Formal Verification of Post-Quantum Cryptography
NIST PQC Standardization Conference

Manuel Barbosa[1] mbb@fc.up.pt
Andreas Hülsing[2] andreas@huelsing.net
Matthias Meijers[2] m.c.f.h.p.meijers@student.tue.nl
Peter Schwabe[3] peter@cryptojedi.org

[1]University of Porto, [2]Eindhoven University of Technology,
[3]Max Planck Institute for Security & Privacy

In this talk we will discuss the state of the formal verification of the two NIST PQC standardization process submissions Kyber and Saber.

Formal Verification

Throughout most of history, the security properties of cryptographic schemes have been proven by means of hand-written security proofs. However, the innovation and development in the field of cryptography has led to a significant increase in the complexity of cryptographic schemes. As such, hand-written security proofs have become substantially more difficult to carry out correctly. In fact, multiple instances of security proofs exist which, despite the fact that they were extensively scrutinized and considered to be correct, turned out to be flawed. Even worse, in some of these cases, the corresponding cryptographic scheme was additionally found to be insecure [1]. These instances clearly exemplify the importance and difficulty of properly constructing and verifying a cryptographic scheme and its security proof.

In addition to the above concern, even if a cryptographic scheme and its security proof are completely correct, implementation errors may still invalidate any of the scheme's properties and guarantees. As a result, despite the scheme being completely sound at the design level, no effective security might be provided at all. Akin to the previous concern, ample examples exist of faulty implementations of sound cryptographic systems allowing for security compromises [2]. This signifies the importance of sound and secure implementations of cryptography.

In part to remedy the aforementioned issues, the field of computer-aided cryptography was established. This field of research seeks to develop approaches to the construction of cryptography that utilize computers to formally verify and guarantee any related correctness, efficiency and security claims. The utilization of computers in this manner significantly reduces the complexity of the manual labor required in verifying the security and correctness of a scheme and its implementations, while simultaneously providing a consistently high level of rigorousness. This enables the procurement of a higher level of confidence in the security and correctness of a cryptographic scheme and its implementations.

General Formal Verification Process

Recent progression in the field of computer-aided cryptography demonstrates that a general formal verification process can be applied to real-world cryptographic systems [3]. Particularly, given the scheme's specification and (hand-written) security proof, this process allows one to formally verify any scheme and its implementations. Additionally, the process does not depend on any concrete tools, provided they allow one to perform the desired verification task. Nevertheless, the choice of tool may significantly impact the difficulty of the process, depending on, for example, the properties to verify or the type of proof used. Abstractly, the process goes as follows. First, one formalizes the scheme's specification, security properties and security proof. Subsequently, the specified scheme is formally verified to possess the desired security properties by means of the formalized security proof.

Afterwards, an implementation of the scheme can be formally verified to be functionally correct with respect to the scheme's specification. As such, the implementation is shown to correspond to a specification that, in turn, is verified to possess certain desirable properties, implying the implementation also possesses these properties. Finally, depending on the available and utilized tools, several other properties of the implementation can be verified. Examples of such properties are memory-safety and constant-time behaviour.

EasyCrypt

EasyCrypt is a tool predominantly aimed at formally verifying the security properties of cryptographic constructions [4, 5]. To this end, the tool adopts the code-based approach to provable security; that is, security properties and hardness assumptions are modeled as probabilistic programs. Moreover, the tool's higher-order ambient logic, standard library, and built-in mechanisms allow for, among others, extensive mathematical reasoning, different types of proofs (e.g., game-playing and simulation-based), and modular construction of cryptographic systems.

Albeit more conveniently applied at the design level, EasyCrypt can be employed both at the design and the implementation level. Especially with the development of frameworks such as Jasmin, utilizing EasyCrypt to verify the functional correctness and constant-time properties of concrete implementations is made significantly less complex.

At the time of writing, EasyCrypt does not yet allow for analysis considering quantum adversaries. Nevertheless, an ongoing project is attempting to implement the support for such analysis.

Jasmin

Jasmin is a framework designed for implementing high-assurance and high-speed cryptography [6]. The framework comprises a programming language, a compiler, and several tools for (partially) automated verification of desirable program properties. In particular, the framework's tools assist the developer in formally proving a Jasmin implementation is memory-safe, constant-time, and functionally correct. Here, contrary to the tool for memory-safety, the tools for constant-time and functional correctness are not fully automated; as such, verifying these properties still requires some manual labor, although this effort is minimal for the constant-time property. Specifically, given a Jasmin implementation, these tools generate EasyCrypt code aimed at verifying their respective property; subsequently, this code can be used to actually verify the considered property in EasyCrypt.

Current PQC Projects

This presentation will present the progress of two formal verification projects of PQC competition finalists: one for Kyber and one for Saber. The eventual goal of both projects is identical and twofold; specifically, both projects aim to (1) formally verify the IND-CCA2 security of (the specification of) their respective KEM and (2) formally verify the functional correctness, memory-safety, and constant-time properties of a reference and an optimized implementation of their respective KEM. For this purpose, both projects use EasyCrypt and Jasmin.

At the time of writing, the progress of both projects is as follows. For both Kyber and Saber, a reference and optimized implementation have been constructed in Jasmin. In addition, the Kyber project has formally verified the IND-CPA property and (1 - δ)-correctness bound of the (specification of the) PKE, the functional correctness and memory-safety of the reference implementation, and the memory-safety of the optimized implementation. The Saber project has formally verified the IND-CPA property of the PKE scheme, and the memory-safety of both implementations.

Although it seems unfortunate that analysis considering quantum adversaries is not yet possible in EasyCrypt, formal verification in the classical setting already provides valuable insight into these schemes and their properties. Additionally, in recent work, Unruh formally verified the Fujisaki-Okamoto transform in his qRHL-tool [7]. As such, building on this work, verifying the schemes in the classical setting also increases the confidence in the correctness and security of the schemes when considering quantum adversaries.

References

[1] N. Koblitz and A. Menezes, "Critical perspectives on provable security: Fifteen years of "another look" papers." Cryptology ePrint Archive, Report 2019/1336, 2019. `https://eprint.iacr.org/2019/1336`.

[2] D. Lazar, H. Chen, X. Wang, and N. Zeldovich, "Why does cryptographic software fail? a case study and open problems," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, APSys '14, (New York, NY, USA), Association for Computing Machinery, 2014.

[3] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "Sok: Computer-aided cryptography." Cryptology ePrint Archive, Report 2019/1393, 2019. `https://eprint.iacr.org/2019/1393`.

[4] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, *EasyCrypt: A Tutorial*, pp. 146–166. Cham: Springer International Publishing, 2014.

[5] G. Barthe, F. Dupressoir, B. Grégoire, B. Schmidt, and P. Strub, "Computer-aided cryptography: some tools and applications," 2014.

[6] J. B. Almeida, M. Barbosa, G. Barthe, A. Blot, B. Grégoire, V. Laporte, T. Oliveira, H. Pacheco, B. Schmidt, and P.-Y. Strub, "Jasmin: High-assurance and high-speed cryptography," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), p. 1807–1823, Association for Computing Machinery, 2017.

[7] D. Unruh, "Post-quantum verification of fujisaki-okamoto," in *Advances in Cryptology – ASIACRYPT 2020* (S. Moriai and H. Wang, eds.), (Cham), pp. 321–352, Springer International Publishing, 2020.