Machine-checking Multi-Round Proofs of Shuffle: Terelius-Wikstrom and Bayer-Groth

Thomas Haines
Australian National University

Rajeev Goré
Polish Academy of Science

Mukesh Tiwari
University of Cambridge

Abstract

Shuffles are used in electronic voting in much the same way physical ballot boxes are used in paper systems: (encrypted) ballots are input into the shuffle and (encrypted) ballots are output in a random order, thereby breaking the link between voter identities and ballots. To guarantee that no ballots are added, omitted or altered, zero-knowledge proofs, called proofs of shuffle, are used to provide publicly verifiable transcripts that prove that the outputs are a re-encrypted permutation of the inputs. The most prominent proofs of shuffle, in practice, are those due to Terelius and Wikström (TW), and Bayer and Groth (BG). TW is simpler whereas BG is more efficient, both in terms of bandwidth and computation. Security for the simpler (TW) proof of shuffle has already been machine-checked but several prominent vendors insist on using the more complicated BG proof of shuffle. Here, we machine-check the security of the Bayer-Groth proof of shuffle via the Coq proof-assistant. We then extract the verifier (software) required to check the transcripts produced by Bayer-Groth implementations and use it to check transcripts from the Swiss Post evoting system under development for national elections in Switzerland.

1 Introduction

Two fundamental principles of any free and fair election are the privacy of the voter and the integrity of the ballot. Paper-based elections support voter privacy by ensuring that a cast ballot contains no identifying link back to the voter. They support ballot integrity by ensuring that ballot-boxes are locked and placed under scrutiny at all times prior to their being opened for counting. As many jurisdictions around the world move to electronic voting (evoting), these two properties have to be guaranteed using different means, and cryptographic techniques play a prominent role. But even the most sophisticated cryptographic techniques are useless if their software implementation contains bugs, and so evoting also requires "software independence":

> *"A voting system is software-independent if an (undetected) change or error in its software cannot cause an undetectable change or error in an election outcome." [37]*

Therefore, many modern evoting systems implement "end to end verifiable voting" via a cascade of three notions:

Cast as intended: voters receive verifiable evidence that the electronic ballot correctly captures their intended ballot;

Collected as cast: the election authority publishes verifiable evidence that these electronic ballots are received by the election authority without tampering;

Counted as collected: the election authority publishes verifiable evidence that the result is obtained by correctly counting only and all these collected ballots.

A voter or scrutineers can then inspect this public evidence and accept the results of the election only if all the published evidence can be verified as correct.

The shuffling of the ballots before decryption falls within the purview of counted-as-collected. The most common way to produce publicly verifiable evidence is via so-called "zero-knowledge proofs" (ZKP) [25], which are computer-based protocols in which the "prover" (evoting software) and the "verifier" (scrutineering software) interact with each other with the statement claimed by the ZKP being accepted by the verifier only if their interaction passes previously published correctness criteria. The appellation "zero-knowledge" indicates that the protocol leaks no information other than the truth of the claimed statement.

One crucial use of such ZKPs is to provide publicly checkable evidence of voter privacy and ballot integrity. Typically, the electronic ballots cast (as intended) by the voters are encrypted and transmitted to the election authority. Before counting, the link between the voter and the encrypted ballot is broken by stripping off the voter-identification and passing the initial sequence of encrypted ballots through a "mixnet": a sequence of computers known as "mixers" [16]. As the name

suggests, each mixer receives a sequence of encrypted ballots, re-encrypts them and then "shuffles" their order to produce a different sequence of re-encrypted ballots. Each mixer also produces a ZKP (publicly verifiable evidence) that proves the statement "*my new sequence contains all and only the encrypted ballots from my initial sequence without tampering*". Given at least one honest mixer, it is infeasible to invert the final sequence to learn the initial link to the list of voters. If all published ZKPs are accepted by the (public) scrutineering verifier software then we can guarantee the voter privacy and ballot integrity are preserved in this process.

The ZKPs, called "proofs of shuffle", proving the mixnet's statement are normally the most complicated pieces of cryptography in an evoting system and are prone to both design and implementation bugs. For example, the Scytl-Swiss Post system intended for extended use in national elections in Switzerland was pulled from use because it contained an appalling collection of errors [32]. Virtually every ZKP in the system had some degree of vulnerability. The only errors found in the proof of shuffle ZKP related to incorrect parameter generation; however, given the complexity of that ZKP, it was unclear whether it was secure or simply too complex to analyse properly. Similar issues have been found in Australia [34] and Estonia [40] to which must be added many more general issues [12, 13, 17].

Thus, it is vital to guarantee that the cryptographic theory underlying proofs of shuffle is itself correct and correctly implemented. Only then can we utilise this theory to implement verifiers that check these proofs of shuffle. Since the software required to verify the evidence is much simpler than the entire system, "software independence" is a very valuable feature. But the attacks listed above highlight the crucial fact that "software independence" is vulnerable to design or implementation errors in the verification software. Guaranteeing the correctness of the verifier specification and code is the main aim of our work, and the line of work which we extend.

Here, we assume that the election authority uses mixnets to shuffle the ballots which accurately correspond to the intention of the eligible voters who voted. We therefore use shuffles and mixnets interchangeably. Moreover, we will use the term "proofs of shuffle" even when the protocols we refer to are a strictly weaker beast called "zero-knowledge arguments" (because a computationally unbounded adversary can fake their "proofs.").

Specifically, we use a state-of-the-art interactive proof-assistant called Coq to mathematically analyse the cryptographic theory and practice of the Bayer-Groth proof of shuffle which is used by Scytl and Swiss Post, among others, to provide anonymity in their evoting software. We first encoded the detailed cryptographic design of the Bayer-Groth mixnet, which consists of the steps taken by the prover and verifier, and its security definitions into the logical language of Coq. We then used Coq to produce a machine-checked, and hence formally verified, mathematical proof that the design meets the security definitions. We then utilised Coq's "extraction facility" to obtain an actual OCaml implementation of the verifier (software) and used it to verify test-vectors produced by the Swiss Post's implementation of Bayer-Groth.

Thus, our principal contribution is an executable encoding of the Bayer-Groth proof of shuffle into Coq and machine-checked mathematical proofs of completeness, soundness and zero-knowledge (defined shortly). We have also checked that our encoding produces an OCaml verifier that is compatible with a deployed implementation of the Bayer-Groth mixnet.

Outline Having now introduced our work, we will provide more details in Sec. 1.1, summarise our contributions in 1.2 and limitations in 1.3. We will provide formal definitions in Sec. 2 and summarise our proofs in Sec. 3. Sec. 4 summarises the applications of our work before we conclude in Sec. 5. Sec. 5.1 details future work, particularly around proving the correctness of how the Fiat-Shamir transform is used.

1.1 Background

Having elaborated on the crucial role that zero-knowledge proofs play in producing verifiable evidence for electronic voting schemes, we will now go into detail on the required background for sections 2 and 3 of this paper. Specifically we will introduce in more detail: zero-knowledge proofs and sigma protocols (Sec. 1.1.1), mixnets and proofs of shuffle (Sec.1.1.2), and machine-aided proving and machine-checked proofs (Sec. 1.1.3).

1.1.1 Zero-knowledge proofs and sigma protocols

Zero-knowledge proofs are possible for all languages in the complexity class $\mathcal{NP}$: that is, for all languages which have non-interactive proofs of membership. Recall that one way of looking at $\mathcal{NP}$ is to make explicit the polynomial length witness w which, if given, allows efficient (polynomial time) verification that a given statement s is in the language. We can use a binary relation $\mathcal{R} \subseteq S \times W$ over the set S of all statements and the set W of all witnesses with $(s,w) \in \mathcal{R}$ iff witness $w \in W$ demonstrates that statement $s \in S$ is in the language.

Sigma protocols were first defined by Ronald Cramer [18], they are a particularly simple and efficient kind of zero-knowledge proof and have seen wide deployment; they remain a leading kind of proof both in terms of simplicity and deployment but recent advances in succinct zero-knowledge proofs [26] offer greater efficiency. The first efficient sigma protocol was introduced by Schnorr in [38], several years before the class was defined.

A sigma protocol is a 3-round interactive proof between two parties (a prover P and a verifier V) where P convinces V that she knows a witness for a statement. More concretely, P convinces V that she knows a (private) witness w for a

public $\mathcal{NP}$-relation $\mathcal{R}$ and a public input statement s such that $(s,w) \in \mathcal{R}$; we will formally define a sigma protocol in Sec. 2.1.

Zero-knowledge proofs must satisfy three properties:

Completeness: the protocol will accept with overwhelmingly probability on inputs (s,w) which belong to the relation $\mathcal{R}$. Perfect completeness is used in cases, such as ours, where acceptance is guaranteed.

Zero-knowledge: no information, other than the truth of the statement s, is leaked. For sigma-protocols, a weaker property called honest-verifier zero-knowledge (defined shortly) is sufficient because of the way they are used;

Soundness: the adversary should be caught with at least a certain probability if the claimed statement s is not in the language. Sigma-protocols use the stronger property of special soundness (defined shortly) which requires that a witness w must be efficiently computable if the adversary is able to produce accepting proofs for different challenges.

The three properties are adjusted for sigma protocols as follows:

Completeness: when P and V follow the protocol, V always accepts the proof.

Special soundness: when given two valid proofs for $\mathcal{R}$ of a particular form (which we will define in Sec. 2.1), then w can be extracted efficiently.

Honest verifier zero-knowledge proof: there exists an efficient simulator that produces valid proofs without using the secret input w with the same probability distribution as the transcripts of the real protocol that involves the secret input w.

A key observation in [31] was that only the last property above involves probabilities; even this property can be rendered without explicitly mentioning probabilities by showing a bijection between the output of runs of the honest protocol and the runs of the simulator. This is useful as it allows us to avoid the significant complications that arise in machine-checked proofs when handling probabilities.

To give the reader some insight into sigma protocols, we briefly discuss the most famous sigma protocol, the Schnorr protocol [38]. Given some public input (G,g,q,h) where G is a cyclic group of prime order q, and g and h are two generators of the group G, the prover claims that she knows a witness w for the statement $h = g^w$; the existence of such a w is immediate because g generates the group. However, does the prover know the witness w? In order to convince the verifier, the prover and the verifier do the following:

- the prover picks a random number u, computes $c = g^u$, and sends c to the verifier
- the verifier picks a random challenge e and sends it to the prover
- the prover computes $t = u + e * w$ and sends t to the verifier

The verifier accepts if $g^t = c * h^e$, otherwise rejects.

