Literature Review

u6650903 Ratmir Mugattarov

May 2023

1 Introduction

1.1 Election

The election process is a critical component of democracy. However, traditional paper voting faces numerous challenges. Voting and tallying are labor-intensive, the vote counting process is not transparent, audit is not available for the general public, and the process does not scale well. As society advances rapidly, our democratic processes must adapt to meet new demands. Electronic voting provides the benefits of automation and online access; however, it introduces its own set of problems. Since electronic voting relies on software, it inherently contains errors, which can be difficult to eliminate entirely [1]. Fixing one issue often introduces another. Numerous instances of errors in electronic voting have been documented [2, 3, 4]. Such errors undermine public trust in the electoral process. Given the importance of elections, we cannot tolerate more than zero errors in the election process. Therefore, if we want to employ electronic voting, it is crucial to provide a zero-error guarantee in electronic voting software for it to be considered both reliable and trustworthy.

1.2 Software

To ensure an error-free election, it is vital to provide the same guarantee for the software conducting the voting. Software errors are common, mainly because the development process inherently doesn't guarantee correctness. Historically, there is a gap between the theoretical correctness and the actual operation of the software. The industry standard for ensuring operational correctness is through testing, but this method only identifies existing errors and cannot assure their complete absence. A more effective, though less frequently used, method is a formal proof of the program code or algorithm. Regrettably, this proof is often conducted separately from the compiled code, either on paper or within a proof assistant environment, resulting in a gap between theory and practice. This separation occurs largely due to two factors: programming languages are not typically designed for proof writing, and compilers, which have been found to contain errors, are not purposed to maintain code correctness. Therefore, software industry is not yet mature enough to satisfy high requirements for electronic election.

1.3 Cryptography

The use of cryptography in electronic voting systems introduces an additional source of complexity, thereby increasing the potential for errors. The cryptographic techniques employed in these systems are based on sigma protocol technology and abstract algebra theories. Given their non-trivial nature, even a minor mistake could significantly compromise an election. There are well-documented cases, [3, 5, 4], where cryptographic errors have resulted in severe election mishaps. For example, a minor flaw could enable an election official to covertly alter votes, as demonstrated in a case involving the SwissVote election system [1]. The combination of complex mathematical sigma protocols and the inherent difficulty in ensuring software accuracy makes the task of implementing error-free electronic voting exceedingly challenging.

1.4 Problem

Electronic voting has the potential to enhance democratic elections, but the trustworthiness of electronic voting is currently in question due to challenges in ensuring its accuracy. This primarily stems from three factors: the high quality requirements for electronic elections, the complex nature of the underlying theories such as cryptography, and the current limitations of software development technology. As we strive towards this goal, it's important to define what we mean by "correctness" before we proceed.

Generally, the word "correctness" refers to adherence to specifications and requirements. However, each election is unique, with varying vote counting schemes, cryptography protocols, settings, and other properties, many of which are interdependent. Therefore, the specific definition of correctness will differ from one election to another, based on its unique set of required properties. Broadly, election correctness properties can be classified into three categories: privacy, security, and integrity. Privacy ensures the confidentiality of sensitive information, security safeguards information from unauthorized changes, and integrity guarantees accurate computations. Furthermore, these properties can be universally verifiable, meaning anyone can check them, individually verifiable, meaning only one person can check them due to privacy concerns, or not verifiable, if the election protocol is not designed to enable verification of these properties. Consequently, the assurance of an election's correctness is contingent upon the specifics of that particular election. In this work, we refer to "correctness" as an abstract property of an election protocol that is required by its specification and varies from one election to another. This property also depends on the position of the verifier, whether they are on the public or private side.

[1]https://www.unimelb.edu.au/newsroom/news/2019/march/researchers-find-trapdoor-in-swissvote-election-system

1.5 Related Work Overview

There are various angles from which to address the issue of ensuring the correctness of elections, and extensive efforts are being made to improve the situation on multiple dimensions and scales. In this work, we focus on the efforts that contribute to ensuring election correctness and identify four main areas of work in this regard. We will discuss these areas in descending order of breadth.

- Firstly, there is the implementation of sigma protocols in general. These protocols are commonly used in electronic elections, but their application extends beyond this context, making this a broader area of work.
- Secondly, a more specific area of focus is the design of election protocols and their verifiability. This involves mostly theoretical designing of protocols to meet specific requirements for election implementation.
- Thirdly, an even narrower field of work involves identifying and exposing errors and vulnerabilities in electronic elections. This can involve analyzing election protocols or software code for errors.
- Finally, the most specific area of work is the verification of deployed electronic election. This involves verifying that the particular conducted election has been done correctly, in accordance to requirements specified in the election protocol and that the errors did not occur in this election instance.

