Research Statement

Mukesh Tiwari

August 18, 2022

My work aims to build **correct software programs used in democratic processes**, using the Coq theorem prover. I focus on formal verification of software programs used in elections, cryptographic primitives used in elections, and computational social choice theory. Various critical decisions, e.g., electing the winner of an election, are taken by governments based on the output produced by these software programs. However, if these software programs contain bugs, then they may produce a wrong output. As a result, it could hamper the trust of members of general public in the decision making of government. Worse yet, nowadays there is more and more automation of decision-making in various bodies of a government (democracies) based on software programs, and therefore it is more imperative than ever that we formally verify these software programs and develop software tools to automate the formal verification process to make the decision-making more trustworthy.

Fair elections are the only way to keep a democracy alive. Most countries (election commission) use paper ballots to record its citizens' votes (preferences), and then election commission officials produce the final tally from these votes, while many scrutineers –workers of the participating political parties and members of general public in some countries, e.g., France– observe the counting process. These scrutineers and paper ballots, both, are crucial to ensure that the election is free from fraud. Scrutineers keep the counting process transparent by making sure all the vote counting rules are followed, while paper ballots ensure verifiability because in case of a dispute these paper ballots can be counted again. Many countries, however, in recent years are introducing computers to conduct some part, or all, of election processes because it is cost effective, accessible to disables voters, faster result, convenient, etc. In addition, some voting methods are very complex to count by hand, e.g., single transferable vote, used in Australian senate elections, and it may take months to declare the final result if counted by hand. Therefore, the Australian election commission scans all the ballots of senate election and uses a software program to produce the final tally. In addition, several cantons in Switzerland use an electronic voting system developed by Swiss Post, India uses electronic voting machines, various municipalities in France use electronic voting machines, etc.

1 PhD Work

During my PhD, my research was focused on verifying electronic voting, specifically vote-counting schemes, in the Coq theorem prover and bringing three important ingredients, correctness, privacy, and (universal) verifiability, of paper ballot election to electronic setting (electronic voting). In a paper ballot election, correctness and verifiability are assured by scrutineers. Moreover, privacy in a paper ballot election comes for free because of the secret ballot, introduced by Australia in

1855. However, achieving these desirable properties –correctness, privacy, and verifiability– can be difficult in electronic voting because software programs, used in various stages of an election, work in a opaque (blackbox) manner. In this (electronic) setting, a voter cast their ballot in an election and the software program produces the result without the involvement of any human (scrutineers) in the counting process when the election finishes, other than pressing some buttons to run the software program. This opacity can cause harm of various level, including electing a wrong winner that is not intended by the voters but because of software bugs[1]. Most of these software programs, used across the world by various nations to conduct elections for a public office, lacks quality measures [1, 2] but more importantly they are treated as commercial in confidence and therefore, the members of general public are not allowed to inspect these software programs[3].

In my thesis, I addressed the concern of correctness of vote counting software by proving the correctness of the Schulze Method's implementation in Coq theorem prover [4]. Schulze method is a preferential (ranking) voting method where voters rank the candidates according to their preferences. It is one of the most popular voting method amongst the open-source projects and political groups[2]. While no preferential voting scheme can guarantee all desirable properties that one would like to impose due to Arrow's impossibility theorem [5], the Schulze method offers a good compromise, with a number of important properties establish by economists, social choice theorists, political scientists, etc. In addition, I ensured that when my implementation produces a winner, it also produces a scrutiny sheet that can be checked independently by scrutineers (universal verifiability). However, for this formalisation the extracted OCaml code was very slow, so I wrote another (fast) implementation that I proved equivalent to the slow one. This implementation was able to count millions of ballots [6]. In both formalisation, I assumed that (preferential) ballots were in plaintext, i.e., ranking on every ballot was in a (plaintext) number. Preferential ballots, however, admit "Italian" attack [7, 8]. If the number of participating candidates is significantly high in a preferential ballot election, then a ballot can be linked to a particular voter if published on bulletin board. A coercer demands a voter to mark them as first and for the rest of candidates in a certain permutation. Later, the coercer checks if that permutation appears on the bulletin board or not. In order to avoid this attack on the Schulze method, I used homomorphic encryption to count the (encrypted) ballots, without decrypting any individual ballot, and addressed verifiability by generating a independently checkable scrutiny sheet (certificate) augmented with zero-knowledge-proofs for various claims, e.g., honest decryption, honest shuffle, made during the counting [9]. Finally, I wanted to develop to formally verified scrutiny sheet checker for encrypted ballots Schulze election, but due to lack of time[3] I worked on a scrutiny sheet checker for a simple approval voting election, International Association of Cryptologic Research (IACR) election [10]. In addition, I was involved in formalisation of single transferable vote, used in the Australian Senate [11].