A sigma protocol of this kinds underpins many of the deployed digital signature schemes. In such schemes g is normally some canonical group generator, h is the user's public key, and the w is the user's signing key.

1.1.2 Mixnets and proofs of shuffle

Since mixnets were introduced, numerous techniques have been proposed for verifying that the mixers followed the protocol, as without this guarantee, mixnets do not provide privacy or ballot integrity: see Haines and Müller [33] for a summary. The most common technique, in practice, is zero-knowledge proofs which provide the highest level of security, require the weakest trust assumptions, and provide good efficiency (at least for discrete log based systems).

The earliest zero-knowledge proof proposal for mixnets still in common use is due to Terelius-Wikström [43]; the interactive variant of this proof of shuffle takes 5 rounds of interaction between the prover and verifier. The proof is constructed by taking an underlying sigma-protocol and adding two extra rounds.

Since the work of Terelius and Wikström, many alternative proposals for (mixnet) proofs of shuffle have been presented with better communication or computational efficiency. The most used of these is that due to Bayer and Groth [8] which achieves sublinear complexity in the proof size and reduces the verifier's computation. But the increased efficiency comes at a cost, the interactive version of the Bayer-Groth proof requires 9 rounds (compared to the 5 of Terelius-Wikström) and the overall proof is constructed from 6 subproofs (compared to the 2 of Terelius-Wikström).

Both proofs of shuffle are normally used in non-interactive variants in which the actions of the verifier are replaced by the outputs of a hash function. These variants are obtained by applying the Fiat-Shamir transform [20] to the interactive version of the proof; care is required in implementing the transform [13] or the soundness of the proof may be compromised. We discuss how to prove the correct use of this transform in our future work Section 5.1.

1.1.3 Background on our approach

As should be clear, all of this sophisticated mathematics must cohere if it is to provide security. But the history of cryptography is littered with proposals whose claims are soon made void by another paper of the form "A new attack on ..." because pen-and-paper mathematics is notoriously prone to errors which are only found after publication. Formalised

mathematics [44] allows us to encode the mathematical statements of the definitions, theorems and the proofs of the theorems into a computer-based "proof assistant" which checks all three artefacts for correctness. For example, the Coq interactive proof-assistant [14] has been developed for decades and is now trusted by the formal methods community.

The relationship between pen-and-paper proofs and machine-aided proofs depends greatly on the kinds of machine-aided techniques applied and the rigour of the initial pen-and-paper proof. Generally the machine-aided techniques can be broken into two categories: symbolic and computational (see [5] for details). In the former—which is used in tools such as Tamarin, Proverif, and Verifpal—there are rarely, if ever, pen-and-paper equivalents to the machine-aided proofs because these tools are intended as automatic tools that produce such proofs. These tools tend to exploit large computational resources to generate the proofs themselves, but may not always succeed. In the computational case—which is used in tools such as Coq and EasyCrypt— the aim is for the user to interact with the proof-assistant until the proof-assistant accepts the user's proof. Thus these proofs tend to follow the same structure as the user's pen-and-paper proofs but with additional details as needed to make rigorous the argument in the pen-and-paper proof. The challenge in constructing machine-aided proofs in the computational model often boils down to working through how to formalise the gaps in the pen-and-paper proofs; our work is no exception.

A significant body of work has been completed on cryptography in Coq, principally in the CertiCrypt project [6]. CertiCrypt has now been largely abandoned in favour of EasyCrypt which is a separate tool for verifying cryptographic protocols.

Verifying sigma protocols and extracting efficient implementations has a significant history due to their simplicity, efficiency, and wide deployment; one prominent early example is by Barthe et al. [7]. Almeida et al. [2] developed a compiler which accepts an abstract description of the statement to be proved and produces a sigma protocol for that statement along with an Isabelle/HOL proof that the sigma protocol is correct. Both of these works form the background and basis for Almeida et al. [3]. There are two salient differences between the approach we follow and that of Almeida et al. [3]. Firstly, their approach is more general while ours is more specific which allows us to define and prove combinations of sigma protocols which are not otherwise available. Secondly, in their own words, the "*catch is that our verification component is highly specialised for (a specific class of) ZK-PoK and relies on in-depth knowledge on how the protocol was constructed.*" However, since we aim at verifying existing deployed implementations we need to prove that the deployed protocol is correct, and extract a correct verifier for it. Almeida et al's work would give us a correct sigma protocol for the statement but not a verifier for the existing system.

The line of work we follow is by Haines, Goré, Tiwari, and Sharma [30, 31] machine-checking various proof protocols and their associated cryptography using Coq, directly, while avoiding explicitly reasoning about probabilities. Their first work [31] focused on proving the properties of sigma-protocols which underlie verifiable election voting schemes; it's main contribution was a (formally) verified verifier for the Helios scheme which they used to verify the 2018 election for director of the International Association for Cryptological Research. They achieved this by formalising in Coq several well-known methods to derive more complicated (three round) sigma protocols from simpler (three round) sigma protocols; we give several examples in Fig. 1, we note that all the examples are fairly direct logical combinations which contrasts with the structural formalisations presented in our work. They also showed that their approach of avoiding explicitly reasoning about probabilities extends to the verifiable mixnets (proofs of shuffle), specifically, the Terelius-Wikström proof of shuffle, although their work had significant limitations. More recently, Haines et al. [30] improved their previous work [31] to produce a much more general result which they used to produce a verified verifier for the CHVote [41] and Verificatum [46] mixnets. Actually, they machine-checked [30] only the soundness of the TW mixnet and also machine-checked the conditions which were generally accepted by the community as being sufficient for correctness and zero-knowledge without machine-checking these latter two requirements *per se*.

Combiner	Statement	Witness	Relation
And	$\mathcal{S} \times \mathcal{S}'$	$\mathcal{W} \times \mathcal{W}'$	$(s_1, s_2), (w_1, w_2)$ are related iff $(s_1, w_1) \in \mathcal{R}$ and $(s_2, w_2) \in \mathcal{R}'$
Or	$\mathcal{S} \times \mathcal{S}'$	$\mathcal{W} \times \mathcal{W}'$	$(s_1, s_2), (w_1, w_2)$ are related iff $(s_1, w_1) \in \mathcal{R}$ or $(s_2, w_2) \in \mathcal{R}'$
Equality	$\mathcal{S} \times \mathcal{S}'$	$\mathcal{W}$	$(s_1, s_2), (w)$ are related iff $(s_1, w) \in \mathcal{R}$ and $(s_2, w) \in \mathcal{R}'$

Figure 1: Combined sigma-protocols derived automatically using Haines *et al* [31] when given two sigma-protocols with statement sets $\mathcal{S}$ and $\mathcal{S}'$, relationships $\mathcal{R}$ and $\mathcal{R}'$, and witness sets $\mathcal{W}$ and $\mathcal{W}'$, respectively, with $\mathcal{W} = \mathcal{W}'$ for Equality.

Here, we give a formal definition of a 5-round sigmaesque-protocol which was not present in their work [30]. Using our new formal definition of a 5-round sigmaesque-protocol, we attempted to show that Terelius-Wikström was complete and enjoyed (honest verifier) zero-knowledge; this we were unable to do because the full protocol is not actually complete for the relationship claimed by Telerius-Wikström [43]; we have tried to contact the authors but so far have not received a response.

Here, we proved the full properties of the Terelius-Wikström proof of shuffle which we now show is actually a zero-knowledge argument, for a similar but different relation, rather than a proof: see Sec. 3.1. We stress that, in practice, the distinction between the original claim and the actual property

of the TW mixnet do not affect the soundness of their proof of shuffle, and hence, the integrity of the election, which was the main focus of the previous work [30]. Nevertheless, our discovery of the mistake in the security claim, which seems to have gone undetected for over a decade, despite repeated use in government binding elections, is significant because detecting even such subtle design errors in such a security critical area is important.

1.1.4 Other machine-assisted approaches to zero-knowledge proofs

The application of machine-assisted proof approaches to zero-knowledge proofs remains a relatively niche area. For example, in the excellent systematisation of knowledge of computer-aided cryptography [5] published at S&P in 2021 only a handful of the 190 cited works pertain to zero-knowledge proofs .

Recent work by Firsov and Unruh [21] has progressed the issues of rewinding which we mention in this paper but it does not impact the issue of the Fiat-Shamir transform which remains an open problem. Much of the research focus on zero-knowledge proofs has switched to a direction called MPC-in-the-Head [35], where MPC stands for Multi-party Party Computation. MPC is a well developed field which allows, as the name suggests, secure computation distributed across multiple parties. The key observation in MPC-in-the-Head is that a single party can simulate an MPC protocol and reveal the views of some of the "participants" to establish the correctness of the computation. The full benefits of this technique are too many to be detailed here but much of the machine-assisted work has pivoted to support this approach. For example, the recent work by Almeida et al. [4] provides the first machine-checked implementations of MPC-in-the-Head using Easycrypt and Jasmin. This work is similar to ours in some respects because it also builds the fully-fledged protocol of interest from its components. However, in their work the components are normally other kinds of cryptography such as secret sharing schemes and MPC protocols whereas ours are mainly simpler zero-knowledge proofs. We do have in common that we both generalise over the commitment scheme and encryption scheme (if any). Concurrently to this, similar work has been done by Sidorenco et al. [39]

There have been a couple of recent papers which bring the state-separating proof paradigm for game-based cryptographic proofs (not be confused with zero-knowledge proofs) into interactive proof-assistants; for example, the SSProve [1] framework does this in Coq and there is similar work in EasyCrypt [19]. Both of these works exhibit a fair degree of modularity which is similar to our work but do so for game-based cryptographic proofs whereas our security proofs for the zero-knowledge proofs are not structured in this way.

1.2 Contributions

The closest previous work [30,31] focused on ordinary sigma-protocols (that is three round zero-knowledge proofs as described previously) with an emphasis on ensuring compatibility with existing deployed systems. The first work [31] included several ways of combing sigma protocols and the second work [30] formalised inside Coq the proof of soundness from TW [43] but the definition remained very bespoke to the particular five round protocol. Our contributions compared to these and other works in the literature are summarised below.