We discuss notable contributions in each of these areas, understand their value, relevance, and interconnections of these areas of work as well as position our own work in relation to them.

1.6 Proposed Solution

This study provides a technique for verifying a deployed electronic election and illustrates this approach by constructing a verifier for universally verifiable properties of an actual election. If the verifier itself is proven to be correct, it suffices to guarantee correctness of a conducted election in the verified properties, based on the "Software Independence" concept introduced by Rivest[6]. It is important to note that this implemented election doesn't necessarily need to be proven correct, and it's acceptable for it to be developed using imperfect methods. Additionally, we provide formally verified building blocks to construct an election verifier for other elections in a way that ensures correctness.

2 General Sigma Protocol Correctness

Sigma protocol ia a cryptographic primitives that have applications beyond elections. Several publications have contributed to the correctness of general sigma protocol implementation, thereby aiding in ensuring the correctness of electronic elections. This work began with seminal contribution of Barth et al.[7], who formalized sigma protocols using the Coq theorem prover. They formalised the fundamental definition of sigma protocol, and its properties: Completeness, Special Soundness, and Honest Verifier Zero-knowledge, introduced by Cramer et al. [8], providing formal machine-checked proofs. To prove the Honest Verifier Zero-knowledge property, they used a stronger property called "special Honest Verifier Zero-knowledge" and demonstrated that it implies the classic Honest Verifier Zero-knowledge. Additionally, they provided formalization for Conjunctive and Disjunctive combiners, which are used for constructions of more useful protocols, and proved the required properties about the combiners. As well as for some instances of sigma protocols such as Schnorr and Fiat-Shamir sigma protocol. This work offers formal machine-checked proof of sigma protocols, bridging the gap between theoretical concepts and proven implementation. However, despite its theoretical significance, it does not yield any proven executable program. In my view, while Barth's use of Coq is commendable, it lacks foresight, as the ultimate goal of the code is to be compiled and run, however, Coq's extraction mechanism is not formally verified correct. Thus the compilation does not transmit proven correctness to the operation of the program. It would be more justified to use HOL theorem prover, because proven correct code in HOL theorem prover can be compiled into a guaranteed correct executable using the verified CakeML compiler [9, 10].

Building on work of Barth et al.[7], Almeida et al. [11] developed a toolkit that generates an implementation of sigma protocol on demand along with the correctness proof. This toolkit can produce correctly implemented sigma protocol code in C language and a correctness certificate, given a required proof description. This approach is more efficient as it absolves users from any errors in the sigma protocol and delegates the complexity of sigma protocols implementation to professional cryptographers. Unlike Barth, Almeida used the Isabelle/HOL theorem prover to develop definitions and theorems, which is a better approach because from Isabelle/HOL environment executable program can be compiled by the verified compiler [12]. However, the generated correct code that Almeida's system produces is in C language, which does not formally guarantee correct execution of the compiled program. Therefore, it is the developer's responsibility to compile the code and ensure that the executable is operating correctly.

In a subsequent work, Almeida et al. [13] improved the sigma protocol compiler further. The updated compiler can now return a correct sigma protocol in either C or Java implementation. They expanded the functionality and optimized the implementation, but now proofs are implemented in the Coq theorem prover. While this work aims to delegate responsibility from the software developer to the cryptographer, it does not fully realize this goal. The compilation of Java or C code still leaves a correctness verification gap and retains partial responsibility on the developer. Compilers commonly have errors and Yang et al. [14] have demonstrated instances of compiler errors being exposed. While this level of guarantee might suffice for certain systems, it falls short for electronic elections.

Haines et al.[15] criticize Almeida's sigma protocol toolkit for being excessively specific and only suitable for a few special cases of sigma protocols in electronic elections. However, I believe that the main reason it is not suitable for developing of an election verifier is more fundamental: regardless of the functionality (specific or generic) and available range of sigma protocols (wide or narrow), Almeida's toolkit cannot conceptually guarantee the correctness of an executable, which is required for election verification if we want to rely on "Software Independence" concept [6]. Nevertheless, Almeida's work represents a significant advancement in the development of correct sigma protocols.

