Research proposal

Australian National University
COMP 2550/4450/6445 Advanced Computing Research Methods
Assignment 1
Student name: Ratmir Mugattarov
uni id: u6650903

Project title

Proven correct election verification program

1. Introduction

1.1 Aim

The objective of this project is to enhance electronic elections by demonstrating a technique for developing an end-to-end verified computer program that can perform verification of the election results.

1.2. Real world problem

Traditional election

Elections are essential for democracy, and to ensure public trust in the democratic process, it is crucial that elections are conducted with transparency and integrity. However, many modern elections still rely on paper-based ballots and manual tallying, which can be labor-intensive and limit accessibility to the auditing process, particularly in large-scale elections. For example, it is impossible to verify by hand counting an election with several thousands ballots. Electronic voting systems offer the benefits of automation, accelerating the election process, and providing easier access to auditing.

Electronic election

To ensure privacy, electronic voting systems often use cryptography techniques, which involve complex mathematics and increase the likelihood of errors. Given the importance of elections, such errors are unacceptable and can erode public trust. Therefore, it is critical to ensure the correctness and integrity of electronic voting systems to uphold the credibility of the election process. Unfortunately, electronic elections, like many other software, often contain errors.

Correctness gap

Traditionally, there is a gap between the correctness of the underlying concept and that of the operating program. While correctness proof is often done on paper or in proof assistant environments, the actual code being executed is not directly proven to be correct. Demonstrating correct operation is typically achieved by testing, but this approach is inadequate for elections, as tests cannot cover all scenarios.

There are programming environments that allow for the correctness of compiled code to be proven. While this may appear to be a solution for electronic elections, the proof environment and compiler are computer programs themselves and may not necessarily be end-to-end correct, leaving the possibility of incorrect proofs being accepted and errors being present in the program that is being proved. Therefore, even if the program code has been formally proven to be correct and compiled from the same source code, it does not guarantee correct operation.

Solution

To advance the current state of electronic elections, our proposed work aims to develop a guaranteed correct election verifier. This will demonstrate a technique to bridge the gap in correctness applied to cryptography software. We will select a proof environment and compiler that have been formally proven correct using a self-verification technique, to only accept correct proofs and produce only correct executable, respectively. This approach will yield an executable election verifier with the guaranteed absence of errors. To test the performance and effectiveness of our technique, we will verify a real election.

1.3 Background

In this section, we will provide a comprehensive description of the key items that will be referenced throughout this research proposal.

Proof-based software development

Proof-based software development is a technique applied in correctness-critical fields such as healthcare or cryptography. This technique employs mathematical proofs about the program code to ensure correct operation. Unlike test-driven development, proof-based techniques provide a much stronger guarantee of correct operation as tests can only demonstrate the presence of errors, whereas proof-based techniques can demonstrate the absence of certain errors.

HOL

HOL is an interactive environment that is specifically designed to assist in proving theorems with a high level of automation. It is equipped with extensive standard libraries of basic mathematical theorems, for example, Arithmetic theory, Group theory. HOL provides proof tactics, for example, proof by induction and proof by contradiction. This allows partially automate proofs, making the process of theorem proving easier. Through complex self-verification technique [1] [2] [3] HOL is proven to be able to accept only valid proofs, which eliminates the possibility of errors in the proof process. In addition to its powerful theorem proving capabilities, HOL also includes the cakeML compiler, which is a function that is defined within the HOL environment. The cakeML compiler has also been proven to be correct within HOL [4].

CakeML

CakeML is a programming language that includes a formally verified compiler, which means that the produced executable program behaves exactly as intended by source code. CakeML is closely related to HOL. This means that functions defined and proven correct in HOL can be compiled by the CakeML compiler. This connection allows to preserve proven properties of the function over compilation, effectively bridging the gap in correctness (described in Introduction Section).

Helios Voting

The Helios Voting is an open-source voting platform that offers secure and anonymous electronic voting. Helios employs cryptographic protocols to conduct elections. Voters' choices are encrypted and transmitted to the Helios server, where encrypted ballots are tallied and the election results are decrypted. Importantly, individual choices never get decrypted, and it is computationally infeasible to determine individual voters' choices or identities. Additionally, Helios rejects invalid voter choices. However, it is important to note that all the above properties are guaranteed only if cryptography is correctly implemented, which currently remains not proven.