1. **More rounds and parametric structural relations.** We give a formal definition of sigmaesque-protocol protocols which contain 5 rounds and 9 rounds, as required by the Bayer-Groth proof of shuffle. We encoded and machine-checked the structural relationships between these sigmaesque-protocols which then allowed us to machine-check the pen-and-paper proofs of the security of the shuffle while retaining the gist of the pen-and-paper proofs. The alternative would have been to define the 9 round BG protocol, monolithically, and machine-check it directly. Moreover, whereas prior work allowed us to combine two input sigma-protocols into a new sigma-protocol using one of three fixed logical relationships as shown in Figure 1, here we formalise how to add additional rounds to a sigmaesque-protocol to produce a new protocol using a relationship that is a parameter of the combiner rather than fixed. To ensure that a valid sigma protocol can be constructed the user is also required to prove properties of the parameters (see for example Sec. 2.2).

   In a related minor contribution, we generalise the definition of a sigma-protocol from Haines et al. [30] from "zero-knowledge proofs" to "arguments of knowledge" where the extraction of a witness may fail under certain conditions which should occur only with negligible probability, and to allow for protocols where proving zero-knowledge is more complicated.

2. **Clarified the claims of Terelius and Wikström [43].** Utilising these definitions, we found that the Terelius-Wikström proof of shuffle, contrary to the claims in the literature, is not a zero-knowledge proof for the claimed relationship but rather a zero-knowledge argument for a different relationship. We have formally verified (machine-checked) the associated refined claim. We discuss this result and its implications in Sec. 3.1.

3. **Machine-checked proof of Bayer-Groth [8].** We machine-checked the completeness, special-soundness, and honest-verifier zero-knowledge of the Bayer-Groth proof of shuffle (Sec. 3.8) including the five underlying subarguments (for details see Sections 3.3,3.4,3.5, 3.6,3.7). This proof of shuffle is much more complicated

than the zero-knowledge proofs verified in previous work [30, 31].

4. **Tested compatibility with deployed systems** We extracted the verifier (of the Bayer-Groth proof of shuffle) as an OCaml program and used it to verify transcripts produced by the Bayer-Groth implementation inside the Swiss Post evoting system.

Our main contribution (3) is a significant advancement on the state-of-the-art in terms of the complexity of zero-knowledge proofs with machine-checked security. We are facilitated in achieving this by our contributions (1) which enabled the zero-knowledge proof and cryptographic proof (of the zero-knowledge proof) to be broken down into sub-components which collectively prove the overall result. We believe the advantages in handling complexity offered by this approach are absolutely essential to completing complicated machine-checked proofs in a sensible way. Our last contribution (4) is important as it bridges the gap between theory and practice and shows the practical applicability of our work to deployed systems. We consider our middle contribution (2) to be more a curiosity but one which highlights the ability of our approach to catch subtle and longstanding previously undetected errors; we discuss in Sec. 3.1 under what conditions this result would become more important.

1.2.1 The necessity of the new definitions

The main aim of our work is to produce a verified verifier which is compatible with the Bayer-Groth proof of shuffle as deployed in several government binding elections. Given the errors in every other zero-knowledge proof in these systems and the complexity of Bayer-Groth, the value of knowing these zero-knowledge proofs are generated correctly is immense if focused in impact.

During the process of constructing the several thousand line proof of the Bayer-Groth proof of shuffle (for Coq to check) we, by necessity, had to formalise the original pen-and-paper proof. The full version of [8] when proving the security of zero-knowledge arguments built on underlying arguments says over half a dozen times things like "*completeness now follows from the completeness of [the sub-argument]*" or "*honest-verifier zero knowledge now follows from the honest-verifier zero knowledge of [the sub-argument].*" However, as we have seen with Terelius-Wikström such claims are not necessarily true.

Our definitions and structural relations contribute to the literature a precise description of which conditions suffice for the completeness and honest-verifier zero knowledge to follow from the underlying zero-knowledge arguments. To our knowledge no such precise description is currently in the literature; existing pen-and-paper proofs have been able to sidestep this issue by being informal in their argumentation. Our formal structural relations allow the same line of reasoning but without the gaps.

1.3 Limitations

To facilitate a clear understanding of the scope of our work we detail limitations below.

Side channel attacks: The methods we use to machine-check the security of proofs of shuffle and produce the OCaml implementation of the verifier do not preclude the presence of side channel attacks. There are two reasons why this issue is less significant for mixnets than other pieces of cryptography. First, the main software of interest is the verifier which is a public algorithm running on public data. Secondly, the provers are in essence large batch proofs which (normally) operate only once on a given statement and often on air-gapped machines. If, for some reason, timing attacks are a realistic threat in a particular deployment scenario, it should be relatively easy to make the algorithms constant time, using a constant time mathematics library, since the algorithms contain no branching.

Fiat-Shamir transform: We have machine-checked the interactive variants of the ZKP, but when deployed, their non-interactive variant will be used. To close this gap, we would need to machine-check the Fiat-Shamir transform which would involve machine-checked reasoning about the rewinding of random oracles in the presence of arbitrary adversaries. Alas, to the best of our knowledge, no interactive proof-assistant supports such mechanised reasoning. In Section 5.1, we discuss how to prove that the transform is used securely under the assumption that the underlying theory is valid, without machine-checked reasoning about rewinding or arbitrary adversaries.

Code extraction: Coq's extraction facility into OCaml has not been formally verified by anyone. The process is mature but could contain bugs: a limitation shared with all similar work. This does not detract from the value in machine-checking the proofs of shuffle because, in practice, we encourage the use of multiple independently implemented verifiers.

Efficiency: Our extracted OCaml code is roughly half the speed of the Swiss Post Java implementation. We expect this could be rectified by using well-known optimisations [28] but doing so and machine-checking the optimisations is left as future work.

2 Definitions

All the security definitions in our work are variations of sigma-protocols. We use strongly related, but different, definitions

for 3-round, 5-round, and 9-round variants of sigma-protocols. We initially tried to encode a single definition, parameterised on the number of rounds, but found that the Coq types became very unwieldy. Our current definitions, while much more verbose, are also much easier to manipulate inside Coq which seems a good trade-off in making our work re-usable.

For this paper, we have adapted the convention of referencing the name of the Coq objects which relate to the subject under discussion to allow interested readers to look up the formal definitions; notwithstanding this, we aim our paper to make clear our contributions without the reader having to look at the Coq source. We have included a few Coq snippets of security definitions to illustrate their form and structure but we largely avoid including Coq code in the paper.

2.1 3-round protocols

With the exception of some Coq technicalities,[1] our formalisation of a sigma-protocol, defined in a module called `SigmaPlus`, remains close to past work [30]. We give the natural language definition below for completeness.

Definition 1 (Sigma-Protocol)**.** *A protocol $\mathcal{P}$ between a prover P and a verifier V over statement space S and witness space W is a sigma-protocol for relation $\mathcal{R} \subseteq S \times W$ if:*

Form: *$\mathcal{P}$ is of the appropriate 3-move form where the prover P sends a message c, then the verifier V sends a challenge e, then P sends a reply t, and finally V decides to accept or reject based on the statement s and the three messages (c,e,t). It is hopefully obvious that while the prover can base it's computation on the witness w, the verifier is not given this value as input.*

Completeness: *If P and V follow the protocol on statement s and witness w, where $(s,w) \in \mathcal{R}$, the verifier accepts.*

Special soundness: *For any statement s and any pair (c,e,t) and (c,e',t') of accepting conversations, with $e \neq e'$, we can efficiently compute some w such that $(s,w) \in \mathcal{R}$.*

Honest-verifier zero-knowledge: *There exists an efficient simulator, which on statement s and random e outputs an accepting conversation (c,e,t) with the same probability distribution as conversations between the honest P and V on input s.*

We extended the definitions of zero-knowledge and special soundness from Haines et al. [30] as follows. Haines et al. avoid reasoning about probabilities by observing that the standard security definitions for sigma-protocols do not directly refer to (arbitrary) adversaries but only to defined algorithms which internally sample certain random values called (random) coins. By passing these random coins to the algorithms as inputs, we can refer to the transcript produced by the (honest) prover and (honest) verifier for a given statement and witness using specific random values.

Haines et al. [30] used a function `simMap` to constructively show a bijection between the random coins used in the honest runs and the coins used by the simulator. They further require that for a given random coin r, the honest run using r and the simulator using `simMap`(r) produce the same transcript as formalised in Definition 2 below.

We extend `simMap` with extra "trapdoor information" about the statement s, via the function `simMapHelp`: for example, in the Bayer-Groth proof of shuffle, `simMapHelp` is used to provide `simMap` with the discrete log relation between commitment key elements. Since `simMap` exists only to demonstrate the bijection between the random coins spaces, providing it with information that would normally be secret has no impact.

We show the encoding of the extended definition for (honest-verifier) zero-knowledge below.

Definition 2 (Extended Honest Verifier Zero Knowledge)**.** *For all statements s, witnesses w, challenges e, trapdoor information x , honest run random coin r, and simulator random coin t, if the trapdoor information x fits the statement s as defined by simMapHelp and $(s,w) \in Rel$ then on these inputs, the honest run (P1,P0) using r produces the same transcript as the simulator using the coin returned by* `(simMap s w e x r)`*:*

```
Axiom honest_verifier_ZK :
  forall (s : St)(w : W)(e : G)(x : X)(r : R)(t : TE),
    simMapHelp s x ->
    Rel s w ->
    ((P1((P0 s r w), e) r w) =
    simulator s (simMap s w e x r) e) /\
    simMapInv s w e x (simMap s w e x r) = r /\
    simMap s w e x (simMapInv s w e x t) = t.
```

Following [30], the last two lines constructively encode that `simMap` is bijective by using its inverse `simMapInv`.

The other change we make is to special soundness. We extend the definition to what Bootle et al. [15] call l-Special Soundness, the difference is twofold. First, the extractor is allowed to have l-responses with the same commitment but different challenges. Second, we introduce a failure-event parameter to our definition of a sigma-protocol (`SigmaPlus`) which allows the definition of special soundness to be satisfied even if the extractor fails to find a witness.

