Formal methods - fun for academics or a realistic tool for industry and governments?

Andreas Hülsing
Eindhoven University of Technology & SandboxAQ

Spoiler: It's both

The title was assigned to me :)

I am biased

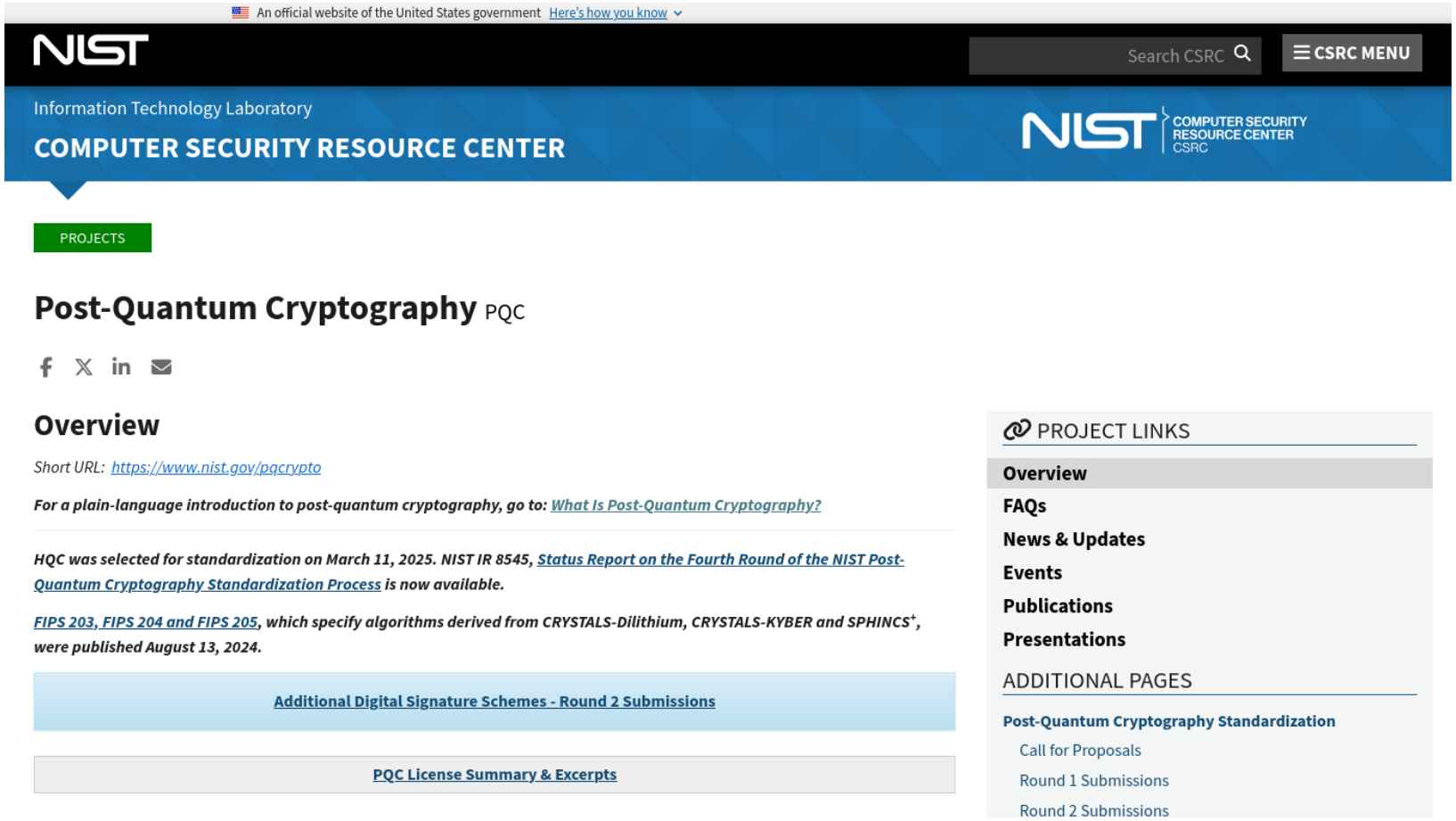
- Background in "provable security"
- I am a user of the tools!
- EasyCrypt user
- Member of Formosa crypto
- Project with Cryspen in HAX toolchain

Act I: A realistic tool for industry and governments

(the why)

How to build trust in cryptography?

Run a competition



Ask for security proofs

Von Mikhail Kudinov

Antworten | Liste antworten | Weiterleiten | Archivieren | Junk | Löschen | Mehr

An pqc-forum@list.nist.gov 23-07-2020 17:09

Betreff **[pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+**

List-ID <pqc-forum.list.nist.gov>

Dear all,

In this comment, we would like to point out a flaw of existing security proofs of the SPHINCS+ hash-based scheme. Particularly, we would like to pay attention to security proofs of the underlying WOTS+ scheme with preimage resistance (PRE) requirement replaced by second preimage resistance (SPR) + "at least two preimages for every image" requirements [see eq. (14) in Round 2 submission] or decisional second preimage resistance (DSPR) + SPR requirements [see Bernstein et al. "The SPHINCS+ signature framework" 2019].

