Research Statement

Mukesh Tiwari

My work aims to build **correct software programs**, which affect common people, using the Coq theorem prover[1]. I focus on formal verification of software programs used in elections, cryptography, networking, and computational social choice theory. Numerous critical decisions, e.g., producing the winner of an election by an election commission, are taken based on the output of a software program. However, if the software program contains bugs, then it may produce a wrong output. Therefore, the government entity (election commission) can lose its reputation. Worse yet, nowadays there is more and more automation of decision-making in various bodies of a government (democracies) based on software programs, AI models and machine learning algorithm, without any justification. Therefore, it is more imperative than ever that we formally verify these software programs to make the government decision-making more trustworthy.

1 Research Experience

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, correctness, privacy, and (universal) verifiability, of a paper ballot election to an electronic setting (electronic voting). In a paper ballot election, correctness is ensured by scrutineers, and privacy and verifiability come for free because of secret paper ballots. However, achieving these three desirable properties are difficult in electronic voting because software programs, used in various stages of an election, work in a opaque (blackbox) manner.

In my thesis, I demonstrated the correctness of a vote counting software program by implementing the Schulze method, a voting method popular amongst the open-source projects and political groups[2], and proving its correctness in Coq theorem prover [?]. 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 [?]. However, plaintext preferential ballots admit "Italian" attack [?, ?]. Therefore, 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[3] 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 [?].

In my current project *Combinators for Algebraic Structures*[4], I am formalising various graph algorithms on *semiring* algebraic structure and combinators (functions) to combine two, or more,

[1] `https://coq.inria.fr/`
[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/`

algebraic structures. This formalisation can be used in the networking and optimisation research. In addition, our goal is to also understand the graph neural network using our library. 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* to formalise an email server, an auction server, and a location server. In addition, I developed a information flow secure gradient descent algorithm in federated learning setting for trusted execution environment, e.g., Intel SGX and ARM TrustZone. This work has been informally presented at PaveTrust[5]. Our goal with federated learning was to come with a prototype where hospitals can share their data in a secure manner, without any concern of regulatory policies.

2 Future Research

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, Cryptography, Networking, and Security** gives me an unique perspective to solve challenging problems that affect to many common people. At Cambridge Engineering Department, I will focus on formally verifying the components used in federated learning, e.g., networking libraries, machine learning models, etc., in Coq theorem prover [**?**][6] [**?**][7]. Federated learning is now a gold standard for distributed machine learning. However, it suffers from many problem, e.g., software bugs[8], membership inference attack [**?**], etc. For many applications software bugs do not matter, but if the machine learning model deployed in a public domain (safety-critical) for decision making, then it can be fatal, e.g., Tesla cars not detecting children (child-sized mannequin) in the road and ended up hitting[9]. We can eliminate software bugs by mathematically (formally) proving the correctness of the software, used in federated learning. In addition, I will focus on developing formally verified homomorphic encryption for federated setting to train a machine learning model on encrypted (sensitive) data, e.g., medical data, to avoid membership inference attack. One area that I am very keen to explore is developing an economic market for sensitive data sharing using consortium blockchain[10]. The goal is to promote hospitals to share their data securely and reward them financially. To achieve this, I will focus on the verification of sigma protocols (zero-knowledge-proof), verifiable (shuffling) mix-networks, multi-party computations, secret sharing, secure communication, zk-snark, etc. Logic (theorem proving) can also be used in understanding fairness, bias, etc., of a machine learning model that I am willing to explore with the help of the researchers at Cambridge Engineering Department.

3 Teaching Experience

My teaching philosophy is to not immediately reach a solution but to develop a thinking process (problem solving mindset) that leads to the solution. I have practiced this for 3 years of teaching at a technical university. I can teach any information technology course, e.g, *Software Engineering and Design (4M21), Computing (1P4), Information Engineering (2P8), Data Transmission (3F4), Computer Systems (4F14), Advanced Information Theory and Coding (4F5).* In addition, I would like to introduce a formal methods course to teach proving the correctness of a computer program.

[5] `https://www.acsac.org/2021/workshops/pavetrust/PAVeTrust-Program.pdf`
[6] https://github.com/dselsam/certigrad
[7] `https://github.com/IBM/FormalML`
[8] `https://github.com/theano/theano/issues/4770`
[9] https://www.theguardian.com/technology/2022/aug/09/tesla-self-driving-technology-safety-children
[10] There is already a similar project for carbon credit at Cambridge `https://4c.cst.cam.ac.uk/`