High-Assurance Cryptographic Software for Post-Quantum and Distributed Secure Computation

Abstract. Computer Aided Cryptography (CAC) aims to develop tools for the analysis and implementation of cryptographic protocols, including specification and implementation languages, compilers and formal verification tools. The formal verification dimension opens the way for High-Assurance Cryptographic Software. The objective of this work is to progress beyond the state of the art at both the foundational and applied research levels by developing new extensions to the Jasmin tool-chain and demonstrating its applicability to concrete examples that can have real-world impact. A recent SoK paper [**SoK**] published at IEEE S&P 2021 summarises challenges in this area; two of these challenges serve as our main motivation: 1) removing scalability bottlenecks in the analysis of distributed cryptographic protocols and 2) enabling the transition to post-quantum cryptography. This work will focus on concrete cryptographic constructions and implementations of post-quantum cryptography (PQC) and secure distributed computation that can serve as inputs to standardisation processes or as contributions to widely deployed cryptographic libraries and protocol software stacks.

Keywords: High-Assurance Cryptographic Software · Program Verification · Post-Quantum Cryptography · Secure Distributed Computation.

- ODS 10: Reduce the inequalities inside and between countries.
- ODS 16: Promote pacific and inclusive societies aiming at a sustainable development.
- ODS 17: Strengthen the implementation and revitalize the global partnership for sustainable development.

State of the art

Formal methods progressed to an impressive level of maturity, and several tools for systematically preventing entire classes of bugs in crypto software now exist. Frameworks like F* [**FSTR**], EasyCrypt [**EC**], and Coq [**FIAT**] are used to verify high-performance cryptographic code written in C and assembly. Tools like CryptoVerif [1] and EasyCrypt are used to verify the correctness of crypto security proofs. In practice, protocol stacks for TLS [**TLS**], Signal [7], Key Management [**KMS**] and crypto standards [**SHA3**, **HACL**] have been verified with these tools.

This work focuses on the Jasmin language. Jasmin [**JASM**] is a programming language designed to allow assembly in the head (a mixture of high-level and low-level, platform-specific, programming); it is supported by a formally verified (certified in Coq), predictable, compiler which empowers programmers to write highly efficient fine-tuned code. The generated (verified) assembly code matches the performance of the best implementations for this primitive. Jasmin code can be proved correct and secure via an equivalence proof to a high-level specification in EasyCrypt: the Jasmin compiler is able to create an EasyCrypt translation of its source. End-to-end security and correctness follow from the certification of the Jasmin compiler (source-to-assembly) and from the EasyCrypt proof (source-to-spec).

Our experience using EasyCrypt and Jasmin [**SoK**] shows that both foundational and applied research are needed to tackle two classes of cryptographic protocols that are within our reach: 1) post-quantum cryptography (PQC) and 2) secure distributed (multiparty) computation. We now explain why this is the case.

PQC should not be confused with quantum computing or quantum cryptography; its goal is to create cryptographic schemes that can be used today and resist potential quantum attacks in the future. PQC relies on different maths abstractions and computational assumptions than classical crypto such as lattice-based [3] and isogeny-based assumptions [9]. Implementing such primitives raises new challenges for the Jasmin framework; e.g., we have not yet considered rejection sampling mechanisms, which are crucial in PQC.

Orthogonal challenges arise in the verification of interactive protocols in Jasmin and EasyCrypt. Each cryptographic primitive is proved secure in a security model that captures its use in the real world. Primitives such as key exchange have been proved secure in a variety of models [6], with surprising complexity for two-party protocols: the goal is to capture concurrent executions and deal with composition within the security proof itself. General approaches to composability [2] exist, but these exclude some of the more efficient instantiations, or require ad-hoc adaptations to capture the associated caveats.

HACSpec [5] is a common specification language for cryptography that can be used by technical standards, software developers, and feed formal verification tools. Syntactically, HACSpec is a subset of Rust, and hence is familiar to developers; most importantly for standards, specifications are executable, they can be tested for correctness and interoperability, and to generate test vectors. HACSpec comes with a translator tool that can feed various formal verification frameworks. This proposal will contribute to the HACSpec development and ensure that the extensions we make to Jasmin can have a greater impact via integration into the HACSpec tool-chain.

1 Objectives

The goal of this proposal is to progress beyond the SotA at both the foundational and applied research levels by 1) developing new extensions to the HACSpec, and Jasmin tool- chain, and 2) demonstrating its applicability to concrete examples that can have real-world impact.

We will focus on concrete cryptographic constructions and implementations of PQC and secure distributed computation that can serve as inputs to standardisation or as contributions to widely deployed cryptographic libraries and software stacks. These use-cases are just outside the reach of existing CAC tool-chains, as described in the SotA analysis.

In what follows, we explain the approach that we will adopt to change this state of affairs by working in the concrete setting of the HACSpec-Jasmin tool-chain to reduce scalability bottlenecks in the analysis of distributed cryptographic protocols and enable the transition to post-quantum cryptography.

To fulfill our goals this proposal is divided into four main tasks:

- **T1**: Integration and Interoperability with External Tools
- **T2**: Formal Reasoning About Secure Distributed Computation
- **T3**: Computer-Aided Post-Quantum Cryptography
- **T4**: Use-Case Implementation and Tool-Chain Validation

References

[1] B. Blanchet. “Composition Theorems for CryptoVerif and Application to TLS 1.3.” In: (2018).

[2] R. Canetti. “Universally Composable Security.” In: (2020).

[3] “CRYSTALS: Cryptographic Suite for Algebraic Lattices.” In: (2021).

[4] A. Paskevich J.-C. Filliâtre. “Why3 - Where Programs Meet Provers.” In: (2013).

[5] P.-Y. Strub K. Bhargavan F. Kiefer. “HACSpec: Towards Verifiable Crypto Standards.” In: (2018).

[6] P. Rogaway M. Bellare D. Pointcheval. “Authenticated Key Exchange Secure against Dictionary Attacks.” In: (2000).

[7] B. Blanchet N. Kobeissi K. Bhargavan. “Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach.” In: (2017).

[8] M. Varia R. Canetti A. Stoughton. “EasyUC: Using EasyCrypt to Mechanize Proofs of Universally Composable Security.” In: (2019).

[9] “SIKE: Supersingular Isogeny Key Encapsulation.” In: (2021).