In practice, the failure-event must be infeasible for an adversary to satisfy, for example breaking the binding property of the commitment scheme or some other event which occurs with negligible probability.

We show the encoding of l-special soundness below.

Definition 3 (l-special soundness)**.** *For all statements s, commitments c, vector e of l-challenges, and vector t of l-responses, if the challenges in vector e are pairwise distinct wrt the underlying group G, and all l-transcripts are valid (meaning that the verifier accepts on $\{(s,c,e_i,t_i)\}_{i=1}^{l}$), then*

[1] We changed the encoding of a sigma-protocol from a class to a module.

either the extractor finds a witness which satisfies the relationship or the failure-event occurs:

```
Axiom special_soundness : forall (s : St)(c : C)
       (e : vector G l)(t : vector T l),
   allDifferent e  ->
   allValid s c e t ->
   Rel s (extractor t e) = true \/ fail_event s c e.
```

Both changes introduce new parameters to the definitions which need to be set sensibly in practice, we will discuss our instantiation for the Bayer-Groth proof of shuffle in the next section. We stress that these security definitions, but not their exact encodings into natural language and Coq, are already inherent in the work by Terelius and Wikström, and Bayer and Groth.

We also machine-checked that any two sigma-protocols, as defined above (with the same challenge space), can be combined to give a sigma-protocol for the conjunction of their two relationships. This is a well known result which has proved useful for building more complicated sigma-protocols from simpler ones; a closely related result was proved in [30].

2.2 5-round protocols

Sigma-protocols can be easily generalised to five rounds by adding two extras rounds in front of the existing three rounds. First the prover sends a commitment and the verifier responds with a challenge, they then follow the existing commit, challenge, and response phases. We encoded this in a Coq module called `SigmaPlus5`. Completeness and honest-verifier zero-knowledge both generalise in the obvious way, the full protocol should accept if the statement and witness pair is in the relationship and the simulator should produce transcripts with the same distribution as the honest parties. Special soundness is a little more complicated, but still essentially the obvious generalisation, instead of l-challenges and responses the extractor receives l-challenges from the first verifier round, and for each of these challenges also receives l' challenges from the second verifier round, and finally l*l' responses such that all the challenges are unique and they are part of accepting transcripts.

We show the encoding of l-l'-special soundness below.

Definition 4 (l-l'special soundness)**.** *For all statements s, first commitments c_0, l first round challenges each with a corresponding l' second round challenge e, and l*l' responses t, if the first round challenges are all different and each list of second round challenges is duplicate free, and all the transcripts are valid then either the extractor finds a witness which satisfies the relationship or the argument is broken because the failure event occurred.*

```
(* The special soundness of sigmaplus5 *)
 Axiom special_soundness : forall (s : St)(c0 : C0)
        (c1 : vector C1 l)
        (e : vector (E0*vector E1 l') l)
        (t : vector (vector T l') l),
      allDifferent e ->
      allValid s c0 c1 e t ->
  Rel s (extractor t e) = true \/ fail_event s (c0,c1) e.
```

A common construction used in both the proofs of shuffle [8, 43] is to build a 5-round sigma protocol for a relationship $\mathcal{R} \subseteq \mathcal{S} * \mathcal{W}$ by extending a 3-round protocol for a relationship $\mathcal{R}' \subseteq \mathcal{S}' * \mathcal{W}''$ with two additional rounds; the resulting 5 round protocol uses the same verification equation as the 3-round protocol. In addition, some protocols supplement the commitment produced by the underlying protocol with additional information.

We define a Coq module called `SigmaPlusTo5` which captures the additional information required to construct a 5-round protocol from a 3-round protocol. The module `SigmaPlus5Comp` encodes how the 5-round protocol is constructed from the underlying 3-round protocol and an instance of `SigmaPlusTo5`, and machine-checks that the result is indeed a 5-round sigma protocol. Those interested in the exact details can review the provided source but we wish to highlight the theorems which the user must machine-check to instantiate `SigmaPlusTo`. We have designed these theorems to be as close as possible to the pen-and-paper proof sketches found in the literature while still being formally sufficient. Below are two examples.

The axiom `to_valid` which must be proven whenever `SigmaPlusTo` is instantiated is below. Prior to instantiating the axiom, the user has provided to Coq the set of random coins used to produce the commitment in the first round, supplementary random coins for the second commitment if any, and the set of challenges for the first round. The user also provides to Coq a mapping from the overall statement, first commitment, and challenge to the underlying statement (`ToSt`) and a mapping from the same and the witness to the underlying witness (`ToWit`).

Definition 5 (to_valid)**.** *For all statements $s \in \mathcal{S}$, witnesses $w \in \mathcal{W}$, random coins r for the first commitment, supplementary random coins r_1 for the second commitment, and challenge e, if s and w are in the relationship $\mathcal{R}$ then the mappings produce a statement and the witness in the relationship $\mathcal{R}'$.*

```
Axiom to_valid : forall s w r r1 e,
    Rel s w ->
   Rel'
        (ToSt (P1 (P0 s r w, e) r1 w))
        (ToWit (P0 s r w, e) r r1 w).
```

Definition 6 (special_soundness)**.** *For all statements $s \in \mathcal{S}$, all first round commitments c, l first challenges e, l second commitments c_1, and l witnesses $\{w_i \in \mathcal{W}'\}_{i=1}^{l}$ if all the statements constructed by the mapping (ToSt) are in the relationship with the corresponding witness, and the challenges are distinct then either the extractor finds a witness or the failure event occurs.*

```
Axiom special_soundness : forall s c (e : vector E l)
        (c1 : vector C1 l)(w : vector sig.W l),
 bVforall3 (fun a b d => Rel' (ToSt (s,c,a,b)) d)
        e c1 w ->
 allDifferent e  ->
 Rel s (extractor w e) \/ fail_event s c e.
```

Here, for a given predicate P and three vectors (v_1, v_2, v_3) of length l, the function `bVforall3` encodes that $\{P(v_{1,i}, v_{2,i}, v_{3,i})\}_{i=1}^{l}$. For brevity we omit a full discussion of our work on 5-round protocols from the paper, the Coq source contains numerous other ways to build 5-round sigma-protocols from underlying 5- and 3-round protocols which are used in encoding and machine-checking the Bayer-Groth proof of shuffle.

2.3 9-round protocols

None of the protocols or sub-protocols in Bayer-Groth are 7-rounds protocols and hence we did not define these, though it would be easy to do. The definition of a 9-round sigma protocol, as captured by the Coq module `SigmaPlus9`, is the natural generalisation of what we have already shown in the 5-round protocols. We also encoded the combination which will be later used in Bayer-Groth: specifically how to combine a 3-round and a 5-round into a 9-round in a specific format. The module `SigmaPlus5plus3to9` encodes the details which the user needs to provide to Coq and machine-check, whereas `SigmaPlus5plus3to9Comp` encode the 9-round protocol and machine-checks that it is secure. This separation of the structural elements of the security proof, which are largely omitted by Bayer and Groth [8], from the specifics helped enormously in handling the complexity of the machine-checked proof of security.

2.4 Commitments and Encryption

In practice, most deployed evoting system make use of ElGamal encryption [22] and Pedersen commitments [36]. However, we follow Haines et al. [30] in using abstract versions of both encryption and commitments. They defined a class of encryption schemes, called Terelius-Wikström compatible encryption schemes, which is closed under both parallel and pairwise composition of ciphertexts. For example, it is common that the ballot is too large to be encrypted into a single ciphertext, so we need to encrypt the message into a vector of ciphertexts; when it comes time to shuffle, we want to mix but keep together the ciphertexts belonging to a single voter: this variant of the encryption scheme and corresponding mixnet can be generated automatically using the techniques from Haines et al. [30]. For example, to machine-check the Terelius-Wikström proof of shuffle for the PPATC encryption scheme [24], which provides everlasting privacy to the votes, Gjosteen et al. showed only that the encryption scheme was in the class, at which point, Coq could automatically use the existing machine-checked proofs that any scheme in this class can be mixed securely.

Due to the complexity of Bayer-Groth, we were forced to narrow the definition of compatible encryption scheme slightly, we give the natural language additions below while their Coq encodings can be found in module `EncryptionSchemePlus`.

Challenge acts appropriately on message and randomness. If a ciphertext $c = Enc(m;r)$, encrypting message m using randomness r, is raised to a challenge e, then the result is a ciphertext which is equivalent to $Enc(m^e, r*e)$.

Encryption of zero is zero. The encryption $Enc(0;0)$ of the message group identity element with the randomness group identity element is the ciphertext group identity element.

These additional restrictions hold for all the use cases of Terelius-Wikström that we are aware of.

3 Results

In this section we will summarise our results for the Terelius-Wikström and the Bayer-Groth proofs of shuffle.

3.1 Terelius-Wikström

The Terelius-Wikström proof of shuffle is the result of combing the work of Wikström [45] with the follow up work by Terelius and Wikström [43]; the implementation of this proof of shuffle called Verificatum [46] has been used for political elections in at least Norway and Estonia and possibly elsewhere. It was, for a significant period, the most commonly used proof of shuffle in political elections, but now only Estonia still uses it, with Bayer-Groth most likely to soon become the most used. Haines et al. [30] machine-checked the soundness of the Terelius-Wikström proof of shuffle for a wide class of encryption schemes and were able to extract the verifier and use it to check proof transcripts produced by the CHVote [41] and Verificatum implementations.

3.1.1 The incompleteness of the TW proof of shuffle

As we have already mentioned, Haines et al. [30] "only" showed the soundness of the TW mixnet and machine-checked what were widely believed [43] to be sufficient conditions for completeness and zero-knowledge. While machine-checking the full properties of the Terelius-Wikström proof of shuffle, we found that it is an argument rather than a proof. We only discovered this weakness after we had encoded our 5-round protocol definition and were trying (unsuccessfully) to machine-check the completeness of the 5-round protocol.

