. In *EVOTE*, pages 1–8. IEEE, 2014.

[40] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, pages 339–353, 2002.

[41] Wojciech Jamroga, Michal Knapik, and Damian Kurpiewski. Model checking the SELENE e-voting protocol in multi-agent logics. In *E-Vote-ID*, volume 11143 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2018.

[42] Shahram Khazaei, Tal Moran, and Douglas Wikström. A Mix-Net from Any CCA2 Secure Cryptosystem. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 607–625. Springer, 2012.

[43] Shahram Khazaei, Björn Terelius, and Douglas Wikström. Cryptanalysis of a universally verifiable efficient re-encryption mixnet. In *EVT/WOTE*. USENIX Association, 2012.

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

[45] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 341–354, 2016.

[46] Ralf Küsters and Tomasz 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.

[47] Ralf Küsters, Tomasz Truderung, and Andreas 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.

[48] Helger Lipmaa and Bingsheng Zhang. A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. In *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*, pages 477–502, 2012.

[49] C. Andrew Neff. A Verifiable Secret Shuffle and its Application to E-Voting. In Michael K. Reiter and Pierangela Samarati, editors, *8th ACM Conference on Computer and Communications Security (CCS 2001)*, pages 116–125. ACM, 2001.

[50] Lan Nguyen, Reihaneh Safavi-Naini, and Kaoru Kurosawa. Verifiable shuffles: A formal model and a Paillier-based efficient construction with provable security. In *ACNS*, volume 3089 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2004.

[51] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology EUROCRYPT 99*, pages 223–238. Springer, 1999.

[52] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*, 1993.

[53] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[54] Ronald L Rivest. On the notion of 'software independence' in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3759–3767, 2008.

[55] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C, 2nd Edition*. Wiley, 1996.

[56] Ben Smyth, Mark Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In *ARSPA-WITS*, volume 6186 of *Lecture Notes in Computer Science*, pages 146–163. Springer, 2010.
[57] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J Alex 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.
[58] The state of Geneva. CHvote. https://github.com/republique-et-canton-de-geneve/chvote-protocol-poc, 2018.
[59] Björn Terelius and Douglas Wikström. Proofs of Restricted Shuffles. In Daniel J. Bernstein and Tanja Lange, editors, *Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa*, volume 6055 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2010.
[60] Douglas 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.
[61] Douglas Wikström. A Universally Composable Mix-Net. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 317–335. Springer, 2004.
[62] Douglas Wikström. A Sender Verifiable Mix-Net and a New Proof of a Shuffle. In *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, pages 273–292, 2005.
[63] Douglas Wikström. A Commitment-Consistent Proof of a Shuffle. In *Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Brisbane, Australia, July 1-3, 2009, Proceedings*, pages 407–421, 2009.
[64] Douglas Wikström. Special Soundness Revisited. *IACR Cryptology ePrint Archive*, 2018:1157, 2018.
[65] Douglas Wikström. Verificatum. https://github.com/verificatum/verificatum-vcr, 2018.
[66] Douglas Wikström and Jens Groth. An Adaptively Secure Mix-Net Without Erasures. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, pages 276–287, 2006.

APPENDIX A
COQ SOURCE CODE

We have made our code available at https://github.com/gerlion/secure-e-voting-with-coq.

APPENDIX B
PAPER PROOF OF THE OPTIMIZED TERELIUS-WIKSTRÖM MIX NET FOR THIS CLASS OF ENCRYPTION SCHEMES

In this section we give the paper proof of security of the optimized Terelius-Wikström mix net for any (Terelius-Wikström compatible) encryption scheme. We note that the proof is very similar in many respects to the existing proofs of the optimized Terelius-Wikström mix net for ElGamal [33]. Indeed, one way to understand the generalisation we have contributed is to think of what we have done as distilling the salient properties of ElGamal for the proof to work and defining our class of encryption schemes to be the class of encryptions schemes which has these properties.

Correctness follows immediately from the definitions and from the homomorphic properties of the encryption schemes, which we have proved in Coq, but we omit the tedious paper proof.

The proof, as we give it here[18], is simplified slightly to the case where the challenge space is $\mathbb{Z}_q$ rather than an arbitrary field. However, all algebraic arguments that follow hold in general under the requirement that this field forms a vectorspace with both the commitment space and the ciphertext space. It is possible to relax the definition to allow the challenge space to be a ring where the overwhelming majority of elements have inverses (think the integers modulo a semi-prime); this is because the inverses are only used on the challenges in the extractor, since these elements are chosen uniformly at random by the honest verifier with overwhelming probability inverses will exist.

[18]The Coq proof is for the general form

A. Zero-Knowledge

In the Coq formulation, the zero-knowledge property of the mix net follows immediately from the zero-knowledge property of the underlying sigma protocol. A simulator follows step one honestly and also generates $\hat{c}$ honestly and then uses the underlying sigma protocol simulator to simulate steps two through four. The paper proof is as follows:

Proof. The special zero-knowledge simulator chooses $\hat{c}_1, ..., \hat{c}_N, \in \mathbb{Z}_q$, $\mathbf{c} \in \mathbb{Z}_q^N$ $\hat{\mathbf{s}}, \mathbf{s}', \mathbf{u} \in \mathbb{Z}_q^N$, and $s_1, s_2, s_3, c \in \mathbb{Z}_q$ and $s_4 = \mathcal{R}$ randomly and defines $t_1, t_2, t_3, t_4, \hat{t}_i$ by the equations in step five. This is a perfect simulation.

