Signal Shot: End-to-End Formal Verification of Signal's Cryptographic Stack

Liao Zhang and the BAIF team

Beneficial AI Foundation

1 Introduction

The Signal application underpins end-to-end encryption message sharing for over 70 million users. Its cryptographic stack may contain subtle bugs, which can leak the users' information. Formal verification of Signal's cryptography is helpful to guarantees the correctness of Signal beyond finite test cases, giving users unconditional assurance. However, verifying a large and actively evolving Rust cryptographic codebase is highly challenging. The underlying mathematics of cryptography is complex and requires significant expertise to formalize. Moreover, formal verification is inherently labor-intensive. In addition, ensuring the long-term maintainability of the codebase requires that the corresponding verification artifacts remain modular and maintainable.

We determine to start the project Signal Shot to address these challenges and employ AI to discover an efficient infrastructure for program verification. It leverages Lean [3] as the proof assistant and the Rust-to-Lean transpiler Aeneas [6] to establish a verifiable connection between formal security models and Signal's production codebase. Given the rapid progress of AI across multiple domains, we also investigate how to effectively integrate AI into real-world formal verification workflows.

Contributions. First, we present the ongoing effort of Signal Shot that unifies protocol-level security reasoning with function-level correctness in a Lean-based framework. Second, we have completed a large-scale formal verification of the `curve25519-dalek` library, a widely deployed Rust implementation of Curve25519 operations. Finally, we have developed specialized coding agents for formal verification and have constructed a benchmark with ground-truth specifications and proofs for evaluating AI models on real-world program verification tasks.

2 Technical Approach of Signal Shot

The verification effort is organized along two complementary tracks: *protocol correctness* and *function correctness*.

Protocol correctness. Cryptographic protocols are designed to ensure the confidentiality and integrity of message exchange, typically relying on public-key and private-key mechanisms to prevent adversarial access to communicated data. We need to formalize Signal's core cryptographic protocols (X3DH [9], PQXDH [7], the Double Ratchet [1], and CKA [2]) in Lean by encoding their natural language descriptions from the literature as formal mathematical objects. Within this framework, standard security properties of these protocols are rigorously established, with all proofs mechanically verified by Lean's type-theoretic kernel.

Function correctness. We also need to verify that Signal's Rust implementations correctly realize developers' intentions. The verification pipeline proceeds in three stages. First, we transpile the production Rust source into a semantically equivalent representation in Lean. This step is automatically carried out by Aeneas, which models a substantial fragment of Rust's operational semantics in Lean and supports transpilation from Rust to Lean. Second, we need to define formal specifications for the extracted functions. Third, we need to prove that the extracted functions satisfy these specifications within Lean. Assuming that Aeneas preserves Rust semantics, these proofs transfer to the original Rust code.

3 Completed Verification of curve25519-dalek

We report the formal verification of curve25519-dalek, a core cryptographic library written in Rust that implements a set of Curve25519 group operations. The library comprises 288 functions and approximately 35,000 lines of Rust code implementing Curve25519 operations used in systems such as Signal, Tor, and Transport Layer Security, a widely deployed secure communication protocol. To the best of our knowledge, this constitutes the first end-to-end formal verification of an elliptic-curve cryptographic library at this scale using Lean.

The verification effort was carried out by two complementary teams. The division of two teams is due to historical reasons in our development. One team annotated the Rust source with compile-time Verus [8] specifications, whose proof obligations are discharged by SMT solvers. Another team used Aeneas to extract a semantically equivalent Lean representation and established its correctness. Together, these efforts cover the full stack, including finite-field and scalar arithmetic, elliptic-curve group operations, encoding layers, and protocol-level primitives. The process exposed several technical challenges, including large proof states generated by Aeneas, instability in SMT solving, and the complexity of elliptic-curve reasoning. The challenges have motivated several improvements to both Aeneas and Lean's infrastructure.

4 AI-Assisted Formal Verification and Benchmarking

We significantly benefit from the usage of AI in the verification of curve25519-dalek and will continue to investigate how to effectively apply AI to help researchers conduct formal verification using Lean. We found that modern AI such as coding agents is particular good at generating draft function specifications, leaving humans to correct minor errors in variable bounds. Another notable application of AI is the ability to decompose theorems into appropriate subgoals and generate proof sketches. An interesting observation is that most manual work becomes fixing function specifications generated by AI and confirming that AI has decomposed the original theorem into appropriate subgoals such that AI can successfully prove the left subgoals.

We have also developed our own formal verification agents, *FVS* [5], which have been practically deployed in our verification projects. FVS is implemented as a set of Claude Code skills, making it easy to install and use. We design specialized techniques tailored to formal verification within FVS. One notable example is the guidance for checking variable bounds, a subtle aspect that is particularly prone to errors in real-world program verification.

In addition, we construct a benchmark derived from our verified Lean codebase [4], providing ground-truth specifications and proofs for real-world cryptographic functions. This benchmark enables systematic evaluation of AI models across the realworld verification pipeline, from specification synthesis to proof completion.

5 Conclusion and Future Work

We have formally verified the curve25519-dalek library, which implements Curve25519 group operations widely used across numerous projects. We have also launched the Signal Shot project to verify Signal's cryptographic protocols. In addition, we design formal verification agents and construct a benchmark consisting of real-world program verification tasks.

Looking ahead, we first aim to establish the correctness of individual protocols and their underlying functions. We will then bridge these layers by connecting protocol-level verification with function-level correctness. Finally, we plan to verify the correctness of protocol composition.

References

[1] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: security notions, proofs, and modularization for the signal protocol. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–158. Springer, 2019.

[2] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In *Theory of Cryptography Conference*, pages 261–290. Springer, 2020.

[3] Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *International Conference on Automated Deduction*, pages 378–388. Springer, 2015.

[4] Beneficial AI Foundation. Dalek lean bench. https://github.com/Beneficial-AI-Foundation/dalek-lean-bench.

[5] Beneficial AI Foundation. Formal verification skills. https://github.com/Beneficial-AI-Foundation/formal-verification-skills.

[6] Son Ho and Jonathan Protzenko. Aeneas: Rust verification by functional translation. *Proceedings of the ACM on Programming Languages*, 6(ICFP):711–741, 2022.

[7] Ehren Kret and Rolfe Schmidt. The pqxdh key agreement protocol. *Signal*, 2023.

[8] Andrea Lattuada, Travis Hance, Jay Bosamiya, Matthias Brun, Chanhee Cho, Hayley LeBlanc, Pranav Srinivasan, Reto Achermann, Tej Chajed, Chris Hawblitzel, et al. Verus: A practical foundation for systems verification. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 438–454, 2024.

[9] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. *Open Whisper Systems*, 283(10):10, 2016.