3 Election Verifiability

This line of research is closer related to our current project than the previous one, as it aims to enhance the theoretical definition of electronic election properties and contributes to the verifiability of elections. Specifically, these works provide a potential target for election verifiers, such as the one we are developing techniques for.

Haines et al. [16]. contributed to the field of electronic voting by working on the election protocol for the Schulze Voting Scheme. This scheme is a preferential voting system that determines the outcome of an election by calculating a "path strength" between each pair of candidates. The path strength represents the strength of the majority preference between two candidates, and the method considers all possible paths between each pair of candidates to choose the strongest path as the basis for comparison. This method satisfies a number of desirable voting method criteria making it robust. However, the Schulze method may fail the ballot privacy property when the number of candidates is high, which can lead to coercion and bribery.

In electronic voting, coercion refers to situations where voters are unduly influenced or forced to vote a certain way by a third party. This can involve threats, bribery, or intimidation. A robust electronic voting system should be coercion-resistant, ensuring that even under coercion, the secrecy of a voter's choice is maintained, preventing the coercer from verifying the actual vote.

The proposed election protocol designed to preserve ballot privacy while keep tallying universally verifiable. The team used Zero Knowledge Proof and homomorphic tallying to provide the above properties. The protocol involves two phases: homomorphic calculation of the encrypted matrix and determination of winners based on the decrypted matrix. The calculations result an evidence that ensures the correctness of the matrix and decide the winners. Ballots are represented as matrices, where +1 represents preference for one candidate over another, -1 represents preference for the other candidate over the first, and 0 represents equal preference. Each ballot is verified to be valid, and shuffled by a secret permutation, permutations of ballots are used to provide evidence of the validity of encrypted ballots. Zero-knowledge proofs are used to provide evidence of the correctness of shuffles and correctness of decryption. The certificate includes evidence of correct counting, the updated margin matrix, and winners with evidence to support the claim. This work develops an election protocol that provides universally verifiable tallying and preserves ballot privacy.

The authors recognized some limitations in their work, such as the use of unverified cryptographic primitive components from an external library and the extraction of code for compilation using an unverified Coq extractor module. Despite these limitations, we appreciate this work as a theoretical advancement of an election protocol with a universally verifiable tallying property, which could serve as a target for a verifier akin to ours. The fact that the tools are unverified doesn't pose an issue for us, as it aligns with our approach. We can refer to the "Software Independence" concept [6] and employ our proven correct verifier to ensure the correctness of such an election.

The subsequent contribution to the development of the election protocol pertains to improvements applied to the Helios voting system. This work is of interest to us because the properties of the election protocol provide a target for external verification, which is the primary focus of our study. Furthermore, we have utilized the Helios voting system [17] as our test subject to examine and demonstrate the functionality of our developed technique. Therefore, we are particularly interested in the properties of the Helios election protocol and its enhancements.

Kulyk et al.[18] proposed enhancements to the Helios [17] system by amending such properties as "participation privacy" and "eligibility verifiability". Participation privacy property enables an election auditor to confirm that information about voter participation was kept confidential. Eligibility verifiability lets an auditor to confirm that the tally counted only the ballots from the voters who are eligible. However, maintaining these properties simultaneously presents a challenge. Building on this work, Bernhard et al.[19] formally defined and proved these additional properties, enabling them to coexist. Verifying these properties for a real election necessitates the development of verification tools, which should themselves be verified to ensure reliability. These verifiers could be a potential area of our research or similar work, and our developed elementary components could potentially be used to construct a verifier for these properties. However, this is not the focus of our current proposal as we aim to verify different properties.

4 Challenging Elections

Electronic elections can be prone to a variety of errors, which is why election verification is so important. The discovery of these errors highlights the need for stronger election guarantees, which can be achieved through verification. Switzerland is a strong democracy and has a long history of electronic election adoption; however, even in Swiss government elections, multiple errors have been discovered. In the following papers, errors were discovered in SwissPost voting and election verifier.

SwissPost Voting system was disclosed for public review and a number of vulnerabilities were discovered, which are detailed in the report by Haines et al. [4]. One such vulnerability was the system's failure to verify that signatures came from the expected party. This could have allowed integrity attacks by spoofing the ballots of honest election participants. While Swiss Post has announced plans to address the issue, it has not been fully resolved yet. The report recommends checking the identity and key usage during signature verification and ensuring secure initialization of root certificates. Additionally, future versions of the system might want to eliminate certificate chains entirely to avoid the need to trust any root authority.