To show that the simulated and real transcripts have the same statistical distribution we compare their terms as follows:

- $\mathbf{u} \in_R \mathbb{Z}_q^N$ in both.
- $\hat{c}_1, ..., \hat{c}_N \in_R \mathbb{G}_q$ in simulated and as $\hat{c}_i = g^{\hat{r}_i} \hat{c}_{i-1}^{u'_i}$ in the real transcript where $\hat{r}_i \in_R \mathbb{Z}_q$ which randomly distributes them in $\mathbb{G}_q$.
- $t_1, t_2, t_3, t_4, \hat{\mathbf{t}}$ and the corresponding $s_1, s_2, s_3, s_4, \hat{\mathbf{s}}, \mathbf{s}$ have a defined relation which depends on secrets and ws. Since the ws are randomly defined in an honest run and the $s_1, s_2, s_3, s_4, \hat{\mathbf{s}}, \mathbf{s}$ in the simulated, the elements are uniformly distributed in both, up to the defined relationship.
- The challenge c is uniformly distributed in both.

□

B. Soundness

Proof. The extractor from two accepting transcripts

$$(g, h, h_1, ..., h_N, \mathbf{e}, \mathbf{e'}, \mathbf{c}, \mathbf{u}, \hat{\mathbf{c}}, t_1, t_2, t_3, t_4, \hat{\mathbf{t}}, \xi, s_1, s_2, s_3, s_4, \hat{\mathbf{s}}, \mathbf{s}')$$

$$(g, h, h_1, ..., h_N, \mathbf{e}, \mathbf{e'}, \mathbf{c}, \mathbf{u}, \hat{\mathbf{c}}, t_1, t_2, t_3, t_4, \hat{\mathbf{t}}, \xi^*, s_1^*, s_2^*, s_3^*, s_4^*, \hat{\mathbf{s}}^*, \mathbf{s}'^*)$$

with $\xi - \xi^* \in \mathbb{Z}_n^*$ computes $\bar{r} = (s_1 - s_1^*)/(\xi - \xi^*), \hat{r} = (s_2 - s_2^*)/(\xi - \xi^*), \tilde{r} = (s_3 - s_3^*)/(\xi - \xi^*), r' = (s_4 - s_4^*)/(\xi - \xi^*), \hat{\mathbf{r}}' = ((\hat{\mathbf{s}}' - \hat{\mathbf{s}}'^*)/(\xi - \xi^*), \mathbf{u}' = (\mathbf{s}' - \mathbf{s}'^*)/(\xi - \xi^*)$.

We will now show that these extracted values satisfying the following relationships:

$$\prod_{j=1}^N \mathbf{c}_j = g^{\bar{r}} \prod_{i=1}^N h_i \qquad \prod_{j=1}^N \mathbf{c}_j^{u_j} = g^{\tilde{r}} \prod_{i=1}^N h_i^{\mathbf{u}'_i}$$

$$\prod_{i=1}^N (\mathbf{e}'_i)^{\mathbf{u}'_i} = \Sigma.\mathsf{Enc}_{PK}(0, r') \cdot \prod_{j=1}^N \mathbf{e}_j^{\mathbf{u}_j}$$

$$\hat{\mathbf{c}}_i = g^{\hat{\mathbf{r}}_i}\hat{\mathbf{c}}_{i-1}^{\mathbf{u}'_i} \qquad \hat{\mathbf{c}}_N = g^{\hat{r}}h^{\prod_{i=1}^N \mathbf{u}_i}$$

The proof consists of simple algebraic transformations. For neatness of presentation we will denote steps which involve only algebraic manipulation as (by alg.) and those which involve the verification definition as (By ver. def.):

$$\left(\frac{(\prod_{j=1}^N \mathbf{c}_j)^{\xi} t_1}{(\prod_{j=1}^N \mathbf{c}_j)^{\xi^*} t_1}\right)^{\frac{1}{\xi-\xi^*}} = \prod_{j=1}^N \mathbf{c}_j \qquad \text{Tautology}$$

$$\left(\frac{g^{s_1}/(\prod_{i=1}^N h_i)^{-\xi}}{g^{s_1^*}/(\prod_{i=1}^N h_i)^{-\xi^*}}\right)^{\frac{1}{\xi-\xi^*}} = \prod_{j=1}^N \mathbf{c}_j \qquad \text{By ver. def.}$$

$$g^{\frac{s_1-s_1^*}{\xi-\xi^*}} \prod_{i=1}^N h_i = \prod_{j=1}^N \mathbf{c}_j \qquad \text{By alg.}$$

$$g^{\bar{r}} \prod_{i=1}^N h_i = \prod_{j=1}^N \mathbf{c}_j \qquad \text{By def. of } \bar{r}$$