Before we can explain the issue, we need to briefly recap the relationship proved by the Terelius-Wikström proof of shuffle, and some elements of the protocol. The statement is of the form: N input ciphertexts $c_1,...,c_N$, output ciphertext $c'_1,...,c'_N$, a public key pk and commitment parameters $g,g_1,...,g_N$. If Enc is the encryption function of the encryption scheme then the prover must show knowledge of a permutation π and randomness $\bar{r}$ such that for all $i \in [1,...N]$:

$$c'_i = c_{\pi(i)} * Enc(1, r_{\pi(i)}) \qquad (\mathcal{R}_{pk})$$

We denote this relationship by $\mathcal{R}_{pk}$. In the first round of the protocol the prover commits to the permutation π in the form of the corresponding permutation matrix. The verifier then sends a challenge vector $u_1,...,u_N$, alternatively the verifier may send a single element which is expanded into the vector of challenges. The prover then computes u' as $u_{\pi(1)},...,u_{\pi(N)}$.

Letting $\mathcal{R}_{com}$ denote that two distinct openings have been found for one commitment, thereby breaking the binding property, Terelius-Wikström [43, Proposition 3] claims that the protocol is a perfectly complete 5-message honest verifier zero-knowledge proof of knowledge from the relation $\mathcal{R}_{pk} \vee \mathcal{R}_{com}$. Although they show that the underlying 3-round sigma protocol proves a number of relationships, the crucial one is that the prover knows $r_1,...,r_N$ such that:

$$\prod c_i^{u_i} = Enc(1, r_i) * \prod c_i'^{u'_i}$$

The above relationship will be true if the prover's witness is for $\mathcal{R}_{pk}$ but is not true, in general, if the prover's witness is for $\mathcal{R}_{com}$. Consider, for example, the case of ElGamal encryption when each c' decrypts to the group identity and the decryption of $\prod c_i^{u_i}$ is not the group identity. In other words, if the ciphertexts do not have the expected relationship, the span of the vectors of ciphertexts is not guaranteed to overlap, except for the case where u is all zero which occurs only with negligible probability; this means in general that (with overwhelming probability) no witness exists which satisfies the sub-relation. Thus their Proposition 3 [43] is false.

No proof of Proposition 3 appears in the conference version of [43]; the authors state that "*[a] proof of the proposition is given in the full version.*" The full version of the paper can be found in the appendices of Terelius' PhD thesis [42]. The proof found there focus almost entirely on soundness and simply says "*[t]he completeness follows from the completeness of the sigma proof,*" which we have just shown not to be the case since the relationship which the sigma protocol proves may be false.

3.1.2 Our revised proposition

We have machine-checked the following result, which is closely related to their original proposition.

Theorem 1. *The Terelius-Wikström proof of shuffle*[2] *is a perfectly complete 5-round honest verifier zero-knowledge* ***argument*** *of knowledge for the relation* $\mathcal{R}_{pk}$.

By switching the proof of knowledge to an argument of knowledge, we can make the discovery of a witness to $\mathcal{R}_{com}$ the failure event; extraction is allowed to fail when the binding property of the commitment scheme breaks but the verifier is not required to accept just because the prover can break the binding property of the commitment scheme.

Has it mattered? A reader may well ask if the distinction between proof of knowledge and argument of knowledge here is important. In every deployed instance we are aware of, the answer is that it is not important because, in practice, the argument of knowledge is used only when the prover really means to prove a shuffle. Nevertheless, the fact that this erroneous claim has not been detected for more than 10 years shows the value of our work.

Could it matter? For this error to matter, the system would need to intend the proof of shuffle to work when the prover's witness is for $\mathcal{R}_{com}$: that is, the prover's witness consists of two distinct openings to the same commitment. Something similar is done for other kinds of zero-knowledge proofs in some coercion-resistant voting schemes [27, 29] which allow the prover to make "fake" proofs to provide to the coercer. One could imagine a similar system which provided analogous coercion-resistance by allowing the mixnet to fake proofs, but, to our knowledge, no such proposal exists at present within the literature.

3.2 Bayer-Groth

The Bayer-Groth proof of shuffle is a 9-round perfectly complete honest-verifier zero knowledge argument of knowledge of the relationship $\mathcal{R}_{pk}$. The proof of shuffle uses two parameters n and m such that the number of ciphertexts N is equal to n times m. Since in practice, the proofs are constructed once and verified many times, the proof size and verifier complexity is more important than prover complexity. The (asymptotic) proof size for Bayer-Groth is sublinear in the number of ciphertexts and the verification time is roughly a third of Terelius-Wikström. The final shuffle argument in Bayer-Groth is built upon five underlying zero-knowledge arguments.

For the rest of this section we will go through each argument and summarise the relationship it proves; we have included the definition of each argument for completeness and to show the complexity of the pen and paper definitions. We suggest the reader to skim the definitions of the Bayer-Groth arguments to get a feel for the kind of calculations involved;

[2] as encoded in modules `wikSigma` and `WikstromMixnet`

since the security has been machine checked and the compatibility has been tested (Sec. 4), they can be safely skipped if the reader prefers. We will reference where in the Coq source the arguments is machine-checked, along with any interesting observations from that exercise.

3.2.1 Notation

We have largely tried to keep the same notation as the original paper [8] except where the notation was overloaded. The following notation is required to read the following subsections:

- $\mathbb{H}$ is the ciphertext space of the encryption scheme, and $\mathbb{G}$ is the commitment space of the commitment scheme. The field of integers modulo a prime q is denoted $\mathbb{Z}_q$, and the encryption operation is denoted Enc_{pk}. We will denote by $x \leftarrow S$ the independent and uniform sampling of x from a set S.
- For two vectors $\vec{v}_1$ and $\vec{v}_2$, we use $[\vec{v}_1;\vec{v}_2]$ for their concatenation. Sometimes the vectors themselves contain vectors. For example, $[\vec{a}_0;A]$ is the concatenation of the column vector $\vec{a}_0 \in \mathbb{Z}_q^n$ with the matrix $A \in \mathbb{Z}_q^{n\times m}$ resulting in a matrix in $\mathbb{Z}_q^{n\times(1+m)}$.
- We have replaced the responses in the multi-exponentiation argument of knowledge which were previously $\vec{a}, r, b, s, \tau$ with $t_a, t_r, t_b, t_s, t_\tau$ to remove the overlap with the witnesses to the statement.
- We have adjusted some of the indices in the soundness of the multi-exponentiation argument of knowledge to clean up the presentation.
- We have replaced the randomness used in the Hadamard proof previously denoted as $\vec{b}_i$ as $\vec{d}_i$ to remove the overloading with the witness.
- We have changed com_{ck}, which previously referred to matrix commitments, vector commitments, Pedersen commitments and vectors of Pedersen commitments, to refer solely to matrix commitments. We now refer to vector commitments as EPC_{ck} and Pedersen commitments as PC_{ck}. In the original paper [8], when $\vec{a} \in \mathbb{Z}_q^N$ and $N = mn$, the notation com_{ck} denoted $com_{ck}(\vec{a};\vec{r}) = (\text{EPC}_{ck}(a_1,\cdots,a_n;r_1), \cdots, \text{EPC}_{ck}(a_{(m-1)n+1},\cdots,a_{(m-1)n+n};r_n))$. We remove this notation and instead write $\{\text{EPC}_{ck}(a_{ni+1},\cdots,a_{ni+n};r_i)\}_{i=0}^{m-1}$.

3.3 Multi-exponentiation Argument of knowledge

The multi-exponentiation is the only argument which involves the ciphertexts, except for the overall shuffle argument (Sec. 3.8) which uses multi-exponentiation as a subargument. The formal definition is given below but can be summarised as: the prover proves knowledge of the randomness contained in the ciphertexts, raised to some challenge A. Due to the structure of A, as instantiated by the overall shuffle argument, this will later allow us to extract all the random values used to re-encrypt.

The description consists of a common reference string with certain global parameters of the system, in this case, the public key pk of the encryption scheme and the commitment key ck of the commitment scheme.

Common reference string: pk, ck.

Statement: $\vec{C}_1,\ldots,\vec{C}_m \in \mathbb{H}^n, C \in \mathbb{H}$ and $\vec{c}_A \in \mathbb{G}^m$

Prover's witness: $A = \{\vec{a}_j\}_{j=1}^m \in \mathbb{Z}_q^{n\times m}, \vec{r} \in \mathbb{Z}_q^m$ and $\rho \in \mathbb{Z}_q$ such that

$$C = Enc_{pk}(1;\rho)\prod_{i=1}^{m}\vec{C}_i^{\vec{a}_i} \quad \text{and} \quad \vec{c}_A = com_{ck}(A;\vec{r})$$

Initial message: Pick $\vec{a}_0 \leftarrow \mathbb{Z}_q^n$ and $r_0 \leftarrow \mathbb{Z}_q$ and $b_0, s_0, \tau_0, \ldots, b_{2m-1}, s_{2m-1}, \tau_{2m-1} \in \mathbb{Z}_q$ and set $b_m = 0$ and $s_m = 0$ and $\tau_m = \rho$. For $k = 0$ to $2m-1$, with $k \neq m-1$, compute:

$$\vec{c}_{A_0} = \text{EPC}_{ck}(\vec{a}_0;r_0) \qquad c_{B_k} = \text{PC}_{ck}(b_k;s_k)$$

$$E_k = Enc_{pk}(G^{b_k};\tau_k)\prod_{i=0}^{k}\vec{C}_{m-i}^{\vec{a}_{k-i}}$$

Challenge: $x \leftarrow \mathbb{Z}_q$.

Answer: Set $\vec{x} = (1, x, x^2, \ldots, x^m)^T$ and compute

$$t_a = [\vec{a}_0;A]\vec{x} \quad t_r = [r_0;\vec{r}]\cdot\vec{x} \quad t_b = \sum_{k=0}^{2m-1} b_k x^k$$

$$t_s = \sum_{k=0}^{2m-1} s_k x^k \quad t_\tau = \sum_{k=0}^{2m-1} \tau_k x^k$$

Send $t_a, t_r, t_b, t_s, t_\tau$.