One of the vulnerabilities in SwissPost voting system lies in the management of discrepancies during the vote confirmation stage. In particular, the logs of various Confirmation Code Returners (CCRs) might not align, providing an opportunity for a attacker to tamper with the system. This weakness can be leveraged in two ways. First, the attacker could provide the valid Vote Cast Return Code to the voter while generating a vote transcript that results in the voter's vote being discarded. Alternatively, the attacker could generate a vote encryption that results in the acceptance of such vote that was made up without using ballot casting key. To address this issue, the authors suggests that the protocol specification should clearly detail how to handle such discrepancies. Additionally, the proof of security have to illustrate this approach to be compatible with the system. The report offers several examples of possible discrepancies and their potential exploitation. It concludes that the existing security proof does not adequately address scenarios like these and advocates for more rigorous verification standards.

The report highlights a possible weak point in the voting system, associated with the absence of Zero-Knowledge (ZK) proofs and accurate key creation. This weak point could risk the system's privacy, as a few parties might be privy to the secret key, which ideally should have been created in a distributed fashion. The document explains a potential attack scenario such that three parties are simultaneously compromised: voting server, the election board, and one of the online Confirmation Code Makers (CCMs) and are controlled by attacker. The attacker has the ability to alter the public key shares and generate a share that neutralizes the inputs of the trustworthy CCMs. This enables the attacker to gain knowledge of the secret key used for vote encryption, thus breaching privacy. However, the report recognises that this attack is not feasible in the the current security setting of the protocol due to certain protective measures. These include the belief that some members of the electoral board are immune to corruption and that auditors only allow the member of the electoral board to disclose their secret key to the CCM which is offline, following a successful encryption vote mix.

Despite the fact that the SwissPost voting system was supposed to be transparent and verifiable, it demonstrated errors and vulnerabilities. However, using the concept of "Software Independence" [6], a conducted election can still be verified as correct, given the correct verifier and assuming that the errors did not occur or the vulnerabilities were not exploited during election. To implement this, the Swiss government introduced an Election Verifier. The following work of Haines et. al. [20] analyzes this verifier and finds that it was also seriously flawed, to the extent that it could verify a fraudulent election as correct without recognizing it. Thus, the verifier did not help to ensure the election's integrity but only gave false hope. The election could pass verification even when the votes were manipulated, as the verifier was able to accept fake proofs. The essence of the problem is that the election protocol left a trapdoor for a malicious election administrator to replace the votes during the stage of shuffling the votes [4] and the proposed verifier was not able to detect it.

The e-voting system of SwissPost contains a notable error in the way it applies the Fiat-Shamir heuristic. This error enables a dishonest authority to interfere with the decryption process and create a proof of correct decryption that, while passing verification, declares an incorrect plaintext. The error arises because the proof of correct decryption of a ciphertext doesn't hash the ciphertext. Consequently, a dishonest prover can calculate a valid proof and then select a statement based on that proof. This undermines the validity of the proof and can be taken advantage of by a malicious authority, such as the system's CCM1, to alter specific votes during the decryption process and create decryption proofs that can't be distinguished from legitimate ones, thus passing verification.

The aforementioned security flaw has a couple of constraints. First, to fabricate a decryption proof and successfully complete a shuffle proof, the adversary must have knowledge of the random values used in the encryption of the votes they aim to alter. This could be accomplished either by compromising a voting client or by taking advantage of a weak random number generator. Second, while the malicious entity can't arbitrarily declare a false plaintext and still have the shuffle proof function, they can still demonstrate that a ciphertext decrypts to something other than the actual value, resulting in an output vote that's more likely to be gibberish than a legitimate vote. This manipulation could be used to strategically invalidate votes that the adversary disagrees with, potentially swaying the political balance in their favor.

The verification process is flawed due to its reliance on a premise that has been proven incorrect. As a result, its successful execution doesn't necessarily guarantee accuracy. Although the exploit is likely to leave traces of irregularities, identifying the root cause of the issue could be challenging without infringing on the privacy of certain votes, especially if the identified weakness in the Fiat-Shamir transform is not recognized. To address this, it's recommended that all pertinent data be included in the hash during the application of the Fiat-Shamir heuristic. Currently, steps are being taken to rectify this issue in forthcoming iterations of the system.