$$\left(\frac{(\prod_{j=1}^N \mathbf{c}_j^{\mathbf{u}_j})^{\xi} t_3}{(\prod_{j=1}^N \mathbf{c}_j^{\mathbf{u}_j})^{\xi^*} t_3}\right)^{\frac{1}{\xi-\xi^*}} = \prod_{j=1}^N \mathbf{c}_j^{\mathbf{u}_j} \qquad \text{Tautology}$$

$$\left(\frac{g^{s_3}\prod_{i=1}^N h_i^{\mathbf{s}'_i}}{g^{s_3^*}\prod_{i=1}^N h_i^{\mathbf{s}'^*_i}}\right)^{\frac{1}{\xi-\xi^*}} = \prod_{j=1}^N \mathbf{c}_j^{\mathbf{u}_j} \qquad \text{By ver. def.}$$

$$h^{\frac{s_3-s_3^*}{\xi-\xi^*}} \prod_{i=1}^N h_i^{\frac{\mathbf{s}'_i-\mathbf{s}'^*_i}{\xi-\xi^*}} = \prod_{j=1}^N \mathbf{c}_j^{\mathbf{u}_j} \qquad \text{By alg.}$$

$$g^{\tilde{r}} \prod_{i=1}^N h_i^{\boldsymbol{u}'} = \prod_{j=1}^N \mathbf{c}_j^{\mathbf{u}_j} \qquad \text{By def. of } \boldsymbol{u}' \text{ and } \tilde{r}$$

$$\left(\frac{(\prod_{i=1}^N (\mathbf{e}_i)^{\mathbf{u}_i})^{\xi} \mathbf{t}_4}{(\prod_{i=1}^N (\mathbf{e}_i)^{\mathbf{u}_i})^{\xi^*} \mathbf{t}_4}\right)^{\frac{1}{\xi-\xi^*}} = \prod_{i=1}^N \mathbf{e}_i{}^{\mathbf{u}_i} \qquad \text{Tautology}$$

$$\left(\frac{\prod_{i=1}^N (\mathbf{e}'_i)^{\mathbf{s}'_i}\mathsf{Enc}_{PK}(0,\mathbf{s}_4)}{\prod_{i=1}^N (\mathbf{e}'_i)^{\mathbf{s}'^*_i}\mathsf{Enc}_{PK}(0,\mathbf{s}_4^*)}\right)^{\frac{1}{\xi-\xi^*}} = \prod_{i=1}^N \mathbf{e}_i{}^{\mathbf{u}_i} \qquad \text{By ver. def.}$$

$$\prod_{i=1}^N \mathbf{e}_i{}'^{\frac{\mathbf{s}'_i-\mathbf{s}'^*_i}{\xi-\xi^*}} \Sigma.\mathsf{Enc}_{PK}(0, \frac{\mathbf{s}_4^* - \mathbf{s}_4}{\xi-\xi^*}) = \prod_{i=1}^N \mathbf{e}_i{}^{\mathbf{u}_i} \qquad \text{By alg.}$$

$$\Sigma.\mathsf{Enc}_{PK}(0, \frac{\mathbf{s}_4 - \mathbf{s}_4^*}{\xi-\xi^*}) \prod_{i=1}^N \mathbf{e}_i{}^{\mathbf{u}_i} = \prod_{i=1}^N \mathbf{e}_i{}'^{\frac{\mathbf{s}'_i-\mathbf{s}'^*_i}{\xi-\xi^*}} \qquad \text{By alg.}$$

$$\Sigma.\mathsf{Enc}_{PK}(0, r') \prod_{i=1}^N \mathbf{e}_i{}^{\mathbf{u}_i} = \prod_{i=1}^N \mathbf{e}_i{}'^{\mathbf{u}'_i} \qquad \text{By def.}$$

Now, for each $i \in \{1, \ldots, N\}$

$$\left(\frac{\hat{\mathbf{c}}_i^{\xi}\hat{\mathbf{t}}_i}{\hat{\mathbf{c}}_i^{\xi^*}\hat{\mathbf{t}}_i}\right)^{\frac{1}{\xi-\xi^*}} = \hat{\mathbf{c}}_i \qquad \text{Tautology}$$

$$\left(\frac{h^{\hat{\mathbf{s}}_i}\hat{\mathbf{c}}_{i-1}^{\mathbf{s}'_i}}{h^{\hat{\mathbf{s}}_i^*}\hat{\mathbf{c}}_{i-1}^{\mathbf{s}'^*_i}}\right)^{\frac{1}{\xi-\xi^*}} = \hat{\mathbf{c}}_i \qquad \text{By ver. def.}$$

$$h^{\frac{\hat{\mathbf{s}}_i-\hat{\mathbf{s}}_i^*}{\xi-\xi^*}} \hat{\mathbf{c}}_{i-1}^{\frac{\mathbf{s}'_i-\mathbf{s}'^*_i}{\xi-\xi^*}} = \hat{\mathbf{c}}_i \qquad \text{By alg.}$$

$$g^{\hat{\mathbf{r}}_i}\hat{\mathbf{c}}_{i-1}^{\mathbf{u}'_i} = \hat{\mathbf{c}}_i \qquad \text{By def. of } \mathbf{u}'_i \text{ and } \hat{\mathbf{r}}_i$$