Both of these approaches are based on the claim that in the case where the given image has several preimages under some cryptographic hash function, the original preimage is information-theoretically hidden among all preimages (see "Case 2" in the Proof of Theorem 2 in [Hülsing et al. "Mitigating Multi-Target Attacks in Hash-based Signatures" 2016] and "SM-DSPR success probability" in the proof of Claim 23 in [Bernstein et al. "The SPHINCS+ signature framework" 2019]). Though this claim is quite reasonable in the case of a single hash function query, the situation becomes much more complicated when one deals with a chain of hash functions like in the WOTS+ scheme.

Let h_i with i=1,...,w-1 be a hash function used to obtain a value at i'th level of the WOTS+ scheme from the one at (i-1)'th level. That is pk_j = h_{w-1}(h_{w-2}(... h_1(sk_j) ...)), where sk_j and pk_j are elements of secret and public key respectively and w is a Winternitz parameter (commonly w = 16). Here we assume that all bitmasks are included in h_i. Let IMG_i be an image set of h_i, and let PREIMG_i(y) be a set of all preimages for given y taken from IMG_i. The proposed security proofs are based either on assumption that for each y one has |PREIMG_i(y)| > 1, or that it is computationally hard to recognize whether |PREIMG_i(y)| = 1 or not. The latter is called a DSPR property [D.J. Bernstein, A. Hülsing "Decisional second-preimage resistance: When does SPR imply PRE?" 2019].

Consider the set WOTS_IMG_i = h_i(h_{i-1}(... h_1({0,1}^n) ...)) that is an image set of the whole WOTS+ chain up to level i from a set of all possible secret keys {0,1}^n (n is a security parameter, typically equals to 256). One can reasonably expect that for a secure hash function built in the chain functions and i > 1, |WOTS_IMG_i| < |IMG_i| because of collisions at levels 1, ... , i-1. Let WOTS_PREIMG_i(y) be a set of preimages of y under h_i belonging to WOTS_IMG_{i-1}. Having a Challenger's signature, a WOTS+-breaking adversary is able to choose a position in the chain where |WOTS_PREIMG_i(y)| = 1, even though |PREIMG_i(y)| > 1 for some known element y in the WOTS+ structure. In the result, the adversary manages to forge a signature avoiding breaking SPR property (because the forgery consists of the same element used by the Challenger), and by choosing elements having |PREIMG_i(y)| > 1 or |PREIMG_i(y)| = 1 with a proper probability, avoiding breaking DSPR property. Thus the reduction proof fails.

We note that the security proof of the original SPHINCS scheme [Bernstein et al. "SPHINCS: practical stateless hash-based signature" 2015] which is based on PRE+SPR+undetectability (UD) assumptions does not have this flaw, though shows lower security level for the same scheme parameters. We also note that the updated detailed security proof of the WOTS+ scheme based on PRE+SPR+UD assumptions can be found in https://arxiv.org/abs/2002.07419.

With kind regards,
Mikhail Kudinov, Evgeniy Kiktenko, Aleksey Fedorov
Russian Quantum Center (www.rqc.ru) and QApp (www.qapp.tech)

Fixing and Mechanizing the Security Proof of Fiat-Shamir with Aborts and Dilithium

Manuel Barbosa[1], Gilles Barthe[2], Christian Doczkal[2], Jelle Don[3], Serge Fehr[3,4], Benjamin Grégoire[5], Yu-Hsuan Huang[3], Andreas Hülsing[6], Yi Lee[2,7], and Xiaodi Wu[7]

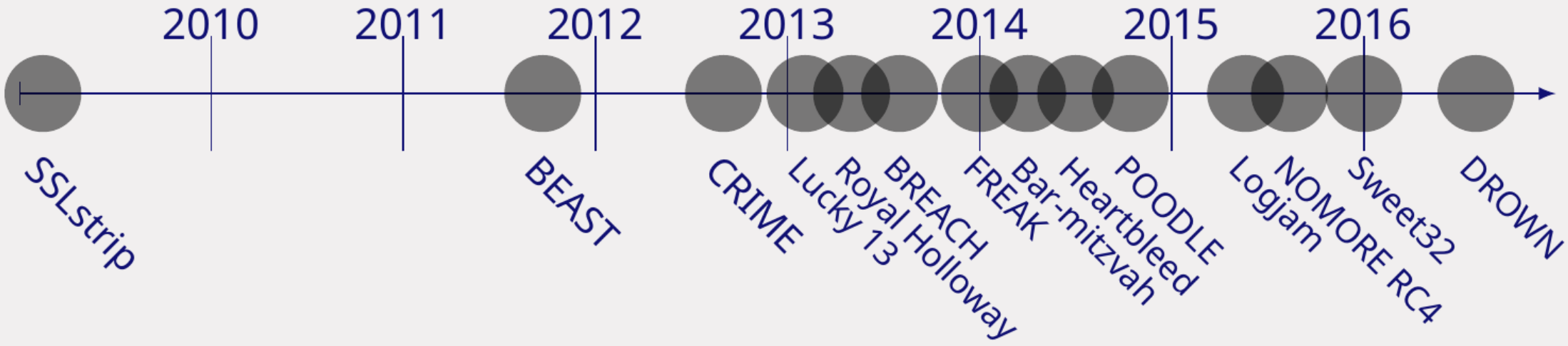
[1] University of Porto (FCUP) and INESC TEC, Portugal
[2] Max Planck Institute for Security and Privacy, Germany
[3] Centrum Wiskunde & Informatica, The Netherlands
[4] Leiden University, The Netherlands
[5] Inria Centre at Université Côte d'Azur
[6] Eindhoven University of Technology
[7] University of Maryland, United States

mbb@fc.up.pt, {gilles.barthe, christian.doczkal}@mpi-sp.org, {jelle.don, serge.fehr, yhh}@cwi.nl, benjamin.gregoire@inria.fr, andreas@huelsing.net, {ylee1228, xiaodiwu}@umd.edu