The above works highlight the serious consequences that errors in an election verifier can cause. If the developers had used proof-based software development or our proposed technique, such errors might have been avoided. This work underscores the need for accessible techniques for building proven correct election verification tools, such as the one we propose.

5 Correctness of Electronic Election

One of the most important aspects of any election is ensuring the correctness of the results. Even in cases where the winner is determined by a simple majority, it is crucial to verify the accuracy of the vote counting process. This stream of work aligns closely with our current project and focuses specifically on election verification.

Pattison et al. [21] have emphasized the importance of formal verification in ensuring accurate election results. Their research specifically focuses on the formal verification of final election results.

Pattison et al. [21] developed a verified correct election result verifier for majority elections. Their work relied on the concept of "Software Independence" [6] and used the Coq Theorem Prover to extract code and produce an executable program. However, their work is limited in scope as it only verifies tallying of votes of plaintext elections and did not encompass other universally verifiable election integrity properties, This leaves a gap which we will try to cover. Additionally, they used the Coq theorem prover, whose extraction mechanism is not formally verified. Despite these limitations, their work serves as a significant example and prototype for formal election verification.

Another noteworthy contribution to election result verification is the work of Ghale et al. [22], who verified a more complex election type known as the Single Transferable Vote (STV) scheme. While Pattison et al.'s work focused on verifying the tallying of votes for plaintext majority elections, Ghale et al.'s work extends this to a more complex election scheme. Their work demonstrates the feasibility of formal verification in complex elections and provides a valuable contribution to the field of election verification.

The Single Transferable Vote (STV) is a voting technique, intended for multi-seat elections, aiming to achieve proportional representation. In an STV election, voters order candidates based on their preferences. During the tallying phase, if a candidate receives votes exceeding the quota needed to win a seat, the surplus of the votes are given to the other candidates according to the voters' secondary preferences. If no one of the candidates can reach the quota, then the candidate who had received the lowest number of votes is removed, and their votes are reassigned according to the voters' subsequent preferences. This procedure is repeated until all seats are filled, ensuring that the greatest number of votes have an impact on the final election result.

Given that the Single Transferable Vote (STV) type of election involves a more complex tallying process, the verification of vote counting becomes even more crucial than in majority elections. However, the election protocol does not incorporate cryptography, which considerably simplifies the computation. Ghale et al. [22] developed a framework for the verification of election vote counting, utilizing the verified proof assistant HOL and the verified compiler CakeML. These tools enable the creation of an end-to-end correct election verifier [23, 24]. They achieved total correctness and developed an end-to-end proven correct verifier for electronic elections. Like the Pattison team, they worked with plaintext election data and verified only vote tallying, not other verifiable properties. Furthermore, Ghale et al.[22] conducted a trial and verified a real election of substantial size, which increases confidence in the practical usability of formally verified software built using HOL and CakeML. Their work serves as a standalone proof of concept and an excellent prototype of an end-to-end proven correct election verification tool.

Next, we examine the work of Haines et al. [15], who developed an election verifier for encrypted elections. This work comes close to achieving end-to-end correctness in election verification and, as they state, "significantly reduced the gap" between correct theory and operational program. Their work includes the development of formal definitions of the elementary components of the verifier and the formal proof of its correctness. The program they developed was capable of verifying the universally verifiable integrity properties of a real election and the proofs of well-formed ballot. Positive aspects of their work is that they used the same code to prove correctness about and compile the executable and worked with encrypted election.

However, a drawback of their work is that the formulation of the Honest Verifier Zero Knowledge theorem does not align with its intended description. While the theorem was intended to establish a bijection between the transcript space and randomness, the mathematical formulation (3.1) of the theorem does not constitute this bijection. Specifically, the authors state: “We define honest verifier zero knowledge in a concrete way without referring to probabilities; we show that there exists a bijection between the transcripts generated by taking the random coin from the commit in P0 and by taking the response at random in the simulation. In addition we require the challenge space to be an abelian group, the algorithms to output the transcript they receive without change, that algorithm V0 outputs the challenge from its randomness tape without modification, and that the simulator produces accepting transcripts on all inputs” [15]. This discrepancy was corrected in their next work.