$$\left(\frac{\hat{\mathbf{c}}_N^{\xi} t_2}{\hat{\mathbf{c}}_N^{\xi^*} t_2}\right)^{\frac{1}{\xi-\xi^*}} = \hat{\mathbf{c}}_N \qquad \text{Tautology}$$

$$\left(\frac{(h_1^{\prod_{i=1}^N \mathbf{u}_i})^{\xi} g^{s_2}}{(h_1^{\prod_{i=1}^N \mathbf{u}_i})^{\xi^*} g^{s_2^*}}\right)^{\frac{1}{\xi-\xi^*}} = \hat{\mathbf{c}}_N \qquad \text{By ver. def.}$$

$$g^{\frac{s_2-s_2^*}{\xi-\xi^*}} h_1^{\prod_{i=1}^N \mathbf{u}_i} = \hat{\mathbf{c}}_N \qquad \text{By alg.}$$

$$g^{\hat{r}} h^{\prod_{i=1}^N \mathbf{u}_i} = \hat{\mathbf{c}}_N \qquad \text{By def. of } \hat{r}$$

a) Extended Extractor: We now sketch the extended extractor which, for a given statement (see the common input in Algorithm 2), for n different witnesses extracted by the basic extractor, produces the witnesses to the main statement. Let the collective output of the basic extractors be denoted as $\bar{\mathbf{r}}, \hat{\mathbf{r}}, \tilde{\mathbf{r}}, \mathbf{r}^\star \in \mathbb{Z}_q^n$, and $\hat{R}, U' \in \mathbb{Z}_q^{N\times N}$ extracted from the primary challenges $U \in \mathbb{Z}_q^{N\times N}$. We denote by U_i the ith column of U which is the challenge vector from the ith run of the basic extractor, and by $U_{j,i}$ the j element of the challenge vector from the ith run of the basic extractor.

First note with overwhelming probability the set of U_is is linearly independent, concretely the probability is bounded by $\frac{q-2}{q}$. From linear independence, it follows that there exists $A \in \mathbb{Z}_q^{N\times N}$ such that UA_l is the lth standard unit vector in $\mathbb{Z}_q$ which we will denote by $\mathbb{I}_l$. A is the inverse of U. We will denote by $EPC(\mathbf{m}, r)$ the extended Pedersen commitment $g^r \prod_{i=1}^N h_i^{\mathbf{m}_i}$. Clearly,

$$\mathbf{c}_l = \prod_{i=1}^{N} (\mathbf{c}^{UA_l})_i \qquad \text{since } UA_l \text{ is } \mathbb{I}_l$$

$$\mathbf{c}_l = \prod_{i=1}^{N} \mathbf{c}_i^{\sum_{j=1}^{N} U_{i,j} A_{j,l}} \qquad \text{by definition of } UA_l$$

$$\mathbf{c}_l = \prod_{i=1}^{N} \left(\prod_{j=1}^{N} \mathbf{c}^{U_{i,j} A_{j,l}} \right)_i \qquad \text{by alg.}$$

$$\mathbf{c}_l = \prod_{j=1}^{N} \left(\left(\prod_{i=1}^{N} \mathbf{c}_i^{U_{i,j}} \right)^{A_{j,l}} \right) \qquad \text{by alg.}$$

$$\mathbf{c}_l = \prod_{j=1}^{N} EPC(U'_j, \tilde{\mathbf{r}}_j)^{A_{j,l}} \qquad \text{by } \prod_{i=1}^{N} \mathbf{c}_i^{U_{i,j}} = EPC(U'_j, \tilde{\mathbf{r}}_j)$$

$$\mathbf{c}_l = \prod_{j=1}^{N} EPC(U'_j A_{j,l}, \tilde{\mathbf{r}}_j A_{j,l}) \qquad \text{by alg.}$$

$$\mathbf{c}_l = EPC(\sum_{j=1}^{N} U'_j A_{j,l}, \langle \tilde{\mathbf{r}}, A_l \rangle) \qquad \text{by alg.}$$

$$\mathbf{c}_l = EPC(U' A_l, \langle \tilde{\mathbf{r}}, A_l \rangle) \qquad \text{by alg.}$$

Therefore, we can open $\mathbf{c}$ to the matrix M, where the lth column of M is $U'A_l$, with randomness $\langle \tilde{\mathbf{r}}, A_l \rangle$. In other words we open $\mathbf{c} = U'A$ using randomness $\tilde{\mathbf{r}}A$.