Abstract. We extend and consolidate the security justification for the Dilithium signature scheme. In particular, we identify a subtle but crucial gap that appears in several ROM and QROM security proofs for signature schemes that are based on the Fiat-Shamir with aborts paradigm, including Dilithium. The gap lies in the CMA-to-NMA reduction and was uncovered when trying to formalize a variant of the QROM security proof by Kiltz, Lyubashevsky, and Schaffner (Eurocrypt 2018). The gap was confirmed by the authors, and there seems to be no simple patch for it. We provide new, fixed proofs for the affected CMA-to-NMA reduction, both for the ROM and the QROM, and we perform a concrete security analysis for the case of Dilithium to show that the claimed security level is still valid after addressing the gap. Furthermore, we offer a fully mechanized ROM proof for the CMA-security of Dilithium in the EasyCrypt proof assistant. Our formalization includes several new tools and techniques of independent interest for future formal verification results.

What if you are not NIST?

Timeline of attacks on SSL/TLS



⟵ 1998 Bleichenbacher's padding oracle attack
⟵ 2002 (Symmetric) Padding Oracle

The secret TETRA primitives and their security

See https://www.midnightblue.nl/research/tetraburst

~~ISO/IEC 19772:2009~~

Information technology — Security techniques — Authenticated encryption

Withdrawn (Edition 1, 2009)
→ New version available: ISO/IEC 19772:2020

Abstract

ISO/IEC 19772:2009 specifies six methods for authenticated encryption, i.e. defined ways of processing a data string with the following security objectives: data confidentiality, i.e. protection against unauthorized disclosure of data; data integrity, i.e. protection that enables the recipient of data to verify that it has not been modified; data origin authentication, i.e. protection that enables the recipient of data to verify the identity of the data originator. All six methods specified in ISO/IEC 19772:2009 require the originator and the recipient of the protected data to share a secret key. Key management is outside the scope of ISO/IEC 19772:2009; key management techniques are defined in ISO/IEC 11770.

General information

Status : Withdrawn
Publication date : 2009-02
Stage : Withdrawal of International Standard [95.99]

Edition : 1
Number of pages : 29

Technical Committee : ISO/IEC JTC 1/SC 27
ICS : 35.030

RSS updates

These attacks were found during deployment!

Ever reviewed a proof for a cryptographic protocol?

On the Tight Security of TLS 1.3: Theoretically-Sound Cryptographic Parameters for Real-World Deployments

Denis Diemert and Tibor Jager*

University of Wuppertal, Germany
{denis.diemert, tibor.jager}@uni-wuppertal.de

Abstract

We consider the *theoretically-sound* selection of cryptographic parameters, such as the size of algebraic groups or RSA keys, for TLS 1.3 in practice. While prior works gave security proofs for TLS 1.3, their security loss is *quadratic* in the total number of sessions across all users, which due to the pervasive use of TLS is huge. Therefore, in order to deploy TLS 1.3 in a theoretically-sound way, it would be necessary to compensate this loss with unreasonably large parameters that would be infeasible for practical use at large scale. Hence, while these previous works show that in principle the design of TLS 1.3 is secure in an asymptotic sense, they do not yet provide any useful *concrete* security guarantees for real-world parameters used in practice.

In this work, we provide a new security proof for the cryptographic core of TLS 1.3 in the random oracle model, which reduces the security of TLS 1.3 *tightly* (that is, with constant security loss) to the (multi-user) security of its building blocks. For some building blocks, such as the symmetric record layer encryption scheme, we can then rely on prior work to establish tight security. For others, such as the RSA-PSS digital signature scheme currently used in TLS 1.3, we obtain at least a *linear* loss in the number of users, independent of the number of sessions, which is much easier to compensate with reasonable parameters. Our work also shows that by replacing the RSA-PSS scheme with a tightly-secure scheme (e. g., in a future TLS version), one can obtain the first fully tightly-secure TLS protocol.

Our results enable a theoretically-sound selection of parameters for TLS 1.3, even in large-scale settings with many users and sessions per user.