Verification: Accept if $c_{B_m} = \text{PC}_{ck}(0;0)$ and $E_m = C$ and

$$[\vec{c}_{A_0};\vec{c}_A]^{\vec{x}} = \text{EPC}_{ck}(t_a;t_r) \qquad \prod_{k=0}^{2m-1} c_{B_k}^{x^k} = \text{PC}_{ck}(t_b;t_s)$$

$$\prod_{k=0}^{2m-1} E_k^{x^k} = Enc_{pk}(G^{t_b};t_\tau)\prod_{i=1}^{m}\vec{C}_i^{x^{m-i}t_a}$$

Comments We directly machine-checked that the multi-exponentiation argument of knowledge meets our encoding of the definition of security in a Coq module called `BGMultiArg`. We set the failure event to be the case where the adversary can find two different openings to the same commitment. The most difficult part of the machine-checked proof was formalising the reasoning around taking the product of diagonals of the matrix (of ciphertexts). The machine-checked proof of soundness is close to the original paper proof, we first prove the various corollaries and lemmas before using them to finish the proof. The proofs of completeness and zero-knowledge which are sketched in about half a page in the original paper, take about 500 lines to machine-check.

3.4 Zero Argument

The zero argument is used to efficiently prove that the inner-product of two committed vectors is zero; it is used in the Hadamard product argument.

Common reference string: pk, ck.

Statement: $\vec{c}_A, \vec{c}_B$ and a specification of a bilinear map

$$* : \mathbb{Z}_q^N \times \mathbb{Z}_q^N \to \mathbb{Z}_q$$

Prover's witness: $A = \{\vec{a}_i\}_{i=1}^m$ and $B = \{\vec{b}_i\}_{i=1}^m \in \mathbb{Z}_q^{n \times m}$ and $\vec{r}, \vec{s} \in \mathbb{Z}_q^n$ such that

$$\vec{c}_A = com_{ck}(A; \vec{r}) \qquad \vec{c}_B = com_{ck}(B; \vec{s}) \qquad 0 = \sum_{i=1}^{m} \vec{a}_i * \vec{b}_i$$

Initial message: Pick $\vec{a}_0, \vec{b}_{m+1} \leftarrow \mathbb{Z}_q^n$ and $r_0, s_{m+1} \leftarrow \mathbb{Z}_q$ and compute

$$\vec{c}_{A_0} = \text{EPC}_{ck}(\vec{a}_0; r_0) \qquad c_{B_{m+1}} = \text{EPC}_{ck}(\vec{b}_{m+1}; s_{m+1})$$

Compute $d_0, ..., d_{2m}$ as the sum of the diagonals

$$\begin{pmatrix} \vec{b}_1 \\ \vec{b}_2 \\ \vdots \\ \vec{b}_{m+1} \end{pmatrix} \begin{pmatrix} \vec{a}_0 & \vec{a}_1 & \cdots & \vec{a}_m \\ \vec{a}_0 * \vec{b}_1 & \vec{a}_1 * \vec{b}_1 & \ddots & \vec{a}_m * \vec{b}_1 \\ \vec{a}_0 * \vec{b}_2 & \vec{a}_1 * \vec{b}_2 & \ddots & \vec{a}_m * \vec{b}_2 \\ \ddots & \ddots & \ddots & \vec{a}_m * \vec{b}_m \\ \vec{a}_0 * \vec{b}_{m+1} & \ddots & \ddots & \vec{a}_m * \vec{b}_{m+1} \\ & d_0 & \cdots & d_{m-1} \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ d_{2m} \\ \vdots \\ d_{m+1} \\ d_m \end{matrix}$$

Pick $\vec{t} = (t_0, ..., t_{2m}) \leftarrow \mathbb{Z}_q^{2m+1}$ and set $t_{m+1} = 0$ and compute commitments $\{\vec{c}_{D_i} = \text{PC}_{ck}(d_i; t_i)\}_{i=0}^{2m}$. Send $\vec{c}_{A_0}, \vec{b}_{m+1}, \vec{c}_D$.

Challenge: $x \leftarrow \mathbb{Z}_q$

Answer:

$$\vec{a} = \sum_{i=0}^{m} x^i \vec{a}_i \qquad r = \sum_{i=0}^{m} x^i r_i \qquad \vec{b} = \sum_{i=1}^{m+1} x^{m+1-i} \vec{b}_i$$

$$s = \sum_{i=1}^{m+1} x^{m+1-i} s_i \qquad t = \sum_{i=0}^{2m} x^i t_i$$

Send $\vec{a}, \vec{b}, r, s, t$.

Verification: Accept if $\vec{c}_{D_{m+1}} = \text{PC}_{ck}(0; 0)$ and

$$\prod_{i=0}^{m} \vec{c}_{A_i}^{x^i} = \text{EPC}_{ck}(\vec{a}; r) \qquad \prod_{i=1}^{m+1} c_{B_i}^{x^{m+1-i}} = \text{EPC}_{ck}(\vec{b}; s)$$

$$\prod_{i=0}^{2m} \vec{c}_{D_i}^{x^i} = \text{PC}_{ck}(\vec{a} * \vec{b}; t)$$

The Schwarz-Zippel Lemma The zero argument is the first of the Bayer-Groth subarguments to depend on the Schwarz-Zippel lemma. We therefore take this opportunity to explain this lemma and how we encode it.

Recall that a polynomial is a zero polynomial iff all coefficients are zero. The Schwarz-Zippel lemma states that if $f(x_1, ..., x_N)$ is a non-zero polynomial of degree d and we pick a point e from $\mathbb{Z}_q^N$ randomly, then the probability that $f(e) = 0$ is at most d/q. Since q was already required to be exponentially large, this means that, in practice, the chance of the polynomial at point e being zero without the polynomial being zero is negligible. This lemma can be used to check that two polynomials f, f' are equal by checking that $f - f'$ is the zero polynomial and is widely used in ZKPs that reason about polynomials, including in the Terelius-Wikström proof of shuffle and several of the Bayer-Groth subarguments.

It is crucial that the point e at which the polynomial is sampled is independent of the polynomial, furthermore, in a zero-knowledge proof, the polynomial is often part of the witness and we do not wish to leak it. To resolve this, the prover commits to the polynomial first and then the verifier replies with a challenge which is a random point $e \in \mathbb{Z}_q^N$ at which the polynomial will be evaluated.

To encode the failure condition, we state that no witness will not need to extracted if the commitment can be opened to a non-zero polynomial which evaluates to zero at the following challenge. In practice, the use is often more complicated and the coefficients of the polynomial are determined by the values of several commitments. Nevertheless, the crux is that the polynomial is determined by values committed to before the challenge is sent.

The 5-round variant of the Terelius-Wikström proof of shuffle also depends on the Schwarz-Zippel lemma. This was handled in an ad hoc way by Haines et al. [30] and in our updated Coq proofs for Terelius-Wikström using the same technique above.

Comments We directly machine-checked that the zero-argument meets our definition of security in a module called `BGZeroArg`. These machine-checked proofs, while still fairly verbose, are much simpler than the multi-exponentiation argument.

3.5 Hadamard Product Argument

For a matrix A in a given commitment and a vector $\vec{b}$ in a different commitment, the Hadamard product argument is used to prove that $\vec{b} = \prod_{i=1}^{m} \vec{a}_i$.

Common reference string: pk, ck

Statement: $\vec{c}_A, c_b$

Prover's witness: $A = (\vec{a}_1, ..., \vec{a}_m), \vec{r}, \vec{b}$, and s such that

$$\vec{c}_A = com_{ck}(A; \vec{r}) \qquad c_b = \text{EPC}_{ck}(\vec{b}; s) \qquad \vec{b} = \prod_{i=1}^{m} \vec{a}_i$$

Initial message: Define $\vec{b}_1 = \vec{a}_1$ and $\{\vec{b}_i = \prod_{j=1}^{i} \vec{a}_j\}_{i=2}^{m-1}$ and $\vec{b}_m = \vec{b}$. Pick $s_2, ..., s_{m-1} \leftarrow \mathbb{Z}_q$ and compute $\{\vec{c}_{B_i} = \text{EPC}_{ck}(\vec{b}_i; s_i)\}_{i=2}^{m-1}$. Define $s_1 = r_1$ and $s_m = s$ and set $\vec{c}_{B_1} = \vec{c}_{A_1}$ and $\vec{c}_{B_m} = c_b$. Send $\vec{c}_B$.

Challenge: $x \leftarrow \mathbb{Z}_q^*, y \leftarrow \mathbb{Z}_q$

Answer: Define the bilinear map $* : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \rightarrow \mathbb{Z}_q$ by $(a_1, ..., a_n)^T * (d_1, ..., d_n)^T = \sum_{j=1}^{n} a_j d_j y^j$. Define $\{c_{D_i} = \vec{c}_{B_i}^{x^i}\}_{i=1}^{m-1}$ and $c_D = \prod_{i=1}^{m-1} \vec{c}_{B_{i+1}}^{x^i}$ and $c_{-1} = \text{EPC}_{ck}(-\vec{1}; 0)$ and engage in the Zero Argument for the committed values satisfying the relation

$$0 = \sum_{i=1}^{m-1} \vec{a}_{i+1} * \vec{d}_i - \vec{1} * \vec{d}$$

The prover's witness in this argument consists of the openings of $\vec{c}_{A_2}, ..., \vec{c}_{A_m}, c_{-1}$ and the openings of $c_{D_1}, .., c_{D_{m-1}}, c_D$.

Verification: Check that $\vec{c}_{B_1} = \vec{c}_{A_1}$ and $\vec{c}_{B_m} = c_b$. Accept if the zero argument is valid.

Comments The Hadamard product argument is a five-round protocol, which builds upon the 3-round zero argument, and it is here that we begin to gain good value from the combinations we proved alongside our definitions. We encode the additional information required to construct the 5-round protocol from the underlying 3-round protocol as a `SigmaPlusTo5sim` module called `BGHadProd`. The encoded definitions and machine-checked proofs in this module are less than half the length of the encoding of the zero or multi-exponentiation arguments.

3.6 Single Value Product Argument

The single value product argument is a 3-round protocol which proves that the product of an opening to a given commitment is equal to a known value.