Helios provides election audit functionality for users to verify the election results. This verification process requires complex computations that cannot be checked manually. However, it's important to note that the election implementation and verification tool are not formally verified, which means they may contain errors. Unfortunately, there have been numerous instances where critical errors were discovered in Helios elections.

2. Research Problem

2.1. Particular problem

Electronic elections are not commonly implemented with a proven, error-free approach. This requires huge trust in order to use them for any real-life election. Errors in electronic election implementation are widespread and well-documented, as multiple papers have highlighted significant flaws in popular electronic election systems

The reason for such state is that proof based software development techniques are not yet mature. Developing code along with its correctness proof significantly increases the effort and cost involved. Furthermore, most of the programing languages were not designed for proofs, and their compilers were not intended to preserve correctness. These challenges make it difficult to build an electronic election system that is guaranteed to be correct.

For an election to be considered trustworthy, it is not enough for it to be correct, but it must also be verifiable by an independent auditor. This verification process is carried out using computer programs. However, to have confidence in the results, it is crucial that these verification programs are proven to be correct. Unfortunately, as described above, this is not always the case due to limitations in current techniques.

Our project aims to address this issue by providing a technique for developing a verified and proven election verification program (using HOL and CakeML). By implementing an election verifies using this technique, we can ensure that the election results are accurate and reliable, giving the public confidence in the democratic process.

2.2. Scope

Here we narrow down the scope of the project from a broad election improvement to a particular plan. There are numerous variations of electronic elections that utilize different types of questions, cryptographic protocols, and produce varying publicly available data. To demonstrate our technique, we select one specific electronic election to serve as the target for our election verifier. It is important to note that our developed verifier will not be immediately capable of verifying any other electronic elections. However, our purpose is to demonstrate useful techniques and not to attempt to prove the validity of all electronic elections.

Target

We select Helios Voting for several reasons. Firstly, it is an open source platform, which allows us to assess the original code. Secondly, it has been subjected to extensive scientific research, with many errors having been revealed [5] [6] [7] and fixed as well as suggested improvement implemented. Because of these changes, Helios is one of the most evolving electronic election platform. In addition, the International Association for Cryptologic Research (IACR) uses Helios Voting for its director elections, providing us with confidence in its security and reliability.

Included

For the purposes of this project, we clarify the definition of election verification. We define it as the property of universal verifiability of an election, as described in the literature on electronic voting. To demonstrate that a system is universally verifiable, three critical verifications must be obtained:

1. Verification that the individual voters' choices have been encrypted correctly.
2. Verification that voters' choices have been collected correctly.
3. Verification that the results have been computed correctly using the individual voters' choices.

Helios Voting System publishes cryptographic data that can be used to verify the election. Verification is based on the checking the cryptographic evidence produced by any Helios and election results published by the election authority. If the evidence passes the three checks mentioned above, it guarantees that the published results are correct.

Exculed

We would like to clarify what falls outside the scope of our election verification process. While there are many potential issues that can arise in electronic elections, we do not aim to control all of them.

In our definition, election verification does not include:

- Security and privacy checks
- Verification of the correctness of Helios code
- Detection of ballot stuffing [8] or other fraudulent activities

2.3. Contribution

The aim of this project is to demonstrate the existence of an efficient software development technique that can be used to build end-to-end correct electronic election verifiers. The successful completion of this project is expected to yield the following contributions:

1. Our work aims to address the gap in the knowledge that was left by Haines [9]. Specifically, we plan to advance present develop a computer program for election verification that is end-to-end verified.

2. Development Technique: In this project, we will present a detailed explanation of the step-by-step approach for building an election verifier with proven correct operation. This technique will bring a substantial improvement over the current common practice in electronic voting, where correctness is usually ensured by multiple layers of specifications and not by proofs of the code. Such an approach does not guarantee the correctness of the computer program. Adopting our proof-based development technique for building an election verifier guarantees the correctness of the program, leaving no room for errors.

