Application of Formal Verification in Cryptography and Voting

Mukesh Tiwari

My research aims to build **correct software** using the Coq theorem prover. I am passionate about verifying the computer programs used for decision making in public domain. In particular, I focus on formal verification of computer programs used in elections, cryptography, and computational social choice theory. Below are some potential projects that I would like to focus, if given the position.

1 Future Work

- I would like to focus on formally verified cryptographic primitives used in verifiable computing, multi-party computation (MPC), and blockchain. The recent development in the area of zero-knowledge-proofs, in particular zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK), has led to an exponential growth in many useful privacy preserving applications in verifiable computing, MPC, etc. Most of these applications, however, are written in unsound language (Rust/Python/C/Java) and contain bugs, lack security proof, and therefore using formal verification to improve their correctness and security would bolster the confidence of a user of these applications. In addition, zk-SNARK has massive potential in electronic-voting and machine learning.
  - The current practice to verify the integrity of an election conducted electronically is to recompute the whole count on the publicly available (electronic) ballots, but this excludes many voters from verifying the integrity of the election because they do not posses a powerful enough computing device. Many of zk-SNARKs produce a very small size proof for the integrity of a computation regardless of the computation data. Therefore, if we deploy zk-SNARK in electronic-voting, it would increase the number of scrutineers because anyone, including the mobile devices holders, can verify the integrity of an election by checking these small size proofs.
  - In recent years, machine learning models are getting bigger and bigger and cannot be trained using a normal computer. Therefore, most of them are trained in cloud, but this raises the question if the cloud has used the right set of data to train them model or not. One way to solve this problem is to use verifiable computing and force the cloud to generate a short proof (zk-SNARK) with the trained machine learning model. This short proof can be then checked by any independent third party to attest the integrity of training.
- In one of my recent project, me with my co-authors –after 1.5 years of painstaking effort– have formalised a critical piece of code (mix-network) in the SwissPost –used in legally binding elections in Switzerland– in the Coq theorem prover (and in the process, we found a flaw in a decade old cryptographic proof). We extracted OCaml code from our Coq formalization that can be used an oracle to test the Java code of SwissPost. This is one step closer to achieve correctness in electronic voting but not sufficient because it does not deal with the actual Java code. Therefore, as an extension to this project, I would like to add reasoning support for the Java language in Coq-Iris so that I can reason about the actual Java code of SwissPost in Coq and prove its correctness.

Related Publication:

- Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Machine-checking Multi-Round Proofs of Shuffle:Terelius-Wikstrom and Bayer-Groth. 32nd USENIX Security Symposium (USENIX 2023), Anaheim, California, USA, August 9-11, 2023. `https://github.com/mukeshtiwari/secure-e-voting-with-coq`

- At the University of Melbourne, I worked on security concurrent separation logic for formally reasoning about the information flow security in concurrent programs. I used *SecureC*, a tool developed at the university of Melbourne, to formalise an email server, an auction server, a location server, and a differentially private gradient descent algorithm. All these works were proven to leak no sensitive information to attackers, assuming that the compiler respects all assumption and soundness of the implementation of *SecureC*. However, *SecureC* is very limited in terms of expressibility and tooling, and therefore it cannot be used to model a real world case study. As a future research, I would like to add information flow support to Coq-Iris –a Coq library– with my co-author Toby Murray.

  Related Publication:

  – Toby Murray, Mukesh Tiwari, Gidon Ernst, and David A. Naumann. Assume but Verify: Deductive Verification of Leaked Information in Concurrent Applications. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS '23, Copenhagen, Denmark, 26-30 Nov. `https://github.com/mukeshtiwari/IFMachine/`.

- At the University of Cambridge, I worked on formalisation of various graph algorithms using the Coq theorem prover in *semiring* (algebraic structure). In our framework, depending on concrete instantiation of semiring operators, the same algorithm can compute shortest path, longest paths, widest paths, multi-objective optimisation, and many more. In addition, I also worked on formalisation of theory of algebraic reductions. However, the focus of this project was path problems but in future I would like to formally verify generalised (semiring) algorithms related to data flow analysis of imperative programs (Newtonian program analysis), Petri nets, etc., in the Coq theorem prover.