We expect M to be a permutation matrix, but if it is not, then one can find a witness to $\mathcal{R}_{com}$ (which, as has been mentioned, can only happen with negligible probability, under our security assumptions). We extract in two different ways depending on whether $M\bar{1} \neq \bar{1}$.

b) Option one: If $M\bar{1} \neq \bar{1}$, then let $\mathbf{u}'' = M\bar{1}$ and note that

$$\mathbf{u}'' \neq \bar{1} \text{ and } EPC(\bar{1}, \bar{\mathbf{r}}_j) = \prod_{i=1}^{N} \mathbf{c}_i = \prod_{i=1}^{N} \mathbf{c}_i^{\bar{1}_i} = EPC(\mathbf{u}'', \tilde{\mathbf{r}}A)$$

in which case we found a witness breaking the commitment scheme.

c) Option two: If $M\bar{1} = \bar{1}$, then recall Theorem 1 from "Proofs of Restricted Shuffles", which states that M is a permutation matrix if and only if $M\bar{1} = \bar{1}$ and $\prod_{i=1}^{N} \langle \mathbf{m}_i, \mathbf{x} \rangle - \prod_{i=1}^{N} \mathbf{x}_i = 0$. Since $M\bar{1} = \bar{1}$ and M is not a permutation matrix, then $\prod_{i=1}^{N} \langle \mathbf{m}_i, \mathbf{x} \rangle - \prod_{i=1}^{N} \mathbf{x}_i \neq 0$. The Schwartz–Zippel says that if you sample, a non-zero polynomial, at a random point the chance that it equals zero is negligible in the order of the underlying field; hence, with overwhelming probability there exists $j \in \{1, \ldots, N\}$ such $\prod_{i=1}^{N} \langle \mathbf{m}_i, U_j \rangle - \prod_{i=1}^{N} U_{i,j} \neq 0$. Since this is true with overwhelming probability, we require it to be true and rewind if this is not the case. (Strictly speaking we should take $N+1$ extractions from the basic extractor, if we recover a different M we win, if we get the same M then U_{l+1} is actually independent of M and the lemma can be applied.)

Let $\mathbf{u}'' = MU_j$ and note that

$$\mathbf{u}'' \neq U'_j \qquad \text{Since } \prod_{i=1}^{N} U'_{i,j} = \prod_{i=1}^{N} U_{i,j} \neq \prod_{i=1}^{N} \mathbf{u}''_i$$

$\prod_{i=1}^{N} U'_j = \prod_{i=1}^{N} U_j$ follows from the base statements and $\prod_{i=1}^{N} U_j \neq \prod_{i=1}^{N} \mathbf{u}''$ by definition of $\mathbf{u}''$ and $\prod_{i=1}^{N} \langle \mathbf{m}_i, U_j \rangle - \prod_{i=1}^{N} U_{i,j} \neq 0$.

$$EPC(U'_j, \tilde{\mathbf{r}}_j) = \prod_{i=1}^{N} \mathbf{c}_i^{U_{i,j}} = EPC(\mathbf{u}'', \langle \tilde{\mathbf{r}}A, U_j \rangle)$$

This completes the proof that M is a permutation matrix or we have found a witness to $\mathcal{R}_{com}$.

d) The correctness of U': We now show that $U'_l = MU_l$ for all $l \in [1, N]$ or we can find a witnesses to $\mathcal{R}_{com}$. Let $\mathbf{u}'' = MU_l$ and by assumption $\mathbf{u}'' \neq U'_l$.

$$EPC(U'_l, \tilde{\mathbf{r}}_l) = \prod_{i=1}^{N} \mathbf{c}_i^{U_{i,l}} = EPC(\mathbf{u}'', \langle \tilde{\mathbf{r}}A, U_l \rangle)$$

e) Extracting the randomness: We have shown that if M is not a permutation matrix we can extract a witness to $\mathcal{R}_{com}$. We now show, that if it is a permutation matrix, we can extract $R \in \mathcal{R}_{pk}$ such that $\mathbf{e}'_i = \mathbf{e}_{\pi(i)} \Sigma.\mathsf{Enc}_{PK}(0, R_{\pi(i)})$.

For reasons of space will write $\Sigma.\mathsf{Enc}_{pk}$ as E in the following section.

$$\mathbf{e}_l = \prod_{i=1}^{N} (\mathbf{e}^{UA_l})_i \qquad \text{since } UA_l \text{ is } \mathbb{I}_l$$

$$\mathbf{e}_l = \prod_{i=1}^{N} \mathbf{e}_i^{\sum_{j=1}^{N} U_{i,j} A_{j,l}} \qquad \text{by def. of } UA_l$$

$$\mathbf{e}_l = \prod_{i=1}^{N} \left(\prod_{j=1}^{N} \mathbf{e}^{U_{i,j} A_{j,l}} \right)_i \qquad \text{by alg.}$$

$$\mathbf{e}_l = \prod_{j=1}^{N} \left(\prod_{i=1}^{N} \mathbf{e}^{U_{i,j}} \right)^{A_{j,l}} \qquad \text{by alg.}$$

Recall that the basic extractor showed that $\prod_{i=1}^{N} \mathbf{e}_i^{U_{i,j}} =$

$\prod_{i=1}^{N} \mathbf{e}_i'^{U'_{i,j}} \mathsf{E}(0, -\mathbf{r}_j^\star)$.