Common reference string: pk, ck

Statement: $c_a \in \mathbb{G}$ and $b \in \mathbb{Z}_q$

Prover's witness: $\vec{a} \in \mathbb{Z}_q^n$ and $r \in \mathbb{Z}_q$ such that

$$c_a = \text{EPC}_{ck}(\vec{a}; r) \quad \text{and} \quad b = \prod_{i=1}^{n} a_i$$

Initial message: Compute

$$\{b_i = \prod_{j=1}^{i} a_j\}_{i=1}^{n}$$

Pick $d_1, ..., d_n, r_d \leftarrow \mathbb{Z}_q$. Define $\delta_1 = d_1$ and $\delta_n = 0$ and pick $\delta_2, ..., \delta_{n-1} \leftarrow \mathbb{Z}_q$. Pick $s_1, s_x \leftarrow \mathbb{Z}_q$ and compute

$$c_d = \text{EPC}_{ck}(\vec{d}; r_d) \quad c_\delta = \text{EPC}_{ck}(-\delta_1 d_2, ..., -\delta_{n-1} d_n; s_1)$$

$$c_\Delta = \text{EPC}_{ck}(\delta_2 - a_2\delta_1 - b_1 d_2, ..., \delta_n - a_n\delta_{n-1} - b_{n-1} d_n; s_x)$$

Send c_d, c_δ, c_Δ

Challenge: $x \leftarrow \mathbb{Z}_q$

Answer: Compute

$$\tilde{a}_1 = xa_1 + d_1 \quad ... \quad \tilde{a}_n = xa_n + d_n \quad \tilde{r} = xr + r_d$$
$$\tilde{b}_1 = xb_1 + \delta_1 \quad ... \quad \tilde{b}_n = xb_n + \delta_n \quad \tilde{s} = xs_x + s_1$$

Send: $\tilde{a}_1, \tilde{b}_1, ..., \tilde{a}_n, \tilde{b}_n, \tilde{r}, \tilde{s}$.

Verification: The verifier accepts if

$$c_a^x c_d = \text{EPC}_{ck}(\tilde{a}_1, ..., \tilde{a}_n; \tilde{r})$$

$$c_\Delta^x c_\delta = \text{EPC}_{ck}(x\tilde{b}_2 - \tilde{b}_1\tilde{a}_2, ..., x\tilde{b}_n - \tilde{b}_{n-1}\tilde{a}_n; \tilde{s})$$

$$\tilde{b}_1 = \tilde{a}_1 \qquad \tilde{b}_n = xb$$

Comments We encode the definitions and machine-checked security proofs for the single value product argument in a module called `BGSingleProd`. The machine-checked proofs are straightforward and similar to the pen-and-paper version.

3.7 Product Argument

The product argument is used to show that the product of all the elements in a committed matrix is equal to a particular known value.

Common reference string: pk, ck

Statement: $\vec{c}_A \in \mathbb{G}^m$ and $b \in \mathbb{Z}_q$

Prover's witness: $A \in \mathbb{Z}_q^{n \times m}$ and $\vec{r} \in \mathbb{Z}_q^m$ such that

$$\vec{c}_A = com_{ck}(A; \vec{r}) \quad \text{and} \quad \prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = b$$

Initial message: Pick $s \leftarrow \mathbb{Z}_q$ and compute $c_b = \text{EPC}_{ck}(\prod_{j=1}^m a_{1j}, ..., \prod_{j=1}^m a_{nj}; s)$. Send c_b to the verifier.

Subarguments: Engage in an Hadamard Product Argument of the relation $\vec{c}_A = com_{ck}(A; \vec{r})$ and $c_b = \text{EPC}_{ck}(\prod_{j=1}^m a_{1j}, ..., \prod_{j=1}^m a_{nj}; s)$. Engage in a Single Value Product Argument that b is equal to the product of $\prod_{j=1}^m a_{1j}, ..., \prod_{j=1}^m a_{nj}$.

Verification: The verifier accepts if both arguments accept.

Comments The product argument builds upon the 5-round Hadamard product argument and the 3-round single value product argument. We encode the additional information required to build a 5-round protocol from an underlying 5-round and 3-round protocol in the module `SigmaPlus5To5` and then provided the details of how to construct the machine-checked proofs of security in the module `SigmaPlus5to5Comp`. Having taken care of the structural issues, the actual encodings of the definitions and machine-checked proofs of the product argument, contained in the Coq module `ProdArg`, are very short at only 200 lines.

3.8 Shuffle Argument

The shuffle argument is the Bayer-Groth proof of shuffle. It draws upon the product argument and the multi-exponentiation argument.

Common reference string: pk, ck.

Statement: $\vec{C}, \vec{C}' \in \mathbb{H}^N$ with $N = mn$.

Prover's witness: $\pi \in \Sigma_N$ and $\vec{\rho} \in \mathbb{Z}_q^N$ such that $\vec{C}' = Enc_{pk}(\vec{1}; \vec{\rho})\vec{C}_\pi$.

Initial message: Pick $\vec{r} \leftarrow \mathbb{Z}_q^m$, set $\vec{a} = \{\pi(i)\}_{i=1}^N$ and compute $\vec{c}_A = \{\text{EPC}_{ck}(a_{ni+1}, ..., a_{ni+n}; r_i)\}_{i=0}^{m-1}$. Send $\vec{c}_A$.

Challenge: $x \leftarrow \mathbb{Z}_q$.

Answer: Pick $\vec{s} \in \mathbb{Z}_q^m$, set $\vec{b} = \{x^{\pi(i)}\}_{i=1}^N$ and compute $\vec{c}_B = \{\text{EPC}_{ck}(b_{ni+1}, ..., b_{ni+n}; s_i)\}_{i=0}^{m-1}$. Send $\vec{c}_B$.

Challenge: $y, z \leftarrow \mathbb{Z}_q$.

Answer: Define $\vec{c}_{-z} = \{\text{EPC}_{ck}(-\vec{z}; 0)\}_{i=1}^m$ and $\vec{c}_D = \vec{c}_A^{\,y}\vec{c}_B$. Compute $\vec{d} = y\vec{a} + \vec{b}$ and $\vec{t} = y\vec{r} + \vec{s}$. Engage in a product argument of openings $d_1 - z, ..., d_N - z$ and $\vec{t}$ such that

$$\vec{c}_D\vec{c}_{-z} = \{\text{EPC}_{ck}(d_{ni+1} - z, ..., d_{ni+n} - z; t_i\}_{i=0}^{m-1}$$

$$\prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(yi + x^i - z).$$

Compute $\rho = -\vec{\rho} \cdot \vec{b}$ and set $\vec{x} = (x, x^2, ..., x^N)^T$. Engage in a multi-exponentiation argument of $\vec{b}, \vec{s}$ and ρ such that

$$\vec{C}^{\vec{x}} = Enc_{pk}(1; \rho)\vec{C}'^{\vec{b}} \quad \text{and}$$

$$\vec{c}_B = \{\text{EPC}_{ck}(b_{ni+1}, ..., b_{ni+n}; s_i)\}_{i=0}^{m-1}$$

Verification: The verifier accepts if the product argument and the multi-exponentiation argument are both valid.

Comments We encode the extra information required to construct a 9-round protocol from a 5-round and 3-round protocol in the module `SigmaPlus5plus3to9`. The module `SigmaPlus5plus3to9Comp` then encodes how to construct the 9-round protocol from this information. The specifics of the shuffle argument are encoded in the 500 line module `ShuffleArg`.

4 Applications

In the previous section, we summarised how we encoded and machine-checked the Bayer-Groth proof of shuffle. Particularly useful was our separation of the encoding of the structural reasoning about how protocols can be combined from the specifics of the arguments by Bayer and Groth. However, how can we be sure that our encoding captures the original pen-and-paper definitions or the actual computer implementations being used? To answer either of these in a machine-checked way is not feasible and ultimately unnecessary. If our concern is the integrity of the elections using the Bayer-Groth proof of shuffle, it suffices to have a correct verifier which works on the proof transcripts produced by the deployed systems. This is what we have shown, as explained next.

4.1 Testing with Swiss Post's system

Background One of the earliest uses of a proof of shuffle in an electronic voting system deployed for government elections was in Norway [23]. That system, produced by the

prominent vendor Scytl, made use of the Verificatum implementation we have alluded to earlier. Verificatum remained the most commonly used proof of shuffle until recently.

The original paper by Bayer and Groth [8] contains the benchmarks of their C++ implementation of their proof of shuffle. We believe that, at one time, this implementation was available on github but this appears to no longer be the case. Scytl created their own implementation of the Bayer-Groth proof of shuffle no later than 2018, this implementation has subsequently been used, to our knowledge, for government elections in Australia and Switzerland. We have already mentioned the issue discovered in 2019 which invalidated the security of the Scytl implementation due to insecure generation of the commitment parameters. Following that debacle, Swiss Post bought the rights[3] to the Scytl system and now manages the development in house and is currently working to have it certified for use in Swiss elections. We have therefor tested our extracted OCaml verifier on the current version of the Swiss Post implementation. The current Swiss Post implementation contains some refactoring of the old Scytl code, but we expect that our verifier will work for the Scytl implementation since it works for Swiss Post one; however, since the current Scytl implementation is not public, we are unable to check this.

We used Coq's extraction facility to produce an OCaml implementation of the verifier in a file called `lib.ml`. We changed this code to use the native OCaml method for computing the modulus of one number with respect to another, this is necessary for performance reasons. The Swiss Post implementation is freely available on github[4] and includes some test vectors in a JSON format. We parsed this JSON file and fed the values into our extracted verifier, see `main.ml`. After fixing some minor compatibility issues in our parser, the tests were passing successfully. We also tried feeding our verifier some invalid data as a sanity check which it rejected as expected. For the purpose of testing our compatibility we hard coded the challenges produced by the hash function standing in for the verifier; we discuss how to do this securely in our future work 5.1.

The redundant data The test vectors provided by Swiss Post contained more data than our verifier was expecting, which was part of the reason why writing the parser took more time than expected. Upon investigation, it turns out that Scytl and Swiss Post had actually followed the original paper by Bayer and Groth [8] in providing this redundant data. The verification equations of the Bayer Groth proof of shuffle involve checking that certain commitments received from the prover are either equal to certain parts of the statement or equal to certain constants, see the definitions in Section 3. This occurs in the Multi-exponentiation, Zero, and Hadamard product arguments. These redundant values do not need to be transmitted, we suspect they were included in the BG paper to simplify notation; this all leads to the slightly odd scenario where our verifier may accept a transcript which the Swiss Post implementation rejects because we ignore the redundant data. The presence of this data is asymptotically irrelevant to the size of the proof of shuffle.