Mathematical formulation differs from the claim and states precisely as follows, direct citation [15]:

$$\begin{aligned}&\text{“}\forall s \in S,\ w \in W,\ r \in R,\ e \in E,\\ &\mathrm{Rel}(s,w) = \mathrm{true} \Rightarrow\\ &P_1(V0(P0(s,r,w),e),r,w) = \mathrm{simulator}(s, \mathrm{simMap}(s,r,e,w),e)\\ &\wedge \forall t \in T, \exists r \in R \text{ s.t. } t = \mathrm{simMap}(s,r,e,w)\text{”}\end{aligned} \tag{3.1}$$

Bijection requires:

$$f(t) = r \quad \text{and} \quad f^{-1}(r) = t \tag{3.2}$$

Here we can see a map from randomness space to response space [15]:

$$\text{“}t = \mathrm{simMap}(s,r,e,w)\text{”} \tag{3.3}$$

Here we do not see, the map from response space to randomness space:

$$r = \mathrm{simMap}^{(-1)}(s,t,e,w). \tag{3.4}$$

Furthermore, the researchers employed the Coq proof assistant for their proofs. However, the extraction mechanism of Coq does not guarantee the correct operation of the resulting program according to the proved properties. As a result, the final executable verifier does not come with an absolute guarantee of correctness. This situation leaves a minor gap in software correctness, a gap that our current work aims to address and potentially fill.

This work addresses the discrepancy in the bijection identified in the previous study[15]. It is noteworthy for its creation of verified logical components that are crucial in developing elections and verifiers[25]. The work provides logical components and proves their correctness. These components are intended for use by election developers to prevent critical errors in elections. The work is primarily relevant to the cryptographic system for electronic elections called mixnets, introduced by Chaum[26]. However, these components are also usable building blocks for constructing election verifiers or implementing election protocols for other electronic elections. One positive aspect of this work is that all essential building blocks are clearly defined and proven correct according to the definition of correct sigma protocol given by [8]. One downside is that this logical machinery is again developed using Coq, which, as mentioned earlier, does not have a proven correct extraction mechanism. Nonetheless, this work provides valuable material and inspiration for present work. We build upon the developed formulations and compile them using a verified compiler, thereby addressing the issue of the correctness gap.

The corrected version of with Honest Verifier Zero Knowledge theorem formulation with bijection in [25]. Direct citation:

$$\begin{aligned}&\text{“honest\_verifier\_ZK} : \forall s \in S, w \in W, e \in E, r \in R, t \in T,\\ &(\mathrm{V}_1(\mathrm{P}_1(\mathrm{V}_0(\mathrm{P}_0\ s\ r\ w)e)rw) = \text{true}\\ &\to (\mathrm{P}_1(\mathrm{V}_0(\mathrm{P}_0\ s\ r\ w)e)rw) = \text{simulator}\ s\ (\text{simMap}\ s\ w\ e\ r)\ e) \wedge\\ &\text{simMapInv}\ s\ w\ e\ (\text{simMap}\ s\ w\ e\ r) = r \wedge\\ &\text{simMap}\ s\ w\ e\ (\text{simMapInv}\ s\ w\ e\ t) = t\text{”}\end{aligned} \tag{3.5}$$

Which does establish a bijection.

6 Summary

Our research is contained within the rapidly developing field of electronic elections and their verification. We have reviewed several papers on topics such as sigma protocol correctness, election protocol design and verifiability, the exposure of errors in electronic elections, and election verification efforts. These studies provide a comprehensive overview of the current state of the industry, highlighting the significant interest and effort invested in achieving reliable and guaranteed correct elections.

The area of election protocol design directly connect to our work. Improvements in election protocol contribute to the definitions of properties that can be verified using a correct verifier. While our current focus is on verifying the universally verifiable integrity property, the defined properties for other election protocols provide avenues for future research. For instance, security or privacy properties could also be verified using a correct verifier.

The importance of our work is underscored by studies exposing errors in electronic elections and verifiers. These demonstrations of vulnerabilities and their consequences underscore the need for a proven correct verifier to guarantee election integrity. Furthermore, work on formalisation reveals the broader applications of sigma protocols beyond elections.

The works of Haines and Ghale [15, 22] identifies a gap in the existing literature. Haines worked with encrypted election verification using unverified tools and Ghale worked on plaintext election verification with verified tools. There is a lack of work on the verification of encrypted elections using verified tools. We intend to build upon their findings and cover the gap, by providing verification of encrypted elections using verified tools. However, our research does not merely focus on verifying a single election; instead, we propose a technique along with reusable elementary components for building a proven correct verifier for other encrypted elections.

References

[1] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975. [Cited on page 1.]