$$\mathbf{e}_l = \prod_{j=1}^{N} \left(\prod_{i=1}^{N} \mathbf{e}_i'^{U'_{i,j}} \mathsf{E}(0, -\mathbf{r}_j^*) \right)^{A_{j,l}} \quad \text{by basic.}$$

$$\mathbf{e}_l = \prod_{j=1}^{N} \left(\prod_{i=1}^{N} \mathbf{e}_i'^{U'_{i,j} A_{j,l}} \mathsf{E}(0, -\mathbf{r}_j^* A_{j,l}) \right) \quad \text{by alg.}$$

$$\mathbf{e}_l = \prod_{i=1}^{N} \mathbf{e}_i'^{\sum_{j=1}^{N} U'_{i,j} A_{j,l}} \mathsf{E}(0, -\langle \mathbf{r}^*, A_l \rangle) \quad \text{by alg.}$$

$$\mathbf{e}_l = \prod_{i=1}^{N} (\mathbf{e}'^{U'A_l})_i \mathsf{E}(0, -\langle \mathbf{r}^*, A_l \rangle) \quad \text{by alg.}$$

$$\mathbf{e}_l = \prod_{i=1}^{N} (\mathbf{e}'^{MUA_l})_i \mathsf{E}(0, -\langle \mathbf{r}^*, A_l \rangle) \quad \text{since U' = MU}$$

$$\mathbf{e}_l = \prod_{i=1}^{N} (\mathbf{e}'^{M\mathbb{I}_l})_i \mathsf{E}(0, -\langle \mathbf{r}^*, A_l \rangle) \quad \text{since } UA_l = \mathbb{I}_l$$

$$\mathbf{e}_l = \mathbf{e}'_{\pi_M^{-1}(l)} \mathsf{E}(0, -\langle \mathbf{r}^*, A_l \rangle) \quad \text{by def. of } \pi_M$$

We have now shown that $\mathbf{ReEnc}_{pk}(\mathbf{e}_l, \langle \mathbf{r}_l^*, A_l \rangle) = \mathbf{e}'_{\pi^{-1}(l)}$; hence, $R_l = \langle \mathbf{r}_l^*, A_l \rangle$ which concludes the proof. □

APPENDIX C
ADDITIONAL COQ CODE

We refer the reader to the repository in Appendix A for the complete code. Nevertheless, we give here some additional code which is referred to in the body of the paper for completeness.

A. DualGroupSig

The Coq module DualGroupSig formalises the direct product of a group with itself.

```
Module Type DualGroupSig (Group : GroupSig) <: GroupSig.
  Import Group.

  Definition G := prod G G.
  Definition Gdot (a b : G) : G :=
        (Gdot a.1 b.1, Gdot a.2 b.2).
  Definition Gone := (Gone, Gone).
  Definition Gbool_eq (a b : G) := Gbool_eq a.1 b.1 &&
        Gbool_eq a.2 b.2.
  Definition Ginv a := (Ginv a.1, Ginv a.2).

  Lemma module_abegrp : AbeGroup G Gdot Gone Gbool_eq
        Ginv
End DualGroupSig.
```

B. DualVectorSpaceSig

The Coq module DualVectorSpaceSig formalises that for any group G which forms a vector space with a field F the product group of G and G forms a vector space with respect to the same field.

```
Module Type DualVectorSpaceSig (Group : GroupSig)
    (DualGroup : DualGroupSig Group)(Field : FieldSig)
    (VS : VectorSpaceSig Group Field) <:
        VectorSpaceSig DualGroup Field.
  Import VS.
  Import DualGroup.

  Definition op (a :G)(b :F) := (op a.1 b, op a.2 b).

  Lemma vs_field : field_theory Fzero Fone Fadd Fmul Fsub
        Finv Fdiv FmulInv (@eq F).

  Lemma module_ring : ring_theory Fzero Fone Fadd Fmul
        Fsub Finv (@eq F).

  Lemma mod_dist_Gdot : forall (r : F) (x y : G),
        op (Gdot x y) r = Gdot (op x r) (op y r).

  Lemma mod_dist_Fadd : forall (r s : F) (x : G),
        op x (Fadd r s) = Gdot (op x r) (op x s).

  Lemma mod_dist_Fmul : forall (r s: F) (x : G),
        op x (Fmul r s) = op (op x s) r.

  Lemma mod_id : forall (x : G), op x Fone = x.

  Lemma mod_ann : forall (x : G), op x Fzero = Gone.

  Infix "^" := op.

  Add Field vs_field : vs_field.
  Add Ring module_ring : module_ring.
End DualVectorSpaceSig.
```