TLS 1.3: ~36 pages
prelim, model & proof
(full width A4)

A Cryptographic Analysis of the WireGuard Protocol

Benjamin Dowling and Kenneth G. Paterson

Information Security Group
Royal Holloway, University of London
benjamin.dowling@rhul.ac.uk, kenny.paterson@rhul.ac.uk

Abstract. WireGuard (Donenfeld, NDSS 2017) is a recently proposed secure network tunnel operating at layer 3. WireGuard aims to replace existing tunnelling solutions like IPsec and OpenVPN, while requiring less code, being more secure, more performant, and easier to use. The cryptographic design of WireGuard is based on the Noise framework. It makes use of a key exchange component which combines long-term and ephemeral Diffie-Hellman values (along with optional preshared keys). This is followed by the use of the established keys in an AEAD construction to encapsulate IP packets in UDP. To date, WireGuard has received no rigorous security analysis. In this paper, we, rectify this. We first observe that, in order to prevent Key Compromise Impersonation (KCI) attacks, any analysis of WireGuard's key exchange component must take into account the first AEAD ciphertext from initiator to responder. This message effectively acts as a key confirmation and makes the key exchange component of WireGuard a 1.5 RTT protocol. However, the fact that this ciphertext is computed using the established session key rules out a proof of session key indistinguishability for WireGuard's key exchange component, limiting the degree of modularity that is achievable when analysing the protocol's security. To overcome this proof barrier, and as an alternative to performing a monolithic analysis of the entire WireGuard protocol, we add an extra message to the protocol. This is done in a minimally invasive way that does not increase the number of round trips needed by the overall WireGuard protocol. This change enables us to prove strong authentication and key indistinguishability properties for the key exchange component of WireGuard under standard cryptographic assumptions.

1 Introduction

WireGuard: WireGuard [11] was recently proposed by Donenfeld as a replacement for existing secure communications protocols like IPsec and OpenVPN. It has numerous benefits, not least its simplicity and ease of configuration, high performance in software, and small codebase. Indeed, the protocol is implemented in less than 4,000 lines of code, making it relatively easy to audit compared to large, complex and buggy code-bases typically encountered with IPsec and SSL/TLS (on which OpenVPN is based).

From a networking perspective, WireGuard encapsulates IP packets in UDP packets, which are then further encapsulated in IP packets. This is done carefully so as to avoid too much packet overhead. WireGuard also offers a highly simplified version of IPsec's approach to managing which security transforms get applied to which packets: essentially, WireGuard matches on IP address ranges and associates IP addresses with static Diffie-Hellman keys. This avoids much of the complexity associated with IPsec's Security Associations/Security Policy Database mechanisms.

From a cryptographic perspective, WireGuard presents an interesting design. It is highly modular, with a key exchange phase, called the handshake, that is presented as being clearly separated from the subsequent use of the

WireGuard: ~16 pages
JUST proof (full width
A4) - several small
mistakes

The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol

Joël Alwen*
Wickr Inc.
jalwen@wickr.com

Sandro Coretti†
New York University
coretti@nyu.edu

Yevgeniy Dodis‡
New York University
dodis@cs.nyu.edu

February 21, 2020

Abstract

Signal is a famous secure messaging protocol used by billions of people, by virtue of many secure text messaging applications including Signal itself, WhatsApp, Facebook Messenger, Skype, and Google Allo. At its core it uses the concept of "double ratcheting," where every message is encrypted and authenticated using a fresh symmetric key; it has many attractive properties, such as forward security, post-compromise security, and "immediate (no-delay) decryption," which had never been achieved in combination by prior messaging protocols.

While the formal analysis of the Signal protocol, and ratcheting in general, has attracted a lot of recent attention, we argue that none of the existing analyses is fully satisfactory. To address this problem, we give a clean and general definition of *secure messaging*, which clearly indicates the types of security we expect, including forward security, post-compromise security, and immediate decryption. We are the first to explicitly formalize and model the immediate decryption property, which implies (among other things) that parties seamlessly recover if a given message is permanently lost—a property not achieved by any of the recent "provable alternatives to Signal."

We build a modular "generalized Signal protocol" from the following components: (a) *continuous key agreement (CKA)*, a clean primitive we introduce and which can be easily and generically built from public-key encryption (not just Diffie-Hellman as is done in the current Signal protocol) and roughly models "public-key ratchets;" (b) *forward-secure authenticated encryption with associated data (FS-AEAD)*, which roughly captures "symmetric-key ratchets;" and (c) a two-input hash function that is a pseudorandom function (resp. generator with input) in its first (resp. second) input, which we term PRF-PRNG. As a result, in addition to instantiating our framework in a way resulting in the existing, widely-used Diffie-Hellman based Signal protocol, we can easily get post-quantum security and not rely on random oracles in the analysis.

Signal: ~56 pages (full
width A4) proof +
model

So, if proofs are sound we are done?

Algorithm vs specification

- Algorithmic description

Algorithm 2 Kyber.CPA.Enc($pk = (\mathbf{t}, \rho), m \in \mathcal{M}$): encryption

1: $r \leftarrow \{0,1\}^{256}$
2: $\mathbf{t} := \mathsf{Decompress}_q(\mathbf{t}, d_t)$
3: $\mathbf{A} \sim R_q^{k \times k} := \mathsf{Sam}(\rho)$
4: $(\mathbf{r}, \mathbf{e}_1, e_2) \sim \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \mathsf{Sam}(r)$
5: $\mathbf{u} := \mathsf{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1, d_u)$
6: $v := \mathsf{Compress}_q\left(\mathbf{t}^T\mathbf{r} + e_2 + \left\lceil \frac{q}{2} \right\rfloor \cdot m, d_v\right)$
7: **return** $c := (\mathbf{u}, v)$