3. A set of Generic Elementary Components. The proposed set will consist of proven correct building blocks that are useful for constructing cryptographic protocols. These protocols are widely used for electronic elections. By using these predefined correct components, software developers working on electronic elections can save time and effort on developing proofs.

4. Demonstration: As part of our software development process, we will be conducting a demonstration trial using a real election to showcase the performance and effectiveness of the software. This trial will not only contribute to the previous phases of development but

will also provide more confidence in the software's capabilities. The demonstration trial will also serve to confirm whether the election is correct or not, depending on the outcome.

2.4. Advance

Our work advances the current state of human knowledge by bridging the gap in correctness of election verifier for electronic elections involving cryptography protocols.

2.5. Related work

Electronic voting systems are constantly evolving thanks to scientific research, with Helios voting being one of the most popular platforms at the forefront of these efforts. In this section, we will explore work related to the Helios voting system.

Types of election improvement

The proposed research aims to improve election via improving election verifiability, which refers to the ability of election auditors to ensure that the election has been conducted correctly. Efforts to achieve this goal can be divided into two major streams.

1. Design of election protocls: this work aims to develop clear definitions and proofs for election protocols and cover all necessary properties to ensure correct election conduct.
2. Developmenmt election verification tools: As verification process involves the use of computer programs, it is crucial to ensure that the program produces accurate results. For instance, the verification program must not falsely claim that an election was conducted properly when it was not, and vice versa.

Design of election

The second stream of work is closely related to our proposed research, while the first stream work is indirectly related to our proposal. First stream work provides a target for the second stream's work. Kulyk et al. (2015) [10] proposed ways to improve Helios by including additional election properties, such as participation privacy and eligibility verifiability. Participation privacy property allows the election auditor to verify that information about the either a particular voter participated in the election or not was kept private during the election. Eligibility verifiability property allows the auditor to verify that the ballots from eligible voters only were tallied to obtain the result. Continuing the above work, Bernhard et al. (2017) [11] formally defined and proved the above additional properties, allowing them to hold simultaneously. The verification of the above properties for a real election requires verification tools to be developed. To ensure the reliability of the tools, they should be verified themselves. Such verifiers could be an area of our research. However, it is not the target for the current proposal as we aim to verify different properties.

Verification of election

In related work, Ghale et al. (2018) [12] developed a verified election verifier for electronic elections. They achieved total correctness and developed an end-to-end proven correct verifier for electronic elections. However, they only targeted the correctness of vote tallying and excluded the other two components of universal verifiability. They developed an election protocol definition and proved its correctness, and then compiled it with CakeML. Furthermore, the election verifier underwent trial testing and was used to verify a real election. Although the election protocol does not include cryptography, which significantly simplifies the computation involved, this work can serve as an isolated proof of concept and a great prototype of an end-to-end proven correct election verification tool.

Plaintext election verifier

Ghale et al. (2018) [12] have developed a verified election verifier for electronic election. The great thing is that they achieved total correctness and developed end-to-end proven correct verifier for electronic election. They only targeted correctness of votes tallying and ignored the other two components of universal verifiability. They developed election protocol definition, proved its correctness and then compiled with CakeML. Even though the election protocol does not include cryptography, which significantly simplifying computation involved, this work can serve as isolated proof of concept and great prototype of end-to-end proven correct election verification tool. Our proposal aims to improve upon this work by incorporating cryptography and focusing on all three components of universal verifiability.

Encrypted election verifier

Haines et al. (2019) [9] developed a verified election verifier using the Coq proof assistant and extracted executables. This work represents a significant step forward in improving electronic elections and in proof-based software development, and provides motivation for the current proposal.

However, the compiling module in Coq, which produces executables from the proven correct function, is not verified itself. This means that it is not guaranteed that the executable verifier operates precisely as it is programmed. Therefore, the correctness gap has been reduced but not completely bridged. Nonetheless, Haines' work provides a foundation for the current proposal to fix this limitation and implement the election verifier using a proven correct environment and compiler, such as HOL and CakeML. Their "future work" section even mentioned implementing such verifier using CakeML, which inspired the current proposal. Essentially, the current proposal aims to address one of the limitations of Haines' work.