2 Current Work at Cambridge and Previous Work at Melbourne

Currently, I am working as a Postdoctoral researcher at the university of Cambridge. In my current project *Combinators for Algebraic Structures*[4], I am formalising various graph algorithms on *semir-*

[1] A software bug elected a wrong winner: `https://www.areanews.com.au/story/3971893/mercuri-robbed/`
[2] `https://en.wikipedia.org/wiki/Schulze_method#Users`
[3] PhD duration is 3.5 years in Australia
[4] `https://www.cl.cam.ac.uk/~tgg22/CAS/`

ing algebraic structure and combinators (functions) to combine two, or more, algebraic structures. In this work, I am developing a mathematical correct-by-construction [4] framework, in Coq thereom prover, based on theory of routing algebra [12, 13] to alleviate network-engineers from proving the correctness of their protocol and allow them to focus entirely on protocol design. All they need to do is express their protocol in our framework, and it will tell what properties the protocol follows and what it does not. In addition, this framework can be used in operation research, given that its underlying principles are very similar to protocol design.

At the university of Melbourne, I worked on security concurrent separation logic for formally reasoning about the information flow in concurrent programs. I used SecureC, a tool developed at the university of Melbourne, to formalise an email server, an auction server, and a location server. All these works were proven to leak no sensitive information to attackers, assuming that the compiler respects all assumption[5]. In addition, I developed a information flow secure gradient descent algorithm in SecureC for trusted execution environment, e.g., Intel SGX and ARM TrustZone. This work has been informally presented at PaveTrust[6].

3 Future Work

My long-term aim is to make formal verification accessible and ubiquitous in software development, specifically for the software programs deployed in public domain. My expertise in **Theorem Proving, Election Security, and Cryptography** gives me an unique perspective to solve challenging problems that matter to many democracies.

Throughout my past and current research, I have extensively used formal verification techniques to verify the software programs, and I would like to continue to do so in the future as well. In future, I would like to:

- focus on developing formally verified (electronic) voting software (components) programs for low coercion situations to promote direct democracy in Coq theorem prover. Switzerland often conducts referendums on issues that matter to their democracy and encourages its citizens to participate actively in decision making. It would be interesting to design and implement electronic voting components, verified in Coq theorem prover, for low-coercion situations for various vote counting methods such as Single Transferable Vote, First Past the Post, Instant-runoff, etc., on encrypted ballots. All these voting methods differs from each other so counting on encrypted ballot would be challenging task, while ensuring correctness, privacy, and verifiability. The rationale is that once we have a formally verified components, anyone –government or members of general public– can use them for conducting elections, referendums, verifying election outcomes, etc. It would encourage the voters to actively participate in various decision making process without fear of vote selling.

- focus on formally verified cryptographic primitives used in electronic voting, e.g., sigma protocols (zero-knowledge-proof), verifiable (shuffling) mix-nets, multi-party computations, etc., in Coq theorem prover. In my all projects, I have formalised cryptographic primitives but ended up extracting OCaml code[7] and used OCaml compiler to compile the code to machine

[5] under submission

[6] `https://www.acsac.org/2021/workshops/pavetrust/PAVeTrust-Program.pdf`

[7] We can extract OCaml/Haskell/Scheme code from Coq formalisation.

level. However, OCaml compiler is not proven correct and therefore it may introduce new bugs in our code. The challenging part of this project would be to compile the formalised Coq cryptographic code to assembly, or binary, code all the way down to machine level. It is highly non-trivial, but it opens the door of collaboration with other research group working in verified compilation.

- focus on formally verified decentralised peer-to-peer technical solution, inspired by [14, 15], in Coq theorem prover which will help whistleblowers in leaking documents and exposing corruption without revealing their identity. Being vocal against the government is one the most fundamental right of any citizen, but many authoritative governments do not appreciate dissent of any form. Therefore, it uses its powerful machinery to punish the dissident, in the name of national security. For example, David McBridge, a former Australian Defence Force lawyer, is facing a threat, if charged guilty, of lifetime jail after leaking the material alleging war crimes by members of the Australia's Special Operations Task Group in Afghanistan (Australia is ranked very high in democracy index[8]). This research opens the door of collaboration with many groups working in verified networking, and verified distributed systems.

