My research focuses on formal verification of software programs used in democratic process, e.g. elections, as a facilitator in decision making process. The rationale for verifying these software programs is that various criticial decisions are taken by the different bodies of a government based on the output produced by these software programs. However, if the software program involved in the decision making process is bug ridden, then it may produce a wrong output, which could lead to a poor decision making. A possible side effect of these kind of poor decision making based on wrong output could hamper the trust of memebers of general public in system despite the fact that there were no such intentions by government. Now a days, there are more and more automation of decision making based on machine learning algorithm, artificial intelligence, etc., it is more imperative than ever that we formally verify these software programs to make the decision making more trustworthy.

1 PhD Work

During my PhD, my research was focussed on verifying electronic voting, specifically vote counting schemes, and brining three important ingredients, correcntess, privacy, and verifiability, of paper ballot election to electronic setting (electronic voting). In a paper ballot election, correctness and verifiability is assured by public scruntineers, appointed by participating candidates and political parties, observing the counting process conducted by election commission officials. Moreover, the privacy in paper ballot election comes for free because of the secret ballot mechanism, introduced by Australia in 1855, where the voters choices are anonymous. However, none of these desirable properties can be achieved in electronic voting because the software programs, used during the various stages of elections, works in very opaque way. This opacity can cause a harm of various level, but it would be devastating if the vote-counting software program produces a wrong winner, which is not intended by the voters, because of a software bug. In this setting, it is very difficult to establish correctness and verifiability because there is no (direct) involvement of human in the counting process (other that pressing some buttons to run the software program). In most cases, the vote-counting software programs lack the quality measures which could lead to various problems including, but not limited to, producing incorrect result, ballot identification, etc. More importantly, these software programs are treated as commercial in confidence and are not allowed to be inspected by members of general public. As a consequence, the result produced by these software programs can not be substantiated, which is very detrimental for any democratic society. Because of these reasons, the early adopter, Germany and The Netherlands, of electronic voting system were the early abondeners. In my thesis, I addressed the concern of correctness of vote counting software by implementing and proving the correctness of the implementation (Schulze Method) in Coq theorem prover. I also proved that my implementation of Schulze Method follows the various meta properties, e.g. condercet winner, reversal symmetry. I addressed the privacy concern by using homomorphic encryption to count the (encrypted) ballot without decrypting any individual ballot and verifiability concern by generating a independently checkable scrutiny sheet (certificate) augumented with zero-knowledge-proofs for various claims made during the counting.

2 Current Work

Currently I am working as a PostDoctoral researcher with Toby Murray at the university of Melbourne, and I work on security concurrent separation logic for formally reasoning the information flow in concurrent programs. My current project focuses on a verified email server, inspired by Hagrid key-server, which leaks no sensitive information to attackers. The basic idea behind this project is that given all the APIs of a email server, any attacker would not be able to get any information which is classified sensitive (high).

3 Future Work

Throughout my past and current research, I have extensively used the formal verification techniques to verify the software programs, and I would like to continue to do so in the future as well. In future, I would like to:

- focus on verifying machine learning algorithm. Now a days, machine learning algorithms are deployed in various domains and decisions are taken based on the output of these programs. In general, the most popular langauge to implement these these algorithms are Python which has no concept of type safety. As a consequence, a programmer writing these implementation can make a subtle error, which could be catastrophic. In contrast, a type safe language, e.g. Haskell, would catch many of these errors at the compile time. However, none of these languages would give 100% guarantee that the implementation is correct. The only way we can get 100% guarantee by implementating these algorithm in a theorem pover, Agda, Coq, Lean, and proving that the implementation follows the specification.

- focus on pushing privacy preserving machine learning algorithm in public domain. However, the training data for these algorithms are in plaintext which poses a significant challenge if it is sensitive. A privacy preserving machine learning would have all the benefits of machine learning without any concern about leaking sensitive data to anyone. Righ now, the field is in very nascent phase, but given that there is a lot of progress in homomorphic encryption since 2009, it would be an excellent idea to explore privacy preserving machine learning. The other possible direction could be multiparty computation, but these concepts are very difficult for machine learning researcher to understand. Developing a domain specific language which abstracts the underlying details would help machine learning researchers to push the boundries of privacy preserving machine learning.

- focus on designing probablistic programming language (embedded in a theorem prover) for cryptographic implementation with main focus on correctness. Cryptographic algorithms play a critical role in any application focussed on privacy. If any part of implementation is not done right, it could lead to severe consequences (possible identification of activist, voter). The long term (and ambitous) goal would be developing a verified compiler which would generate assembly code (having various properties such as memory safe, constant time). There are already some work in this area, e.g. Jasmin and CakeML; however, Jasmin's goal is to generate formally verified efficient assembly code and CakeML is a general purpose language. None of these are focussed on probabilistic programming which is a necessary component to model any cryptographic algorithm.

- focus on designing electronic voting protocol and proving the meta properties. Many counties, e.g. Switzerland, often conduct referendums via internet-voting on various issues that matter to their democracy, and it encourages their citizen to participate actively in decison making. It would be interesting to design and implement electronic voting protocols for low-coercion situations, which would encourage the voters to actively participate in various decison making process without any fear of vote selling.

- focus on information flow security in software design (continue my collaboration with Toby Murray). Proving formally the information flow between various components of a software program can leads to security focussed design. In certain cases, where security is paramount, it helps in understanding the potential information leak, which can further be used for assesing the repercussions.