Summary

The development of reliable election verification tools is crucial to ensuring correctness and fairness of elections. The propopsed project aims to develop tools that can accurately verify the properties of an election and ensure their own reliability. The work similar to Kulyk et al. and Bernhard et al. provides a target for such research, while the work of Ghale et al. and Haines et al. serves as promising prototypes of an proven correct election verification tool.

3. Methodology

Our work builds upon the work of Haines et al. (2019) [9], and we plan to follow a similar methodology. However, due to our use of two programming environments (HOL and cakeML), as opposed to one (Coq) in Haines' work, we will take shallower steps and use auxiliary staging to secure intermediate results. Additionally, HOL has a more strict typing system in comparison to Coq, which may increase complexity. To gain confidence in our approach, we will resort to prototyping stages.

Make a dummy

To ensure that we can achieve our final goal of creating a proven-correct executable function compiled with CakeML compiler, we need to verify that it is feasible. Although CakeML is not yet a production programming language, it is still a work in progress and research object. Thus, we will define a dummy function in CakeML that performs some trivial operations such as adding two numbers together and printing the result. After defining our dummy function, we will compile and test it to ensure that it works as expected. We will then add a proof component to our function by defining it in a HOL environment and proving a trivial property

about it. Finally, we will compile the function using CakeML to obtain an executable. This process will give us confidence in working with CakeML and verifying the correctness of our code.

Final goal definition

Next, we will develop a definition of the function required for our election verification. To assist us, we will draw on the paper by Haines et al. [9], which uses the Helios election system to verify it using the Coq theorem prover and check for updates in Helios protocols. Initially, we will select a specific election, download the publicly available data, and explore Helios protocols. We will then derive a mathematical equation for the final function, which will take publicly available data as input and output true or false.

Prototype

Creating a prototype will give us confidence that we understand what we want to achieve before attempting to create a proven-correct function. We will implement an unverified prototype of the final election verification function in Python and test it on some primitive data. This process will help us refine our understanding of the problem and identify any potential issues before moving on to the next stage.

Building blocks

To construct the election verification function, we will follow the approach outlined by Haines et al. [[9]]. We will define simpler algebraic structures and elementary protocols to use these structures as building blocks for the final verification function. The outcome of this stage will be the election verification function defined in HOL using simple building blocks.

Proofs

To define the properties required for the election verification function, we will develop a set of theorems in HOL. We will then develop proofs of these theorems in HOL, and if necessary, define and prove auxiliary theories and lemmas. We will employ a recursive approach and attempt to transfer the required properties to the low-level components, aiming for proof simplification.

Executables

After developing the election verification function and proving the required correctness properties in HOL, we will use CakeML to compile the verification function. Since CakeML is a difficult programming language with a steep learning curve, we will attempt to minimize the amount of work done in CakeML. We would like to have all operations proven correct and located in HOL. To accomplish this, we will define a wrapper function in CakeML that imports the proven-correct election verification function from the package defined and proved in HOL. The wrapper function will only pass the input data to the actual computation function and print out the result. We will then compile the wrapper function and test it by feeding in the Helios election data and checking the result.

Time Estimation Table

Stage	Time estimation
Make a dummy	3 days
Final goal definition	5 days
Prototype	2 days
Building blocks	5 days
Proofs	15 days
Executables	5 days
Total	7 weeks

3.1. Anticipated challenges

- The proofs may require some cryptography theorems that are not present in standard HOL libraries, which could significantly extend the work needed to build those theorem bases.
- CakeML is not widely used as a programming language for software development, so there may be unexpected complications with the language type system or syntax.
- The data structure for election verification is provided in a format that needs to be converted into a mathematical object that CakeML can ingest without losing formal correctness. At this stage, it is unclear how to perform this conversion while preserving formal correctness. We plan to consult the CakeML community and seek advice on the best approach to this conversion.
- If our trial of an election receives a negative certification result, we will investigate the cause of the result. We expect that, in the worst case, we will find a lack of evidence that the election is correct, rather than evidence of an incorrect election result. Depending on the error, we will use appropriate techniques to trace it down.