- focus on formally verified (computational) social choice theory in Coq theorem prover. Voting methods admit many properties established by political scientists, social choice theorists, and economists. For example, Schulze method follows Condorcet criterion, Reversal symmetry, Polynomial runtime, etc., so when we formalise Schulze method, or in fact any vote-counting method, in Coq theorem prover, we can push the boundary of correctness by proving that our implementation of Schulze method also follows all the properties, i.e., Condorcet criterion, Reversal symmetry, Polynomial runtime, etc [16]. In addition, we can also analyse these methods from computational point of view, also by formalising it in the Coq theorem prover, that how easy from computational perspective to change the outcome of an election, etc. This research opens the door of collaboration with political scientists, social choice theorists, economists, game theorists, etc.

- focus on formally verified combinators for algebraic structure (CAS) in Coq theorem prover (continue my collaboration with Timothy Griffin). Currently, CAS formalisation is highly focused on networking protocols, but it can be adapted for other areas, e.g., optimisation, clustering, algebraic program analysis , etc. In the CAS, we use an abstract algebraic structure *semiring* as an underlying structure for computation and and we model various algorithm at the top this structure. In this setting, an algorithm can compute different values depending on the concrete structure of semiring, e.g., the same algorithm can compute shortest path, longest paths, data flow of imperative programs, and many more, [17] for an appropriate semiring.

References

[1] Andrew Conway, Michelle Blom, Lee Naish, and Vanessa Teague. An Analysis of New South Wales Electronic Vote Counting. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '17, New York, NY, USA, 2017. Association for Computing Machinery.

[8] `https://worldpopulationreview.com/country-rankings/democracy-countries`

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

[3] Australian Electoral Commission. Letter to Mr Michael Cordover, LSS4883 Outcome of Internal Review of the Decision to Refuse your FOI Request no. LS4849, 2013. available via `http://www.aec.gov.au/information-access/foi/2014/files/ls4912-1.pdf`, retrieved October 23, 2019.

[4] Dirk Pattinson and Mukesh Tiwari. Schulze Voting as Evidence Carrying Computation. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving*, pages 410–426, Cham, 2017. Springer International Publishing.

[5] Kenneth J Arrow. A difficulty in the concept of social welfare. *Journal of political economy*, 58(4):328–346, 1950.

[6] Lyria Bennett Moses, Rajeev Goré, Ron Levy, Dirk Pattinson, and Mukesh Tiwari. No More Excuses: Automated Synthesis of Practical and Verifiable Vote-Counting Programs for Complex Voting Schemes. In *International Joint Conference on Electronic Voting*, pages 66–83. Springer, 2017.

[7] J. Otten. Fuller disclosure than intended. 2003. Accessed on October 17, 2019.

[8] Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. Shuffle-sum: coercion-resistant verifiable tallying for STV voting. *IEEE Trans. Information Forensics and Security*, 4(4):685–698, 2009.

[9] Thomas Haines, Dirk Pattinson, and Mukesh Tiwari. Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme. In Supratik Chakraborty and Jorge A. Navas, editors, *Verified Software. Theories, Tools, and Experiments*, pages 36–53, Cham, 2020. Springer International Publishing.

[10] Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified Verifiers for Verifying Elections. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 685–702, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Milad K. Ghale, Rajeev Goré, Dirk Pattinson, and Mukesh Tiwari. Modular Formalisation and Verification of STV Algorithms. In Robert Krimmer, Melanie Volkamer, Véronique Cortier, Rajeev Goré, Manik Hapsara, Uwe Serdült, and David Duenas-Cid, editors, *Electronic Voting*, pages 51–66, Cham, 2018. Springer International Publishing.

[12] R. C. Backhouse and B. A. Carré. Regular Algebra Applied to Path-finding Problems. *IMA Journal of Applied Mathematics*, 15(2):161–186, 04 1975.

[13] Timothy G. Griffin and Joäo Luís Sobrinho. Metarouting. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, page 1–12, New York, NY, USA, 2005. Association for Computing Machinery.

[14] Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups. In *Australasian Conference on Information Security and Privacy*, pages 325–335. Springer, 2004.

[15] Shen Noether and Adam Mackenzie. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[16] Mukesh Tiwari and Dirk Pattinson. Machine Checked Properties of the Schulze Method. In *7th Workshop on Hot Issues in Security Principles and Trust*, 2021.

[17] Michel Gondran and Michel Minoux. *Graphs, Dioids and Semirings: New Models and Algorithms*, volume 41. Springer Science & Business Media, 2008.