Efficiency We tested the efficiency of our verifier by verifying a shuffle of 6,400 votes each encrypted in two ElGamal ciphertexts, a total 25,600 ElGamal ciphertexts including the input and output. Our implementation verified the proof in 16 minutes and 13 seconds running on a single core of an Intel i5 Macbook Pro. The same test data took the Swiss Post implementation 7 minutes and 40 seconds to verify on the same machine. The relatively slow performance is due to the use of a 2048bit safe prime group, that is the group of a quadratic residues modulo a prime p which is itself equal to $2q+1$ where q is another prime. A hypothetical elliptic curve based implementation would be several orders of magnitude faster. If someone did wish to use our implementation for verifying a large election, we would suggest parallelising the exponentiation (which is the most expensive operation) and using some of the tricks we alluded to earlier.

5 Conclusion

We have significantly extended the previous work of Haines et al. [30] to formally verify (machine-check) the soundness, completeness and zero-knowledge properties of both the TW and BG proofs of shuffle using the Coq proof-assistant. In so doing, we have found that the TW "proof of shuffle" is not actually a proof, but a weaker notion of "argument of knowledge" for a relation that is different from the one claimed by TW. Our finding exposes a gap in the accepted wisdom of the cryptographic community over the past ten years, albeit one which does not undermine the soundness of the proof of shuffle. We have also extracted from Coq a formally verified (machine-checked) verifier in OCaml to check the evidence produced by implementations of the BG mixnet. Finally, we have shown that the current Swiss Post implementation of the BG mixnet does indeed produce valid BG proofs of shuffle on the test cases they have provided.

Now, for argument's sake, suppose that there is a bug deep inside the Swiss Post implementation of the BG mixnet. Suppose further that Swiss Post uses it to count some important election in Switzerland and suppose they publish the proofs of shuffle produced by their implementation of BG on the election ballots. Anyone with access to these proofs can now implement a verifier according to our proven secure executable specification, with appropriate use of the Fiat-Shamir transform, and check the proofs of shuffle published by Swiss Post. "Hang on", we hear you say, "You just assumed that there

[3] As far as we aware, the exact rights are not publicly known.
[4] https://gitlab.com/swisspost-evoting/crypto-primitives

is some subtle bug inside the Swiss Post implementation, so why would anyone trust the result of their election count, even if your verifier accepted it?". Because, if you run our verifier on the published proofs of shuffle from their election, and our verifier does not complain, then we can assert with confidence that the subtle bug, or any other bug for that matter, did not affect this **particular run** of the mixnet on these **particular** ballots. That is, for this particular election result, you can safely believe the mixnet's claim that "my new sequence contains all and only the encrypted ballots from my initial sequence without tampering".

5.1 Future work

The downside of interactive proofs is in the name; they are interactive. In electronic voting, this is normally not a feature since we would like the proofs (evidence) to be produced once and for any other scrutineering party to be able to verify later at their leisure. Fortunately, Fiat and Shamir proposed a solution to this problem [20] which is now called the Fiat-Shamir transform; by replacing the verifier's interaction with the output of a hash function we can get a non-interactive version of the zero-knowledge proof which is provably secure in the a security model called the random oracle model.

Despite the attraction of the Fiat-Shamir transform, as is so often the case, the devil is in the details. Bernhard, Pereira, and Warinschi [13] show that, in practice, the deployed zero-knowledge proofs using the Fiat-Shamir transform may offer no security. The issue which distinguishes between what Bernhard et al. call the weak transform and the strong transform is the information that is input into the hash function. An example of the weak variant of the transform is found in the famous foundational paper of the random oracle model [9]. The weak variant only hashes the commitment in the zero-knowledge proof but not the statement. The discrepancy occurs because the weak transform causes the post-conditions of the forking lemma [10] to fail to meet the preconditions of the special-soundness extractor, see 5.1.5 of [11] for full details. As we shall explain shortly, this discrepancy can be avoided by ensuring that the statement is (uniquely) included in the hash.

Definitions Assume the existence of some underlying hash function or key derivation function H which is used to produce the challenges for the protocol. We are interested in the information given as input to this function H and the format of this information. We model this as a function M which takes the statement, commitment, and some auxiliary information and produces a bit string which is passed to H. We will denote the statements by $\mathcal{S}$, the commitments by $\mathcal{C}$, and the auxiliary information by $\mathcal{A}$, thus M has domain $\mathcal{S} \times \mathcal{C} \times \mathcal{A}$ and range $\{0,1\}^*$. We will denote the output of the hash function when used as a challenge as e.

Definition 7 (Strong Fiat-Shamir)**.** *A given sigma protocol σ, made non-interactive using the Fiat-Shamir transform (by replacing the challenge with $H(M(*))$), is implemented strongly if: $\forall s,s' \in \mathcal{S}, c,c' \in \mathcal{C}, a,a' \in \mathcal{A}$ if $M(s,c,a) = M(s',c',a')$ then $(s = s') \wedge (c = c')$.*

As discussed above, the reason the proof fails [11] in the case of the weak transform is that the post-condition on the forking lemma is an insufficient pre-condition on special-soundness. Specifically, the pre-condition on special-soundness is that:

$$(1)\ s = s' \qquad (2)\ c = c' \qquad (3)\ e \neq e'$$

But only the third condition is guaranteed to be true in general, whereas the first two follow from the information input into the hash function. In contrast, the first two conditions follow immediately from the properties of M in Definition 7.

The definition is fairly straightforward but the difficulty will be in machine-checking that the Swiss Post implementation has this property, which we leave as future work.

Acknowledgments

We would like to thank the shepherd and reviewers for their excellent feedback. Thomas Haines is the recipient of an Australian Research Council Australian Discovery Early Career Award (project number DE220100595). Rajeev Goré supported by FWF project P 33548 and the National Centre for Research and Development, Poland (NCBR), and the Luxembourg National Research Fund (FNR), under the PolLux/FNR-CORE project STV (POLLUX-VII/1/2019).

Availability

Our Coq source files and extracted OCaml verifier can be found at https://github.com/gerlion/secure-e-voting-with-coq.

References

[1] Carmine Abate, Philipp G. Haselwarter, Exequiel Rivas, Antoine Van Muylder, Théo Winterhalter, Catalin Hritcu, Kenji Maillard, and Bas Spitters. Ssprove: A foundational framework for modular cryptographic proofs in coq. In *CSF*, pages 1–15. IEEE, 2021.

[2] 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 *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 151–167. Springer, 2010.

[3] 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.

[4] José Bacelar Almeida, Manuel Barbosa, Manuel L. Correia, Karim Eldefrawy, Stéphane Graham-Lengrand, Hugo Pacheco, and Vitor Pereira. Machine-checked ZKP for NP relations: Formally verified security proofs and implementations of mpc-in-the-head. In *CCS*, pages 2587–2600. ACM, 2021.

[5] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. Sok: Computer-aided cryptography. In *IEEE Symposium on Security and Privacy*, pages 777–795. IEEE, 2021.

[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] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In D. Stinson, editor, *Advances in Cryptology – Crypto '93, 13th Annual International Cryptology Conference*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 1993.

[10] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS*, pages 390–399. ACM, 2006.

[11] David Bernhard. *Zero-knowledge proofs in theory and practice*. PhD thesis, University of Bristol, UK, 2014.

[12] 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.

[13] 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.

[14] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 327–357. Springer, 2016.

[15] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[16] Véronique Cortier and Ben Smyth. Attacking and Fixing Helios: An Analysis of Ballot Secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF, 2011*, pages 297–311, 2011.

[17] Ronald Cramer. Modular design of secure yet practical cryptographic protocols. *PhD thesis, Aula der Universiteit*, 1996.

[18] François Dupressoir, Konrad Kohbrok, and Sabine Oechsner. Bringing state-separating proofs to easycrypt - A security proof for cryptobox. *IACR Cryptol. ePrint Arch.*, page 326, 2021.

[19] 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.

[20] Denis Firsov and Dominique Unruh. Zero-knowledge in easycrypt. *IACR Cryptol. ePrint Arch.*, page 926, 2022.

[21] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[22] Kristian Gjøsteen. Analysis of an internet voting protocol. *IACR Cryptol. ePrint Arch.*, page 380, 2010.

[23] Kristian Gjøsteen, Thomas Haines, and Morten Rotvold Solberg. Efficient mixing of arbitrary ballots with everlasting privacy: How to verifiably mix the PPATC scheme. In *NordSec*, volume 12556 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2020.

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

[25] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[26] Sandra Guasch and Paz Morillo. How to challenge and cast your e-vote. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, pages 130–145, 2016.

[27] Rolf Haenni and Philipp Locher. Performance of shuffling: Taking it to the limits. In *Financial Cryptography Workshops*, volume 12063 of *Lecture Notes in Computer Science*, pages 369–385. Springer, 2020.

[28] Thomas Haines. Cronus: Everlasting privacy with audit and cast. In *NordSec*, volume 11875 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2019.

[29] Thomas Haines, Rajeev Goré, and Bhavesh Sharma. Did you mix me? formally verifying verifiable mix nets in electronic voting. In *IEEE Symposium on Security and Privacy*, pages 1748–1765. IEEE, 2021.

[30] 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.

[31] 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.

[32] Thomas Haines and Johannes Müller. Sok: Techniques for verifiable mix nets. In *CSF*, pages 49–64. IEEE, 2020.

[33] J. Alex Halderman and Vanessa Teague. The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election. In *Proc. 5th International Conference on E-voting and Identity (VoteID '15)*, 2015.

[34] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30. ACM, 2007.

[35] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO 1991*, pages 129–140, 1991.

[36] 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.

[37] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, 1991.

[38] Nikolaj Sidorenco, Sabine Oechsner, and Bas Spitters. Formal security analysis of mpc-in-the-head zero-knowledge protocols. In *CSF*, pages 1–14. IEEE, 2021.

[39] 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.

[40] The state of Geneva. CHvote. https://github.com/republique-et-canton-de-geneve/chvote-protocol-poc, 2018.

[41] Björn Terelius. *Some aspects of cryptographic protocols: with applications in electronic voting and digital watermarking*. PhD thesis, KTH Royal Institute of Technology, 2015.

[42] 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.

[43] Various. Special issue on formal proof. *Notices of the American Mathematical Society*, 55, Issue 11, 2008.

[44] 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.

[45] Douglas Wikström. Verificatum. https://github.com/verificatum/verificatum-vcr, 2018.