1 Introduction

1.1 Project Statement

This project aims to design a new scheme for Privacy Preserving Model Checking (PPMC) where the formal specifications in model checking is expressed in linear temporal logic (LTL) and the cryptographic tool of multiparty computation (MPC) is used for privacy preserving. And if time permits, we hope to improve the computational complexity of a current scheme which is based on computation tree logic (CTL).

Credit: This project is based on top of Ning Luo and Sam Judson’s prior work on the design of a multiparty computation protocol for in the privacy-preserving setting for computation tree logic (CTL).

1.2 Preliminary Concepts & Motivation

Informally, in designing schemes PPMC, we try to solve the problem of enabling one party to verify that the program developed by the other party satisfies certain desired properties without either party obtaining more information beyond the verification results. For example, consider a financial trading firm that is required to not exceed certain limits on their leverage or to maintain certain capital requirements. A regulator could theoretically demand not just retroactive verification that the firm has met their requirements, but a proactive guarantee that the control software of the firm will not allow traders to exceed these limits. Such a guarantee could be discharged through model checking, yet however, the firm may not want to expose their trading software in plain text at the risk of potentially
Model Checking holds an eminent place in the suite of techniques for formally verifying the correctness of programs, where programs are formulated as finite transition systems and the various desired properties are expressed as logical formulas (which are called formal specifications). These desired properties are verified through explicitly checking all possible executions of a system model and verify that each of them satisfies the corresponding logical formulas.

Linear temporal logic (LTL) and computation tree logic (CTL) are both examples of temporal logic often used in model checking, capable of expressing linear time properties. Formally, a LTL formula $\phi$ is recursively defined as

$$
\phi := \text{true} \mid \text{AF} \| \phi_1 \land \phi_2 \| \neg \phi \| \text{X}\phi \| \phi_1 \text{U}\phi_2 \| \text{F}\phi \| \text{G}\phi \| \phi_1 \text{R}\phi_2 \| \phi_1 \text{W}\phi_2 \| \phi_1 \text{M}\phi_2
$$

where AF is atomic formula; \{G, F, X\} are unary operators such that $X\phi$ (next) means $\phi$ holds true at the next execution step, $F\phi$ (finally) means $\phi$ eventually has to hold true somewhere on the execution path, $G\phi$ (globally) means that $\phi$ has to hold at all subsequent steps on this execution path; \{U, R, W, M\} are binary operators such that $\phi_1 \text{U}\phi_2$ (until) means $\phi_2$ will hold true at some point along the execution path, and until then, $\phi_1$ has to hold true, $\phi_1 \text{R}\phi_2$ (release) means $\phi_1$ has to hold true until and including the point at which $\phi_2$ first becomes true, and if $\phi_2$ never becomes true, $\phi_1$ must remain true forever, $\phi_1 \text{W}\phi_2$ (weak until) means $\phi_1$ has to hold true until $\phi_2$ holds true, and if $\phi_2$ never becomes true, $\phi_1$ must remain true forever, $\phi_1 \text{M}\phi_2$ (strong release) means that $\phi_2$ will hold true at some point along the execution path, and $\phi_1$ has to hold true until and including the point at which $\phi_2$ first becomes true.

In this research project, the first part will be focused on the design of a new PPMC for LTL, and the second part will be specifically focused on improving the asymptotic time complexity of our existing CTL based PPMC scheme by designing more efficient multiparty computation (MPC) protocols, with the current plan being to leverage automata theory, regular expression and related algorithms.

Formally, Privacy Preserving Model Checking concerns the following problem: One party, the programmer $P$, has a finite transition system $M$ with $n$ states, which implicitly induces infinite traces viewable as either linear runs (LTL) or as a computation tree (CTL). A second party, the verifier $V$, has some properties they wish to check this software to satisfy which are expressed as a formula $\phi$ in a temporal logic $L$ of length $m$ logical operators and $m'$ atoms, the latter referring to whether a label is present on the states of the transition system.
These labels, of which there are $q$, have an interpretation agreed upon by the programmer and verifier in a manner beyond our concern. Finally, we have a public checking algorithm $\text{check}_L(M, \phi)$ which accepts if and only if the formula $\phi$ is satisfied by model $M$. Our goal is to provide a cryptographic protocol such that $\text{check}_L(M, \phi)$ may be computed correctly and efficiently — the latter by the standards of the checking algorithm without such privacy concerns — but with strict limits on the knowledge of $\phi$ learned by the programmer and the knowledge of $M$ learned by the verifier.

Secure multi-party computation (MPC) is a subfield of cryptography with the goal of designing protocols that allow computation of functions that take input from multiple parties yet without letting any party obtain knowledge about other participating parties besides the result of the computation. In this project, we leverage MPC protocols for PPMC as we can formulate model checking as a function that takes as input the program (transition system $M$) and property specifications (LTL-formula $\phi$), with the programmer and the verifier being the two participating parties. This way, the proof of security for any established MPC protocol would serve as a proof of security for our PPMC protocol that uses the MPC protocol.

Privacy in the context of cryptography is formally defined and verifiable, and in the final presentation of this project, we will present such proofs with proper rigor. For now, we explain it informally: a given protocol satisfies the privacy-preserving requirement of PPMC if after the two parties participated in the protocol, the programmer when given the verifier’s input $\phi$ can’t distinguish it from a randomly generated LTL formula of the same length using the same alphabet, and the verifier when given the programmer’s input $M$ can’t distinguish it from a randomly generated transition system with the same number of states and using the same alphabet.

2 The Plan

1. by Sep 21, finishing all the reading currently have: Ning’s slides and writeup, Model checking chapters, automata & regex material by Timos and on the internet.

2. by Oct 6, adapt existing algorithms for LTL model checking to the PPMC setting and do a survey of standard reachability algorithm in the MPC setting.

3. by Oct 13, formulate the problem of model checking in the context of automata & regex in the setting of MPC.

4. by Oct 27, present possible alternatives to garbled circuits in the MPC protocol used
in the PPMC setting for better time complexity, as well as the informal proofs.

5. by Nov 10, apply the above theoretical results to implementation: modify existing model checkers and check the modified version against benchmarks.

6. rest of semester, buffer time for the above plan, if all goes well, can implement code. And of course, write a thesis.

**Deliverables:** theoretical results + writeup + some program simulation/model checker improvement.