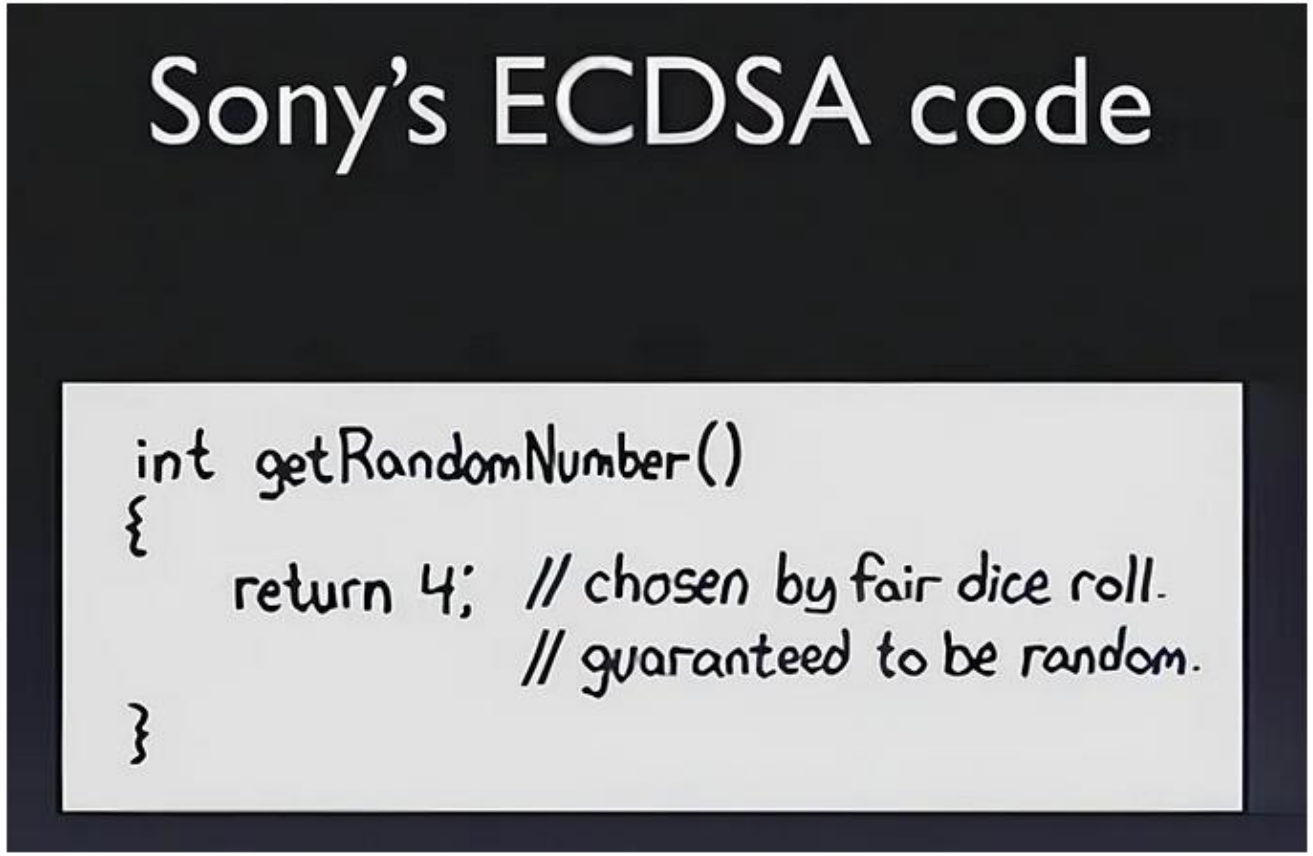
- Specification

Algorithm 5 KYBER.CPAPKE.Enc(pk, m, r): encryption

Input: Public key $pk \in \mathcal{B}^{d_t \cdot k \cdot n/8+32}$
Input: Message $m \in \mathcal{B}^{32}$
Input: Random coins $r \in \mathcal{B}^{32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$

