Research Statement

Mukesh Tiwari

My work aims to build **correct software** using the Coq theorem prover. I focus on formal verification of computer programs used in elections, cryptography, networking, 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 a 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.
- 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, neurosymbolic programming, etc., in the Coq theorem prover.
- 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 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. I would like to work on adding information flow support in Iris –a Coq library– with my co-author Toby Murray.

- In one of my recent project, me with my co-authors –after 1.5 years of painstaking effort– formalised a critical piece of code (mix-network) in the SwissPost –used in legally binding elections in Switzerland– in the Coq theorem prover and extracted an OCaml code from our formalisation against which the Java code of SwissPost can be tested. In the process, we found a flaw in a decade old cryptographic proof. Given my expertise in formal verification, I would like with my PhD students and postdocs researchers to explore and formalise other piece of code used in public domain, e.g., machine learning models and code deployed in self-driving cars, decision-making algorithms, etc.

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. 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.

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. If the number of participating candidates are significantly high in a preferential ballot election, then a ballot can be linked to a particular voter if published on a bulletin board. A coercer demands the 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. Moreover, I addressed verifiability by generating a scrutiny sheet (certificate) augmented with zero-knowledge-proofs for various claims, e.g., honest decryption, honest shuffle, during the counting. 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 election, International Association of Cryptologic Research (IACR) election. In addition, I was involved in formalisation of single transferable vote, used in the Australian Senate election.

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.

[1] `https://en.wikipedia.org/wiki/Schulze_method#Users`
[2] PhD duration is 3.5 years in Australia