APPENDIX D
DETAILS OF COMPOSABILITY

The module ParallelMixable takes as input a mixable and a number N and defines a mixable (itself) which consists of the original mixable being done N times pairwise.

```
Module ParallelMixable (Hack : Nat)
(Message Ciphertext : GroupSig)(Ring: RingSig)
(Field : FieldSig)(VS : VectorSpaceSig Ciphertext Field)
(MVS :  VectorSpaceModuleSameGroup Ciphertext Ring Field VS
    )
(Mix : Mixable Message Ciphertext Ring Field VS MVS)
(NthMessage : GroupNthSig Message Hack)(NthCiphertext :
GroupNthSig Ciphertext Hack)(NthRing : NthRingSig Hack Ring
    )
(NthVS : NthVectorSpaceSig Hack Ciphertext Field
NthCiphertext VS)(NthMVS :
     VectorSpaceModuleSameGroupNthStack
Hack Ciphertext Field Ring VS NthCiphertext NthRing NthVS
MVS)
  <: Mixable NthMessage NthCiphertext NthRing Field NthVS
 NthMVS.
  Import NthMVS.
  Import Hack.

  (* We choose to use different keys for each position.
     We can obvisouly set them to be all the same  *)
  Definition KGR := vector Mix.KGR N.        (* randomness
      for keygen *)
  Definition PK := vector Mix.PK N.          (* public key
      space *)

  Definition M := NthMessage.G.              (* message
      space    *)
  Definition Mop := NthMessage.Gdot.  (* message space is
      an ablelian group *)
  Definition Mzero := NthMessage.Gone.
  Definition Minv := NthMessage.Ginv.
  Definition Mbool_eq := NthMessage.Gbool_eq.

  Definition keygenMix (a : KGR) := Vmap Mix.keygenMix a.
        (* key generation   *)
  Definition enc (pk : PK)(a : NthMessage.G) (b : NthRing.F
      )
  := Vmap2 (fun a b => a b) (Vmap2 Mix.enc pk a) b.

  Lemma M_abgrp : AbeGroup M Mop Mzero Mbool_eq Minv.
  (* proofs redacted *)

  Lemma homomorphism : forall (pk : PK)(m m' : M)
  (r r' : NthRing.F),
    NthCiphertext.Gdot (enc pk m' r')(enc pk m r) =
```

```
      enc pk (Mop m m') (NthRing.Fadd r r').

  Lemma encOfOnePrec : forall (pk : PK)(a : NthRing.F)
    (b : Field.F),
      NthVS.op (enc pk Mzero a) b = enc pk Mzero (op3 a b).

End ParallelMixable.
```

The module ProductMixable takes as input two mixables and defines a mixable (itself) which consists of the original two mixable with pairwise operations.

```
(* Given two mixables which share a field the product of
    the
two mixables is also a mixables *)
Module ProductMixable (M1M M2M M1C M2C : GroupSig)
  (M1Ring M2Ring : RingSig)(Field : FieldSig)(VS1 :
  VectorSpaceSig M1C Field)(VS2 : VectorSpaceSig M2C Field)
  (MVS1 :  VectorSpaceModuleSameGroup M1C M1Ring Field VS1)
  (MVS2 :  VectorSpaceModuleSameGroup M2C M2Ring Field VS2)
  (Mix1 : Mixable M1M M1C M1Ring Field VS1 MVS1)
  (Mix2 : Mixable M2M M2C M2Ring Field VS2 MVS2)
  (* End input *)
  (Message : ProdGroupSig M1M M2M)(Ciphertext :
      ProdGroupSig
  M1C M2C)(Ring : ProdRingSig M1Ring M2Ring)
  (VS : ProdVectorSpaceSig M1C M2C Ciphertext Field VS1 VS2
      )
  (MVS : ProdVectorSpaceModuleSameGroup M1C M2C M1Ring
  M2Ring Field VS1 VS2 MVS1 MVS2 Ciphertext Ring VS)
  <: Mixable Message Ciphertext Ring Field VS MVS.

  Definition KGR := prod Mix1.KGR Mix2.KGR.        (*
      randomness for keygen *)
  Definition PK := prod Mix1.PK Mix2.PK.          (* public
      key space *)

  Definition M := Message.G.                (* message space
      *)
  Definition Mop := Message.Gdot.  (* message space is an
      ablelian group *)
  Definition Mzero := Message.Gone.
  Definition Minv := Message.Ginv.
  Definition Mbool_eq := Message.Gbool_eq.

  Definition keygenMix (a : KGR) :=
    (Mix1.keygenMix a.1, Mix2.keygenMix a.2).     (* key
        generation *)
  Definition enc (pk : PK)(a : Message.G) (b : Ring.F) :=
      (Mix1.enc pk.1 a.1 b.1, Mix2.enc pk.2 a.2 b.2).

  Lemma M_abgrp : AbeGroup M Mop Mzero Mbool_eq Minv.

  Lemma homomorphism : forall (pk : PK)(m m' : M)
    (r r' : Ring.F),
                Ciphertext.Gdot (enc pk m' r')(enc pk m r)
                    =
                  enc pk (Mop m m') (Ring.Fadd r r').

  Lemma encOfOnePrec : forall (pk : PK)(a : Ring.F)
    (b : Field.F),
    VS.op (enc pk Mzero a) b = enc pk Mzero (MVS.op3 a b).

End ProductMixable.
```

APPENDIX E
PRIVACY OF THE MIX NET

There are numerous way to define the privacy of a mix net. One primary distinction is whether or not the decryption of the ciphertexts is modeled; we first address the case where it is not modeled and then comment on the other case at the end.

