Mukesh Tiwari
Cambridge, United Kingdom
+447824648138
mt883@cam.ac.uk

To
The Hiring Committee,
University of Cambridge,
Cambridge

Application for the post of Assistant Professor in Security and Privacy (NR33473)

Dear Hiring Committee,

I am writing to apply the **Assistant Professor** job in Security and Privacy. I have extensive research experience in theorem proving (Coq theorem prover), cryptography, election security, and social choice theory. I have a PhD from the Australian National University, Canberra, Australia and Currently, I am working as a Senior Research Fellow at the University of Cambridge since October 2021. Before moving to Cambridge, I was a research fellow at the University of Melbourne.

The goal of my PhD was to bring three important ingredients, correctness, privacy, and (universal) verifiability, of a paper ballot election to an electronic setting (electronic voting). I demonstrated: (i) correctness by implementing and proving the correctness of a vote-counting algorithm, the Schulze method, in the Coq theorem prover, (ii) privacy by using homomorphic encryption to encrypt the ballots and computed the winner by combining all the (encrypted) ballots, and (iii) verifiability by means of various zero-knowledge-proofs. At Cambridge University, as an Assistant Professor, I would like to expand my research area into other areas of security and formal verification, e.g., domain specific language to reason about functional correctness and security properties –from computational complexity perspective– of cryptographic algorithms, anonymous communication, blockchain, zk-snark, information flow security, computational complexity of social choice methods, etc. My research has been published in Interactive Theorem Proving (ITP), Computer and Communications Security (CCS), Electronic Voting (EVote), International Conference on Cryptology in India (IndoCrypt), and some (finished) works have been submitted to ESOP, USENIX, and SIGCOMM. All my research work has been formalised in the Coq theorem prover.

PhD Thesis

In my thesis, I demonstrated the correctness of a vote-counting software program by implementing and proving the correctness of the Schulze method 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. While no preferential voting scheme can guarantee all desirable properties that one would like due to Arrow's impossibility theorem, the Schulze method offers a good compromise with a number of important properties established by economists, social choice theorists, and political scientists. From my Coq implementation of the Schulze method, I used Coq's extraction mechanism to get an OCaml program. Then I used the OCaml compiler to compile the extracted OCaml program to get an executable to count ballots. In addition to correctness, my implementation also ensured (universal) verifiability by producing a scrutiny sheet. The scrutiny sheet contained all the data to audit the election independently. The (extracted) OCaml program, however, was very slow and could not count more than 10,000 ballots. Therefore, I wrote another Coq implementation, proven equivalent to the slow Coq implementation, from which the (extracted) OCaml program was able to count millions of ballots.

In both, slow and fast, Coq implementations, 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 publicly, on a bulletin board. The attack is: a coercer demands a voter to mark them as first and for the rest of candidates in a certain given order (permutation). Later, the coercer checks if that the order appears on the bulletin board or not. Although my previous two Coq implementations satisfied correctness and verifiability criteria, they lacked privacy due to "Italian" attack. In order to avoid this attack on the Schulze method, I used a homomorphic encryption to encrypt the ballots and computed the winner by combining all the

(encrypted) ballots, without decrypting any individual ballot (privacy). Moreover, I addressed verifiability by generating a scrutiny sheet (certificate) augmented with zero-knowledge-proofs (zkp), e.g., honest decryption zkp, honest shuffle zkp. This work was carried out in the Coq theorem prover, so I ended up achieving correctness, privacy, and verifiability. The downside of an encrypted ballot Schulze election, or in fact any encrypted ballot election, is difficulty in auditing because of involved mathematics of cryptography. Therefore, in the future I want to explore the possibility of a verified prototype of scrutiny-sheet checker for encrypted ballots Schulze elections. Finally, I worked on a verified prototype of a simple approval voting election scrutiny-sheet checker for International Association of Cryptologic Research (IACR). In addition, I was involved in the formalisation of single transferable vote, used in the Australian Senate.

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 that affect common people. My expertise in **Theorem Proving, Cryptography, Election Security, and Social Choice Theory** gives me an unique perspective to solve challenging problems that matter to many democracies and its citizens.

Mathematically Proven Correct Cryptographic Algorithms

Cryptographic algorithms are used ubiquitously to secure the data and correctness is an utmost requirement for any cryptographic algorithm implementation. Therefore, I will focus on developing mathematically proven correct cryptographic algorithms used in electronic voting, blockchain, and secure communication, e.g., sigma protocols (zero-knowledge-proof), verifiable (shuffling) mix-networks, multi-party computations, secret sharing, zk-snark, etc. The rationale behind implementing these algorithms is that anyone can use them to construct an utility, e.g., an election scrutiny-sheet checker, a vote-tallying system based on blockchain, a verifiable ballot mixing service, an auction server, etc. One of the motivation behind this project is to replace the SwissPost Java implementation, used in democratic elections in Switzerland, with a mathematically proven correct Coq implementation.

