Implementations of Benaloh Encryption Scheme in Cryptol and C++

Joseph Tafese

University of Waterloo
Waterloo, Ontario, CA
jetafese@uwaterloo.ca

Abstract. We present a verification framework that utilizes Z3, an SMT Solver, with Cryptol as the front-end language. An implementation is also provided in C++. This approach allows us to establish trust in our implementation, a significant advantage in a safety critical domain. Using existing tools, we compare the two programs and determine if they satisfy the same specification - a critical step in the journey to build verifiable elections.

Keywords: verifiable elections · cryptol · formal verification

1 Introduction

The Benaloh Voting Scheme [1] was designed to bring about the dawn of trusted elections. The underlying Benaloh Encryption scheme employs homomorphic encryption, a key ingredient for the scheme's success. Naturally, this has some significant implications for democratic structures ranging from boards to nation states. Therefore, the correct implementation of this, and many other mathematically verified schemes, becomes crucial to guarantee before their widespread adoption.

We will describe the languages we use in Section 2, implementation overview and challenges in Section 3 and conclude in Section 4.

2 Cryptol

Cryptol [2] is a domain-specific language primarily used for creating and verifying cryptographic algorithms. It offers a unique blend of features tailored for the efficient specification, implementation, and verification of cryptographic systems. This comes in the form of random and exhausing testing as well as formal verification that is built into the language. Tool support for generating executable code makes Cryptol a strong candidate for document reference implementations and prototypes.

To take advantage of the verificaiton features, a developer specifies what properties need to be preserved in the program. For example, given a property like `property addCommutative x y = x + y == y + x`, one can use the

`:check` command to automatically generate random test cases. If a more exhausive testing approach is required, the `:exhaust` command offers the best guarantee within some bounded domain. A formal verificain approach, providing mathematical certainty, can be achieved using the `:prove addCommutative` command, which checks whether addition is commutative over the specified data types. Cryptol's integration of these testing and verification methodologies that are tailored for cryptographic application provides a solid framework for our verification needs.

3 Implementation

The code we contributed [1] has a reference implementation of the Benaloh Cryptographic Scheme [3] in both Cryptol and C++. This approach is valuable since it allows us to establish a source-of-truth, a reference implementation that is proven correct in Cryptol, and compare its behaviour to an optimized implementation that is written in C++. Using this development framework provides a stronger guarantee of correctness for critical cryptographic implementations.

Cryptol. The goal of this implementation was to verify that the encryption and decryption scheme works as we would expect. Properties such as `property encryptDecryptIdentity y = decrypt (encrypt y)` are checked using random testing, exhaustive testing and proven to hold given sample prime numbers. For arbitrary primes, it is challenging to do this verification quickly without access to stronger machines and faster algorithms for primality testing. Furthermore, since the underlying tools are designed for undecidable fragments of First Order Logic, we do not have the ability to direct which engines are used by Cryptol.

C++. Using the standard cryptographic libraries and compilers, we get an executable binary that provides a concrete implementation of the functions we proved correct using Cryptol. The biggest challenge were ensuring proper memory management, modular operations and data encapsulation. Making use of early breaks and common programming paradigms was helpful in getting faster termination compared to the Cryptol implementation. Another challenge was with using existing automatic verification tools [2] with external libraries. Even though these tools are able to model functions as logical statements, the numerical properties that we need to verify were hard to scale for arbitrary primes.

4 Conclusion

Developing provably correct crytographic schemes is critical to the safety of our lives in the modern world. Ensuring that the systems we use correctly implement the proven schemes is even more important. Developing and verifying the

[1] https://github.com/jetafese/benalohCrypto/tree/main
[2] https://github.com/jetafese/seahorn

Benaloh Encryption Scheme within the proposed framework allows for an end-to-end pipeline that preserves the integrity of the final implementation. Even though this approach does not hinder against all types of attacks, it bridges the gap between a mathematically proven cryptographic scheme and its implementation.

References

1. Benaloh, J.: Simple verifiable elections. In: USENIX Workshop on Accurate Electronic Voting Technology (2006), https://api.semanticscholar.org/CorpusID:6760724
2. Browning, S.: Cryptol, a dsl for cryptographic algorithms (10 2010). https://doi.org/10.1145/1900160.1900171
3. Budiman, M.A., Rachmawati, D.: A tutorial on using benaloh public key cryptosystem to encrypt text. Journal of Physics: Conference Series **1542**(1), 012039 (may 2020). https://doi.org/10.1088/1742-6596/1542/1/012039, https://dx.doi.org/10.1088/1742-6596/1542/1/012039