For the purpose of this paper we will adopt the notion of chosen permutation indistinguishability (IND-CPA$_S$) from [50] which we have recast into concrete security. This notation

$$
\begin{array}{l}
\underline{Exp_{\mathcal{A}}^{\text{IND-CPA}_S-b}(\Sigma, S)} \\
(PK, SK) \leftarrow_r \Sigma.\mathsf{KeyGen} \\
(\pi_1, \pi_2, \mathbf{e}) \leftarrow_r \mathcal{A}(PK) \\
\mathbf{e}' \leftarrow_r S(PK, \pi_b, \mathbf{e}) \\
b \leftarrow_r \mathcal{A}(\mathbf{e}') \\
\textbf{return } b = b'
\end{array}
$$

Fig. 1. (IND-CPA$_S$) experiment

$$
\begin{array}{l}
\underline{Exp_{\mathcal{A}}^{ind-cpa-b}(1^\kappa)} \\
(PK, SK) \leftarrow_r \Sigma.\mathsf{KeyGen} \\
(m_0, m_1) \leftarrow_r \mathcal{A}(PK) \\
r \leftarrow_r \mathcal{R}_{PK} \\
c \leftarrow \Sigma.Enc_{PK}(m_b, r) \\
b \leftarrow_r \mathcal{A}(c) \\
\textbf{return } b = b'
\end{array}
$$

Fig. 2. IND-CPA experiment

is for a verifiable shuffle which a pair $(\Sigma, S, (\mathcal{P}, \mathcal{V}))$ where Σ is an encryption scheme (for our purpose as defined in definition 5), S is a shuffle, and $(\mathcal{P}, \mathcal{V})$ is a proof system which proves $(PK, \mathbf{e}, \mathbf{e}' \in \mathcal{R}_{\Sigma_{PK}}^{shuf})$.

Note that due to the soundness of all mixes, we can express the final output vector of ciphertexts as a function of the composition of the respective permutations and randomness vectors of the mixers; even if we fix the inputs of all but one mixer every possible cumulative permutation and vector is still possible (for the class of encryption schemes we consider). Hence, it suffices to show that only negligible information about the permutation used by the honest mixer leaks and hence we let S be the mix of the mix server who by assumption is honest.

Definition 6. *A shuffle (Σ, S) is said to provide (t, ϵ) Indistinguishability under Chosen Permutation Attack (IND-CPA$_S$) if for every t time adversary $\mathcal{A}$ the advantage of $\mathcal{A}$ in $Exp_{\mathcal{A}}^{\text{IND-CPA}_S}(\Sigma, S)$ (Fig. 1) is at most ϵ. For simplicity we will often drop t and ϵ and refer to (Σ, S) as being (IND-CPA$_S$) secure.*

Definition 7. *We say an encryption scheme Σ is (t, ϵ) IND-CPA secure if no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in $Adv^{ind-cpa}(\mathcal{A}, \kappa)$ (Fig. 2). For simplicity we will often drop t and ϵ and refer to Σ as being IND-CPA secure.*

Theorem 4. *$(\Sigma, S, (\mathcal{P}, \mathcal{V}))$ is IND-CPA$_S$ if the encryption scheme Σ, satisfying definition 5), is IND-CPA.*

Proof sketch. For simplicity we will consider a variant of IND-CPA where the the adversary is allowed polynomially many challenge message pairs and receives either the encryption of all the left messages or all the right messages (called poly-IND-CPA), any IND-CPA secure scheme is also

poly-IND-CPA. We note in passing that for ElGamal, and the other encryption schemes we use in practice, a much tighter security proof is possible which is independent of the number of ciphertexts being mixed.

The challenger receives $(\pi_1, \pi_2, \mathbf{e})$ from the adversary $\mathcal{A}$. Let n be the number of ciphertexts in $\mathbf{e}$, the challenger chooses $\{m_i \leftarrow_r \mathcal{M}\}_{i=1}^n$ and sends $\{0, m_i\}_{i=1}^n$ to the poly-IND-CPA challenger and receives back the ciphertexts $\mathbf{c}$. The challenger then chooses $b \leftarrow_r \{0,1\}$ and forms $\mathbf{e}'$ as $\{\mathbf{c}_i * \mathbf{e}_{\pi_b(i)}\}_{i=1}^n$, which it then passes to $\mathcal{A}$. The adversary then returns its guess b', if the guess is correct then the simulator returns 0 otherwise it returns 1.

The simulation is perfect when the IND-CPA challenge is 0 but completely random when the challenge is 1, hence the advantage of the challenger against poly-IND-CPA is at least half of the advantage of $\mathcal{A}$ against IND-CPA$_S$. □

A. *Considering decryption.*

We do not give a formal definition of privacy for the case considering decryption but several can be found in the literature [46], [47]. The intuitions of the definitions is that the adversary gets to choose the possible messages (m_1, m_2) of the honest senders but is constrained that the set of messages must be equal. In addition it is allowed to add ciphertexts of its own.

If the mix net is IND-CPA$_S$ then any permutation is possible in the output of the ciphertext and hence in the decryption so it suffices to know the set of messages underlying the submitted ciphertexts. Since the set of messages is the same irrespective of the challenge for the honest senders that part is trivial. However, the ciphertexts submitted by the adversary are harder to deal with. Normally, this is dealt with by proofs of knowledge on the submitted ciphertexts or by requiring the encryption is IND-1CCA-Poly [16] or stronger.