Mathematically Proven Correct Vote-Counting Algorithms

In future, I will focus on developing mathematically proven correct software programs for vote-counting methods used across the world such as *Single Transferable Vote (STV), First Past the Post (FPTP), Instant-runoff voting(IRV), etc.,* in the Coq theorem prover. All the vote-counting methods, by design, lend themselves well to computing the winner from plaintext ballots; however, so far there is very little research in computing the winner from encrypted ballots, while ensuring correctness, privacy, and verifiability. Therefore, producing the winner from encrypted ballots is a challenging task. The motivation for this project is that once we have mathematically proven correct components, anyone –election commission or members of general public– can use them to conduct elections, referendums, and verify elections' outcome without worrying about software bugs. The cryptographic algorithms formalised in the previous step are going to be used as a building block in this project.

Mathematically Proven Correct Decentralised Application

I will focus on a mathematically proven correct decentralised peer-to-peer technical solution in the Coq theorem prover. The motivation is to help whistleblowers in leaking documents and exposing corruption without revealing their identities. 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 dissidents, in the name of national security. The inspiration for this project comes from David McBridge and Richard Boyle. David McBride is facing a threat of lifetime jail after leaking the material alleging war crimes by members of the Australia's Special Operations Task Group in Afghanistan, while Richard Boyle is facing 161 years for exposing the corruption inside the Australian Taxation office (Australia is ranked very high in democracy index). This research will open the door of collaboration with many groups working in verified networking, and verified distributed systems.

Mathematically Proven Correct Social Choice Properties

Computational social choice theory is a research area that is concerned with aggregation of ballots (preferences) of multiple voters (agents) and encompasses computer science, mathematics, economics, and political science. Typical applications of computational social choice theory is voting (preference aggregation), resource allocation,

and fair division. Most of the proofs in computational social choice theory are pen-and-paper proofs, and one of my long term future research goal is to make them more precise using the Coq theorem prover.

My current focus is voting because voting methods admit many excellent (social choice) properties established by political scientists, social choice theorists, and economists. For example, the Schulze method follows Condorcet criterion, reversal symmetry, polynomial runtime, etc., so when we formalise the Schulze method, or in fact any vote-counting method, we can push the boundary of correctness by proving that our implementation of the Schulze method also follows all the properties. In addition, we can analyse these voting methods from computational complexity perspective of bribery, if by bribing a certain amount voters a specified candidate can be made an election's winner. This research opens the door of collaboration with political scientists, social choice theorists, economists, and game theorists.

Current Work and Past Work

In my current project *Combinators for Algebraic Structures (CAS)*, I am formalising various graph algorithms on *semiring* algebraic structure and combinators (functions) to combine two, or more, algebraic structures. In this work, I am developing a mathematical correct-by-construction framework, in Coq theorem prover, based on theory of generalised path-finding algebra. In our framework, depending on concrete instantiation of semiring operators, the same algorithm can compute shortest path, longest paths, widest paths, multi-objective optimisation, data flow of imperative programs, and many more. In fact, the Schulze method is one instance of our framework. However, the current CAS implementation is highly focused towards networking protocols, so as a future work I will focus on adding more algorithms in CAS related to multi-objective optimisation, data flow analysis of imperative programs, and voting.

As a research associate at the University of Melbourne, I did acquire hands-on knowledge of separation logic and information flow security. I have spearheaded three projects: (i) A formally verified auction server, (ii) A formally verified location server, and (iii) A formally verified machine learning algorithm that is resistant to side-channel attacks and can run inside the Intel SGX (Software Guard Extensions) enclave for learning on sensitive data. All three implementations have been proven memory safe (using separation logic) and free from information leaks using the SecureC tool, a tool developed at the University of Melbourne.

Teaching

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, 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 *learning-how-to-learn* 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.

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. In addition, in the beginning of every single class I would tell my students explicitly the topics we were going to study and their importance. For example, when I taught pointers, I explained a very high-level idea of an operating system and usage of pointers in operating system to access memory locations. In addition, I told them Linux is written in C and encouraged them to check the source code. It helped the class in understanding the importance of every individual topic towards the end goal.

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*. 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 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.

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. In every single feedback that I got during my 3 years of teaching, almost everyone appreciated this idea of making deliberate mistakes.

Projects for Learning

For every course that I taught, I designed projects related to the concepts and ideas presented in the course. It was a part of my teaching philosophy to impart critical thinking. Furthermore, I asked my students to come up with their own ideas to promote idea-exploration, a key step to develop a critical thinking and problem solving mindset. In the process of idea-exploration, the students would come up with various interesting ideas, but there were many instances when I felt very happy. For example, a group of third year students wanted to understand a DNS server by writing their own. Even though they could not write a DNS server from scratch successfully, they managed to understand most of the workings of a DNS server by downloading a C code from the Internet. I consider it a trophy for myself, given that IIIT is a very small technical school. In addition, because of my own competitive programming background, I redesigned the C programming Lab and introduced competitive programming, which led to first ACM-ICPC team participation from IIIT. More importantly, it made many of the students curious about programming, which was considered a difficult subject.

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.

My lab will be a diverse place where students and researchers from formal verification, cryptography, political science, social choice theory will interact, discuss, collaborate on the ideas that matters to democracies and societies. I look forward to hearing from you. Please let me know if you have any questions.

Your Sincerly,

Mukesh Tiwari