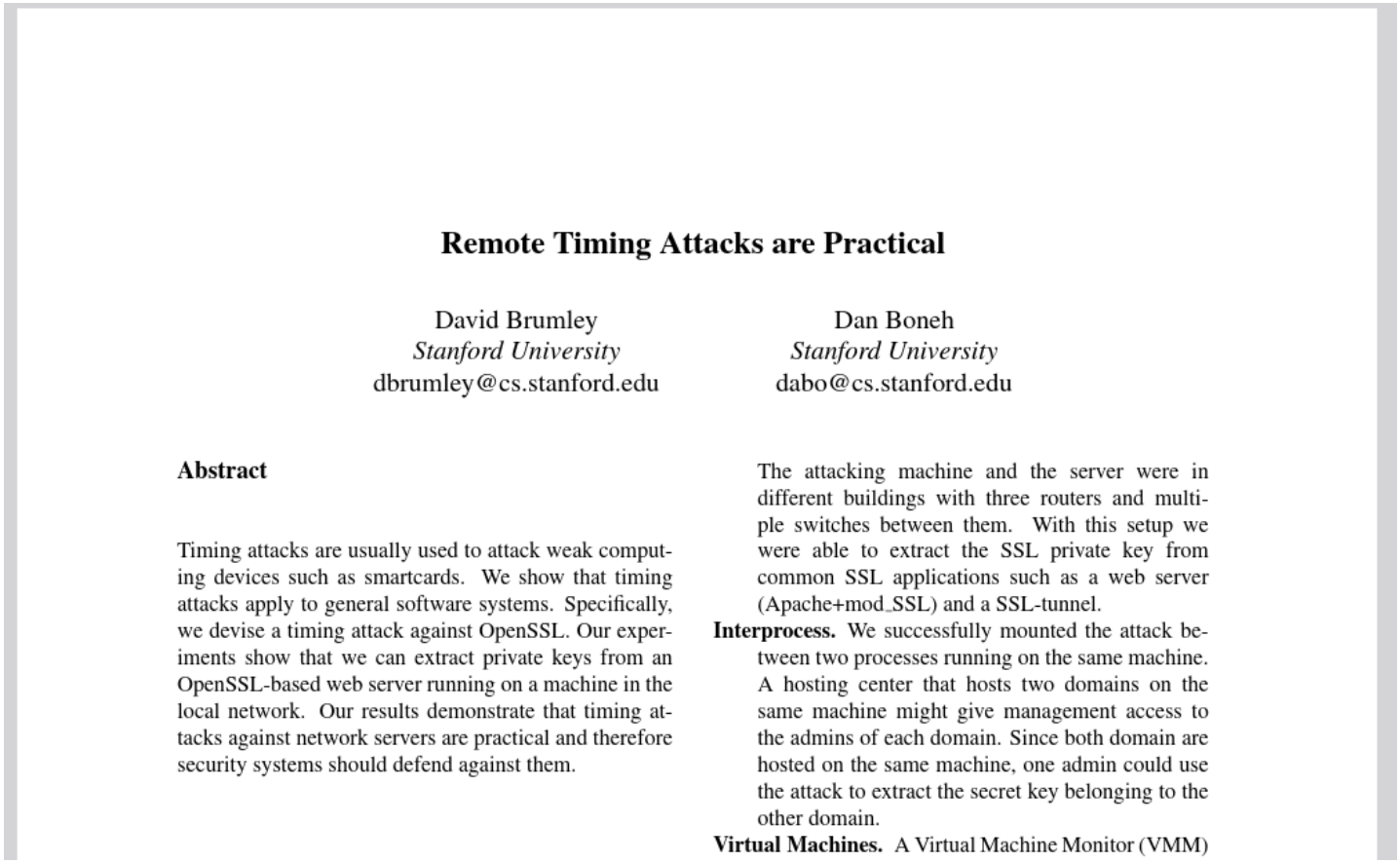
1: $N := 0$
2: $\mathbf{t} := \mathsf{Decompress}_q(\mathsf{Decode}_{d_t}(pk), d_t)$
3: $\rho := pk + d_t \cdot k \cdot n/8$
4: **for** i from 0 to $k-1$ **do** ▷ Generate matrix $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
5: **for** j from 0 to $k-1$ **do**
6: $\hat{\mathbf{A}}^T[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho \| i \| j))$
7: **end for**
8: **end for**
9: **for** i from 0 to $k-1$ **do** ▷ Sample $\mathbf{r} \in R_q^k$ from B_η
10: $\mathbf{r}[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$
11: $N := N + 1$
12: **end for**
13: **for** i from 0 to $k-1$ **do** ▷ Sample $\mathbf{e}_1 \in R_q^k$ from B_η
14: $\mathbf{e}_1[i] := \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$
15: $N := N + 1$
16: **end for**
17: $e_2 := \mathsf{CBD}_\eta(\mathsf{PRF}(r, N))$ ▷ Sample $e_2 \in R_q$ from B_η
18: $\hat{\mathbf{r}} := \mathsf{NTT}(\mathbf{r})$
19: $\mathbf{u} := \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ ▷ $\mathbf{u} := \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$
20: $v := \mathsf{NTT}^{-1}(\mathsf{NTT}(\mathbf{t})^T \circ \hat{\mathbf{r}}) + e_2 + \mathsf{Decode}_1(\mathsf{Decompress}_q(m, 1))$ ▷ $v := \mathbf{t}^T\mathbf{r} + e_2 + \mathsf{Decompress}_q(m, 1)$
21: $c_1 := \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\mathbf{u}, d_u))$
22: $c_2 := \mathsf{Encode}_{d_v}(\mathsf{Compress}_q(v, d_v))$
23: **return** $c = (c_1 \| c_2)$ ▷ $c := (\mathsf{Compress}_q(\mathbf{u}, d_u), \mathsf{Compress}_q(v, d_v))$

Inconsistent standards implementation



Source: https://deeprnd.medium.com/decoding-the-playstation-3-hack-unraveling-the-ecdsa-random-generator-flaw-e9074a51b831

Side-channels



Remote Timing Attacks are Practical

David Brumley
Stanford University
dbrumley@cs.stanford.edu

Dan Boneh
Stanford University
dabo@cs.stanford.edu

Abstract

Timing attacks are usually used to attack weak computing devices such as smartcards. We show that timing attacks apply to general software systems. Specifically, we devise a timing attack against OpenSSL. Our experiments show that we can extract private keys from an OpenSSL-based web server running on a machine in the local network. Our results demonstrate that timing attacks against network servers are practical and therefore security systems should defend against them.

The attacking machine and the server were in different buildings with three routers and multiple switches between them. With this setup we were able to extract the SSL private key from common SSL applications such as a web server (Apache+mod_SSL) and a SSL-tunnel.