  Ongoing Work:

  – Modelling Networking Protocols Mathematically. In this work, we develop a formally verified framework that researchers can use to verify the properties of their protocols (planning to submit to CAV 2024). `https://github.com/mukeshtiwari/Semiring_graph_algorithm`.

  – An Algebraic Framework for Multi-Objective Optimisation. In this work, we develop a formally verified framework in the Coq theorem prover that can be used to model various multi-objective optimisation problem as a graph algorithm in the semiring framwork. `https://github.com/mukeshtiwari/Formally-Verified-MultiObjective-Optimisation`.

- At the University of Oxford, I am exploring the avenues to bridge the gap between a security protocol (formal communication model) with its implementation using session types, and our goal is to produce a more realistic distributed executable model of the security protocol. Currently, as a first step, I am focussing on Signal app where my goal is to prove (or disprove) that its Java implementation follows the communication model described in the Signal's documentation.

2 Past Work

My PhD research was focused on verifying electronic voting, specifically vote-counting schemes, in the Coq theorem prover. The goal was to bring three important ingredients (i) correctness, (ii) privacy, and (iii) verifiability of a paper-ballot election to an electronic setting (electronic voting). In my thesis, I demonstrated the correctness of the Schulze method by implementing it and proving its correctness in the Coq theorem prover. The Schulze method is a preferential (ranking) voting method where voters rank the participating candidates according to their preferences, and it is one of the most popular voting method amongst the open-source projects and political groups[1]. In addition, my implementation ensured (universal) verifiability by producing a scrutiny sheet with the winner of an election. The scrutiny sheet contained all the data related to the election that could be used to audit the election independently. The extracted OCaml code from this formalisation, however, was very slow, so I wrote another fast implementation, proven equivalent to the slow one, capable of counting millions of ballots.

[1] `https://en.wikipedia.org/wiki/Schulze_method#Users`

Related Publications:

- 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. `https://github.com/mukeshtiwari/formalized-voting/blob/master/paper-code`
- 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 Robert Krimmer, Melanie Volkamer, Nadja Braun Binder, Norbert Kersting, Olivier Pereira, and Carsten Schürmann, editors, Electronic Voting, pages 66–83, Cham, 2017. Springer International Publishing. `https://github.com/mukeshtiwari/formalized-voting/tree/master/SchulzeOCaml`

In both formalisation, I assumed that (preferential) ballots were in plaintext, i.e., ranking on every ballot was in a (plaintext) number, but preferential ballots admit the "Italian" attack. The "Italian" attack is a tactic where a coercer seeks to link a specific ballot to a particular voter, when the number of participating candidates are significantly high in a preferential ballot election. The coercer demands the voter to rank a particular candidate first and the remaining candidates in a specific permutation. After the voter casts their ballot, the coercer checks if the exact permutation specified by the coercer appears the published bulletin board to see 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. Moreover, I addressed verifiability by generating a scrutiny sheet (certificate) augmented with zero-knowledge proofs for various claims, e.g., honest decryption, honest shuffle, etc. I formalised every single claim in the Coq theorem prover to ensure that there was no gap between the pen-and-paper proof and the actual implementation. Finally, I wanted to develop to formally verified scrutiny sheet checker for encrypted ballots Schulze election, but due to the lack of time[2] I worked on a scrutiny sheet checker for a simple (approval voting) method used in International Association of Cryptologic Research (IACR) election. In addition, I was involved in formalisation of single transferable vote, used in the Australian Senate election.

Related Publications:

- Thomas Haines, Dirk Pattinson, and Mukesh Tiwari. Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme, In Verified Software: Theories, Tools, and Experiments. Springer, 2019. `https://github.com/mukeshtiwari/EncryptionSchulze/tree/master/code/Workingcode`
- 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. `https://github.com/mukeshtiwari/secure-e-voting-with-coq`
- 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 Serdültt, and David Duenas-Cid, editors, Electronic Voting, pages 51–66, Cham, 2018. Springer International Publishing. `https://github.com/mukeshtiwari/Modular-STVCalculi`

My lab will be a diverse place where students and researchers from formal verification, cryptography, political science, social science, social choice theory will interact, discuss, collaborate on the ideas that matters to democracies and common people.

[2]PhD duration is 3.5 years in Australia