Verified Post-Quantum Cryptography

Tom Arnold
tca4384@rit.edu

28 February 2022

Abstract

New cryptosystems are being developed and standardized to resist attacks from quantum computers; one such cryptosystem is the McEliece cryptosystem which owes its security to coding theory and binary Goppa codes. Implementation of the McEliece cryptosystem involves linear algebra, coding theory, and arithmetic with polynomials of binary Galois field elements.

Cryptosystems are one class of software where the absence of defects is highly desirable; this is where formal verification is useful. Formal verification can help prove mathematically the correct implementation of these systems.

We present a study of formal verification applied to the domain of post-quantum cryptography by implementing the McEliece cryptosystem in LiquidHaskell and verifying it using refinement types. As part of this study we implement all of the supporting mathematics required for the cryptosystem including linear algebra (vectors/matrices), coding theory, polynomial arithmetic, and binary Galois field arithmetic.

1 Introduction

Widely used public key cryptosystems are vulernable to attacks from quantum computers. These cryptosystems are based on one-way functions for which there is no efficient solution to calculate the inverse of the function; the security of the system is built on this fact. Quantum computing has introduced new algorithms which can efficiently compute solutions to some of these problems. As a result, cryptographers are working to implement new cryptosystems based on problems that are not efficiently solvable by quantum or digital computers [1].

The McEliece cryptosystem is one such system that dates back to the 70s when it was proposed by Robert McEliece. The system is based on coding theory and to break it an attacker would have to solve the general decoding problem which is NP-complete [5]. A modern variant of McEliece called Classic McEliece is one of the finalists in the NIST Post-Quantum standardization effort [2].

Formal verification can be used to prove the absense of bugs in a way that automated or manual testing cannot. Typically such verification is only done for software which must be held to a high standard. Cryptography is one such field where even simple errors can have severe real-world consequences [3].

In the following paper we show how a post-quantum cryptosystem (McEliece) can be implemented and formally verified using refinement types with LiquidHaskell [4]. We also examine the

effort involved in performing such verification so as to better understand the costs and benefits involved.

2 Post-Quantum Cryptography

Talk about PQC, NIST PQC competition, and the different variants of the McEliece cryptosystem.

3 Verification Techniques

Talk about Haskell and LiquidHaskell, refinement types and SMT solver, things that can be checked with refinements versus what cannot.

3.1 Static Types

3.2 Refinement Types

3.3 Size-Aware API

3.4 Termination Checking

4 Implementation

Walk through the project with code and expected output. Explain the theory for the backing math enough that someone can follow along. In particular go into detail on what is being verified at each stage and how. This is obviously not going to be a book on all of these topics, but it should read like a tutorial on how to do something similar.

4.1 Linear Algebra

4.1.1 Vectors

4.1.2 Matrices

4.1.3 Binary Matrix Inversion

4.1.4 Matrix Right-Inverse

4.2 Coding Theory

4.2.1 Hamming Codes

4.2.2 Binary Goppa Codes

4.3 Number Theory

4.3.1 Polynomials

4.3.2 Galois Fields

4.3.3 Extended Euclidean Algorithm (EEA)

4.3.4 Polynomial EEA

4.4 McEliece Cryptosystem

4.4.1 Key Generation

4.4.2 Encryption

4.4.3 Decryption

5 Related Work

Talk about related work that motivated this, most notably Project Everest (FStar and HACL). There are some major differences between this project and that (aside from scope). Also FStar has some capabilities that LiquidHaskell does not currently have.

6 Conclusion

Talk about how the implementation went, which parts were harder to verify, how much was verified, some

metrics (lines of code / refinements). Link to full source code on Bitbucket if appropriate.

References

[1] Risse, T. (2011). How SAGE Helps To Implement Goppa Codes and The McEliece Public Key Crypto System.

[2] Computer Security Division. Post-quantum cryptography: CSRC. Retrieved February 24, 2022, from https://csrc.nist.gov/Projects/post-quantum-cryptography

[3] Jean-Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. 2017. HACL*: A Verified Modern Cryptographic Library. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). Association for Computing Machinery, New York, NY, USA, 1789–1806. DOI:https://doi.org/10.1145/3133956.3134043

[4] Vazou, N., Seidel, E.L., Jhala, R., Vytiniotis, D., Peyton-jones, S. (2014). Refinement types for Haskell. ICFP 2014.

[5] McEliece, R.J. (1978). A public key cryptosystem based on algebraic coding theory.