Interprocess. We successfully mounted the attack between two processes running on the same machine. A hosting center that hosts two domains on the same machine might give management access to the admins of each domain. Since both domain are hosted on the same machine, one admin could use the attack to extract the secret key belonging to the other domain.

Virtual Machines. A Virtual Machine Monitor (VMM)

SPECTRE

MELTDOWN

KyberSlash: Exploiting secret-dependent division timings in Kyber implementations

Daniel J. Bernstein[1,2], Karthikeyan Bhargavan[3,4], Shivam Bhasin[5,7], Anupam Chattopadhyay[6,7], Tee Kiah Chia[7], Matthias J. Kannwischer[8], Franziskus Kiefer[4], Thales B. Paiva[9,10,11], Prasanna Ravi[6,7] and Goutam Tamvada[4]

[1] University of Illinois at Chicago, Chicago, IL 60607-7045, USA
[2] Academia Sinica, Taiwan
[3] Inria, Paris, France
[4] Cryspen, Berlin, Germany
[5] National Integrated Centre for Evaluation, Nanyang Technological University, Singapore
[6] College of Computing and Data Science, Nanyang Technological University, Singapore
[7] Temasek Labs, Nanyang Technological University, Singapore
[8] Quantum Safe Migration Center, Chelpis Quantum Tech, Taipei, Taiwan
[9] University of Sao Paulo, Brazil
[10] Fundep, Brazil
[11] CASNAV, Brazil
authorcontact-kyberslash@box.cr.yp.to

15 January 2025

Abstract. This paper presents KyberSlash1 and KyberSlash2 – two timing vulnerabilities in several implementations (including the official reference code) of the Kyber Post-Quantum Key Encapsulation Mechanism, recently standardized as ML-KEM. We demonstrate the exploitability of both KyberSlash1 and KyberSlash2 on two popular platforms: the Raspberry Pi 2 (Arm Cortex-A7) and the Arm Cortex-M4 microprocessor. Kyber secret keys are reliably recovered within minutes for KyberSlash2 and a few hours for KyberSlash1. We responsibly disclosed these vulnerabilities to maintainers of various libraries and they have swiftly been patched. We present two approaches for detecting and avoiding similar vulnerabilities. First, we patch the dynamic analysis tool Valgrind to allow detection of variable-time instructions operating on secret data, and apply it to more than 1000 implementations of cryptographic primitives in SUPERCOP. We report multiple findings. Second, we propose a more rigid approach to guarantee the absence of variable-time instructions in cryptographic software using formal methods.

Keywords: KyberSlash · PQC · Kyber · ML-KEM · Timing attacks · Division timing

What formal verification can do for you

- Increase trust in
  - security of cryptographic protocols / algorithms
  - security & correctness of cryptographic implementations
- Inform certification (Needs politics!)

See also https://www.nist.gov/news-events/events/2024/07/nist-workshop-formal-methods-within-certification-programs-fmcp-2024

What does formally verified mean?

Can mean many things!

- Functional correctness
- Verified side-channel resistance (usually only timing attacks)
- Verified security proof (computational vs symbolic)

(see https://cryptographycaffe.sandboxaq.com/posts/formal-verification-overview/)

TLS 1.3 (symbolic security proof)

Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication

Cas Cremers, Marko Horvat
Department of Computer Science
University of Oxford, UK

Sam Scott, Thyla van der Merwe
Information Security Group
Royal Holloway, University of London, UK

***Abstract*—After a development process of many months, the TLS 1.3 specification is nearly complete. To prevent past mistakes, this crucial security protocol must be thoroughly scrutinised prior to deployment.**

In this work we model and analyse revision 10 of the TLS 1.3 specification using the Tamarin prover, a tool for the automated analysis of security protocols. We specify and analyse the interaction of various handshake modes for an unbounded number of concurrent TLS sessions. We show that revision 10 meets the goals of authenticated key exchange in both the unilateral and mutual authentication cases.

We extend our model to incorporate the desired delayed client authentication mechanism, a feature that is likely to be included in the next revision of the specification, and uncover a potential attack in which an adversary is able to successfully impersonate a client during a PSK-resumption handshake. This observation was reported to, and confirmed by, the IETF TLS Working Group.

Our work not only provides the first supporting evidence for the security of several complex protocol mode interactions in TLS 1.3, but also shows the strict necessity of recent suggestions to include more information in the protocol's signature contents.

on both the manual and automated fronts and resulting in the discovery of many weaknesses.

The various flaws identified in TLS 1.2 [17] and below, be they implementation- or specification-based, have prompted the TLS Working Group to adopt an 'analysis-before-deployment' design paradigm in drafting the next version of the protocol, TLS 1.3 [48]. Most notably, the cryptographic core of the new TLS handshake protocol is largely influenced by the OPTLS protocol of Krawczyk and Wee [35], a protocol that has been expressly designed to offer zero Round-Trip Time (0-RTT) exchanges and ensure perfect forward secrecy. Its simple structure lends itself to analysis via manual and automated means, a benefit that was deemed desirable for TLS 1.3. Although the logic of the protocol has been simplified, the addition of 0-RTT functionality as well as the new resumption and client authentication mechanisms has introduced new complexity.

