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 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

Teaching Statement

Mukesh Tiwari

My teaching philosophy is not to immediately reach a solution but to develop a thinking process (problem solving mindset) that leads to the solution. I believe every student is different and has a unique style of learning, and my role is to help them find and hone their style. When I started teaching as an assistant professor at the International Institute of Information Technology (IIIT), Bhubanesware, India[1], my single biggest challenge was keeping the students engaged in my class, especially the first year students in C programming course. At the IIIT, I taught C programming to first year students, Compiler Design and Java programming to third year students, and Cryptography to final year (4th year) students. In each course, every single problem, more or less, boiled down to keeping the students engaged in a topic. In order to keep them engaged in a class, I took a Coursera course on learning[2] and read many academic articles and non-academic articles about effective learning. I tried some of the techniques suggested in the Coursera course and academic and non-academic articles in my classes. Below I describe my experiences with teaching and efforts to engage my students.

1 Setting Clear Goals

In every course, I started with the end goal of the course. For example, in C programming course, I told my students that by the end of this course they would be able to write simple C programs, e.g., calculator, validating a debit card, and other simple real-world problems. The rationale was to show a vision to excite them for learning and instill the feeling of empowerment, from being a consumer to being a developer of software programs.

2 Start with Why

One thing that I learnt by teaching for 3 years is that if you want a concept to stick in someone's mind, then start with a *why*[3]. For example, when I introduced functions in C programming course, I wanted to convey the need of functions. Therefore, I started the class by asking them to write a program, using pen and paper, of **factorial of a number** n ($n!$). Every one was comfortable with loops, so it was quick. I then asked them to write another program: k^{th} **power of the factorial of** n ($(n!)^k$). In this assignment, some added an outer loop to cover the factorial computation, and some added another loop after the factorial computation (stored the factorial computation result and used it in later computation). At this point, I asked the class how one could write a program

[1]It is a small technical school
[2]`https://www.coursera.org/learn/learning-how-to-learn`
[3]I learnt this idea by reading the book *Start With Why* by Simon Sinek.

of **factorial of the k^{th} power of the factorial of n** $(((n!)^k)!)$. The class slowly realised that they needed to duplicate a lot of code. I then introduced functions and showed them the solution of the previous problem by function composition.

3 Catching my Mistakes

To promote active participation, at the beginning of every class I would announce that on a few occasions I would make deliberate mistakes in solving a problem, and the goal of the class was to catch me on the spot. If the class succeeded, they received class participation marks, otherwise I told them my mistake and no one received any marks. In every class, especially in programming, first I would teach a topic, and later I would solve some problems on a blackboard, related to the topic. It was during the problem solving where I committed deliberate mistakes. It increased the class participation by an order of magnitude[4]. In every single feedback that I got during my 3 years of teaching, almost everyone appreciated this idea of making deliberate mistakes.

4 Potential Courses

I feel qualified to teach most of the computer science courses because of my background in computer science, but my natural preference is the courses close to my research area, e.g., theorem proving, cryptography, logic and its applications in computer science, discrete mathematics, algorithms and data structures, introductory programming course (C/C++/Java/Python/Haskell/OCaml), foundation of computing, etc. In addition, I would also like to design a course to teach various voting methods used around the world and their advantages and disadvantages from a social choice theory perspective. Apart from these topics, I will be more than happy to teach other courses, given enough preparation time, at introductory and intermediate levels, e.g., type theory, theory of programming languages, networking, operating systems, databases, etc.

Finally, I would like to emphasize the my teaching philosophy is not to immediately reach a solution but to develop a thinking process (problem solving mindset) that leads to the solution.

[4] I learnt this from advertising agencies where they write some ads wrong intentionally to grab the attention.