4. Evaluation Criteria

Peer review

Evaluating the quality of proven correct programs can be challenging. The reason for this is that these programs are proven correct before they are compiled, meaning that they are correct by construction. Consequently, traditional tools for measuring software quality, such as testing and evaluations, are weaker than proof and fall short in providing evaluations.

As a result, the evaluation of such programs is conducted through peer review by Helios, HOL, CakeML communities, and other interested parties.

Demonstration

In order to demonstrate the operation of the resulting election verification program, we will conduct a trial election verification. Although this test does not prove anything, it can be used as a sanity check. The outcome of this trial will indicate the success of the project and the correctness of the election under scrutiny.

We plan to use the director election of IACR in 2022. We expect a positive result due to the reliability of the Helios voting system, which has been under review by professionals for a long time. In addition, we use proof-based software development, which guarantees correctness by construction.

Although tests do not prove correctness, they serve as a good sanity check to ensure that nothing has gone disastrously wrong.

Performance
In addition to other metrics, we are interested in evaluating the performance of the election verifier. Specifically, we want to measure the time and memory used to verify a real-size election with thousands of voters.

To achieve this, we will compile the election verifier program for different architectures and assess the metrics mentioned above. Potentially we can generate a simulated election data of the various sizes and measure time and memory usage depending on the election size.

The significance of this evaluation lies in the fact that the correctness of the verifier's code has already been established, and the compiled program will execute exactly the operations that we have verified. Often, it is the case that an optimized algorithm is challenging to prove correct, while a version that is easy to prove may not be computationally efficient. This is the reason we are interested in performance time, this may show the need to improve the definitions of the functions.

Attack
It might be worth attempting to purposefully break the election verifier. This task entails constructing election data that would be correctly verified, even though the election is obviously wrong. This is a challenging and consuming task that requires separate work.

Bibliography

[1] R. Kumar, "Self-compilation and self-verification," 2016.

[2] R. Kumar, R. Arthan, M. O. Myreen, and S. Owens, "Self-formalisation of higher-order logic," *J. Automated Reasoning*, vol. 56, pp. 221–259, 2016.

[3] R. Kumar, M. O. Myreen, S. Owens, and Y. K. Tan, "Proof-grounded bootstrapping of a verified compiler producing a verified read-eval-print loop for cakeml," 2015.

[4] Y. K. Tan, M. O. Myreen, et al., "A new verified compiler backend for cakeml," *Proc. 21st ACM SIGPLAN Int. Conf. Functional Program.*, 2016.

[5] D. Bernhard, O. Pereira, and B. Warinschi, "How not to prove yourself: pitfalls of the fiat-shamir heuristic and applications to helios," in *Int. Conf. Theory Application Cryptology Inf. Secur.*, 2012.

[6] N. Chang-Fong, and A. Essex, "The cloudier side of cryptographic end-to-end verifiable voting: a security analysis of helios," *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, 2016.

[7] D. Bernhard, V. Cortier, O. Pereira, B. Smyth, and B. Warinschi, "Adapting helios for provable ballot privacy," *IACR Cryptol. Eprint Arch.*, vol. 2016, p. 756, 2011.

[8] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène, "Election verifiability for helios under weaker trust assumptions," 2014.

[9] T. Haines, R. Goré, and M. Tiwari, "Verified verifiers for verifying elections," *Proc. 2019 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019.

[10] O. Kulyk, V. Teague, and M. Volkamer, "Extending helios towards private eligibility verifiability," 2015.

[11] D. Bernhard, O. Kulyk, and M. Volkamer, "Security proofs for participation privacy , receipt-freeness , ballot privacy , and verifiability against malicious bulletin board for the helios voting scheme," 2017.

[12] M. K. Ghale, D. Pattinson, R. Kumar, and M. Norrish, "Verified certificate checking for counting votes," in *Verified Software: Theories, Tools, Experiments*, 2018.