[2] Nicholas Chang-Fong and Aleksander Essex. The cloudier side of cryptographic end-to-end verifiable voting: a security analysis of helios. *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016. [Cited on page 1.]

[3] J. Alex Halderman and Vanessa Teague. The new south wales ivote system: Security failures and verification flaws in a live online election. In *International Conference on E-Voting and Identity*, 2015. [Cited on pages 1 and 2.]

[4] Thomas Haines, Olivier Pereira, and Vanessa J. Teague. Running the race: A swiss voting story. In *International Joint Conference on Electronic Voting*, 2022. [Cited on pages 1, 2, 6, and 7.]

[5] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security analysis of the estonian internet voting system. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014. [Cited on page 2.]

[6] Ronald L. Rivest. On the notion of 'software independence' in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366:3759 – 3767, 2008. [Cited on pages 3, 4, 5, 7, and 8.]

[7] Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin, Benjamin Grégoire, and Sylvain Heraud. A machine-checked formalization of sigma-protocols. *2010 23rd IEEE Computer Security Foundations Symposium*, pages 246–260, 2010. [Cited on pages 3 and 4.]

[8] Ronald Cramer. Modular design of secure yet practical cryptographic protocols. 1997. [Cited on pages 3 and 9.]

[9] Yong Kiam Tan, Magnus O. Myreen, Ramana Kumar, Anthony C. J. Fox, Scott Owens, and Michael Norrish. A new verified compiler backend for cakeml. *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016. [Cited on page 4.]

[10] Ramana Kumar, Magnus O. Myreen, Scott Owens, and Yong Kiam Tan. Proof-grounded bootstrapping of a verified compiler producing a verified read-eval-print loop for cakeml. 2015. [Cited on page 4.]

[11] José Bacelar Almeida, Endre Bangerter, Manuel Barbosa, Stephan Krenn, Ahmad-Reza Sadeghi, and T. Schneider. A certifying compiler for zero-knowledge proofs of knowledge based on sigma-protocols. In *IACR Cryptology ePrint Archive*, 2010. [Cited on page 4.]

[12] Lars Hupel and Tobias Nipkow. A verified compiler from isabelle/hol to cakeml. In *European Symposium on Programming*, 2018. [Cited on page 4.]

[13] 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. *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012. [Cited on page 4.]

[14] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in c compilers. In *ACM-SIGPLAN Symposium on Programming Language Design and Implementation*, 2011. [Cited on page 4.]

[15] Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019. [Cited on pages 4, 8, 9, and 10.]

[16] Thomas Haines, Dirk Pattinson, and Mukesh Tiwari. Verifiable homomorphic tallying for the schulze vote counting scheme. In *Verified Software: Theories, Tools, Experiments*, 2019. [Cited on page 4.]

[17] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, 2008. [Cited on page 5.]

[18] Oksana Kulyk, Vanessa Teague, and Melanie Volkamer. Extending helios towards private eligibility verifiability, 2015. [Cited on page 5.]

[19] David Bernhard, Oksana Kulyk, and Melanie Volkamer. Security proofs for participation privacy , receipt-freeness , ballot privacy , and verifiability against malicious bulletin board for the helios voting scheme, 2017. [Cited on page 5.]

[20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. *2020 IEEE Symposium on Security and Privacy (SP)*, pages 644–660, 2020. [Cited on page 7.]

[21] Dirk Pattinson and Carsten Schürmann. Vote counting as mathematical proof. In *Australasian Conference on Artificial Intelligence*, 2015. [Cited on page 8.]

[22] Milad K. Ghale, Dirk Pattinson, Ramana Kumar, and Michael Norrish. Verified certificate checking for counting votes. In *Verified Software: Theories, Tools, Experiments*, 2018. [Cited on pages 8 and 10.]

[23] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. Hol with definitions: Semantics, soundness, and a verified implementation. In *International Conference on Interactive Theorem Proving*, 2014. [Cited on page 8.]

[24] Yong Kiam Tan, Magnus O. Myreen, Ramana Kumar, Anthony C. J. Fox, Scott Owens, and Michael Norrish. The verified cakeml compiler backend. *Journal of Functional Programming*, 29, 2019. [Cited on page 8.]

[25] Thomas Haines, Rajeev Goré, and Bhavesh Sharma. Did you mix me? formally verifying verifiable mix nets in electronic voting. *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1748–1765, 2021. [Cited on pages 9 and 10.]

[26] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *CACM*, 1981. [Cited on page 9.]