The overall complexity of TLS 1.3 implies that to perform a truly complete cryptographic analysis (either manual or tool-supported) of the entire protocol would be a substantial undertaking, and unlikely to be completed in time for the release of TLS 1.3.

However, given the critical importance of TLS, it is paramount that the TLS 1.3 protocol design is critically

Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate

Karthikeyan Bhargavan, Bruno Blanchet, Nadim Kobeissi
INRIA
{karthik.bhargavan,bruno.blanchet,nadim.kobeissi}@inria.fr

***Abstract*—TLS 1.3 is the next version of the Transport Layer Security (TLS) protocol. Its clean-slate design is a reaction both to the increasing demand for low-latency HTTPS connections and to a series of recent high-profile attacks on TLS. The hope is that a fresh protocol with modern cryptography will prevent legacy problems; the danger is that it will expose new kinds of attacks, or reintroduce old flaws that were fixed in previous versions of TLS. After 18 drafts, the protocol is nearing completion, and the working group has appealed to researchers to analyze the protocol before publication. This paper responds by presenting a comprehensive analysis of the TLS 1.3 Draft-18 protocol.**

We seek to answer three questions that have not been fully addressed in previous work on TLS 1.3: (1) Does TLS 1.3 prevent well-known attacks on TLS 1.2, such as Logjam or the Triple Handshake, even if it is run in parallel with TLS 1.2? (2) Can we mechanically verify the computational security of TLS 1.3 under standard (strong) assumptions on its cryptographic primitives? (3) How can we extend the guarantees of the TLS 1.3 protocol to the details of its implementations?

To answer these questions, we propose a methodology for developing verified symbolic and computational models of TLS 1.3 hand-in-hand with a high-assurance reference implementation of the protocol. We present symbolic ProVerif models for various intermediate versions of TLS 1.3 and evaluate them against a rich class of attacks to reconstruct both known and previously unpublished vulnerabilities that influenced the current design of the protocol. We present a computational CryptoVerif model for TLS 1.3 Draft-18

depends crucially on clients and servers negotiating the most secure variant that they have in common. Securely composing and implementing the many different versions and features of TLS has proved to be surprisingly hard, leading to the continued discovery of high-profile vulnerabilities in the protocol.

A history of vulnerabilities. We identify four kinds of attacks that TLS has traditionally suffered from. *Downgrade* attacks enable a network adversary to fool a TLS client and server into using a weaker variant of the protocol than they would normally use with each other. In particular, version downgrade attacks were first demonstrated from SSL 3 to SSL 2 [72] and continue to be exploited in recent attacks like POODLE [60] and DROWN [7]. *Cryptographic* vulnerabilities rely on weaknesses in the protocol constructions used by TLS. Recent attacks have exploited key biases in RC4 [3], [71], padding oracles in MAC-then-Encrypt [4], [60], padding oracles in RSA PKCS#1 v1.5 [7], weak Diffie-Hellman groups [1], and weak hash functions [23]. *Protocol composition* flaws appear when multiple modes of the protocol interact in unexpected ways if enabled in parallel. For example, the renegotiation attack [65] exploits the sequential composition of two TLS handshakes, the Triple Handshake attack [15] composes three handshakes, and cross-protocol attacks [58], [72] use one kind of TLS handshake to attack another. *Implementation bugs* contribute

Verifying PQC (computational security proofs)

- Barbosa, Barthe, Fan, Grégoire, Hung, Katz, Strub, Wu, and Zhou. EasyPQC: Verifying Post-Quantum Cryptography. ACM CCS 2021
- Hülsing, Meijers, and Strub. Formal Verification of **Saber**'s Public-Key Encryption Scheme in EasyCrypt. CRYPTO 2022
- Barbosa, Barthe, Doczkal, Don, Fehr, Grégoire, Huang, Hülsing, Lee, and Wu. Fixing and Mechanizing the Security Proof of Fiat-Shamir with Aborts and **Dilithium**. CRYPTO 2023
- Barbosa, Dupressoir, Grégoire, Hülsing, Meijers, and Strub. Machine-Checked Security for **XMSS** as in RFC 8391 and SPHINCS+. CRYPTO 2023
- Barbosa, Dupressoir, Hülsing, Meijers, Strub. A Tight Security Proof for **SPHINCS$^+$**, Formally Verified. ASIACRYPT 2024
- Almeida, Arranz Olmos, Barbosa, Barthe, Dupressoir, Grégoire, Laporte, Léchenet, Low, Oliveira, Pacheco, Quaresma, Schwabe, Strub. Formally Verifying Kyber Episode V: Machine-Checked IND-CCA Security and Correctness of **ML-KEM** in EasyCrypt. CRYPTO 2024

The Cryptography Caffè ☕

Real-World Verification of Software for Cryptographic Applications

by Tiago Oliveira, Andreas Hülsing, Gaëtan Wattiau, and Steven Yue from SandboxAQ; Karthikeyan Bhargavan, Maxime Buyse, and Lucas Franceschino from Cryspen. Posted on Mar 25, 2025

"Draw an abstract image of formal methods being applied to real-world, security-critical code." by DALL-E.

In this blog post, we describe how we at SandboxAQ, together with Cryspen, formally verified key components of Sandwich—an open-source, unified API that simplifies the use of cryptographic libraries for developers, enabling crypto-agility.

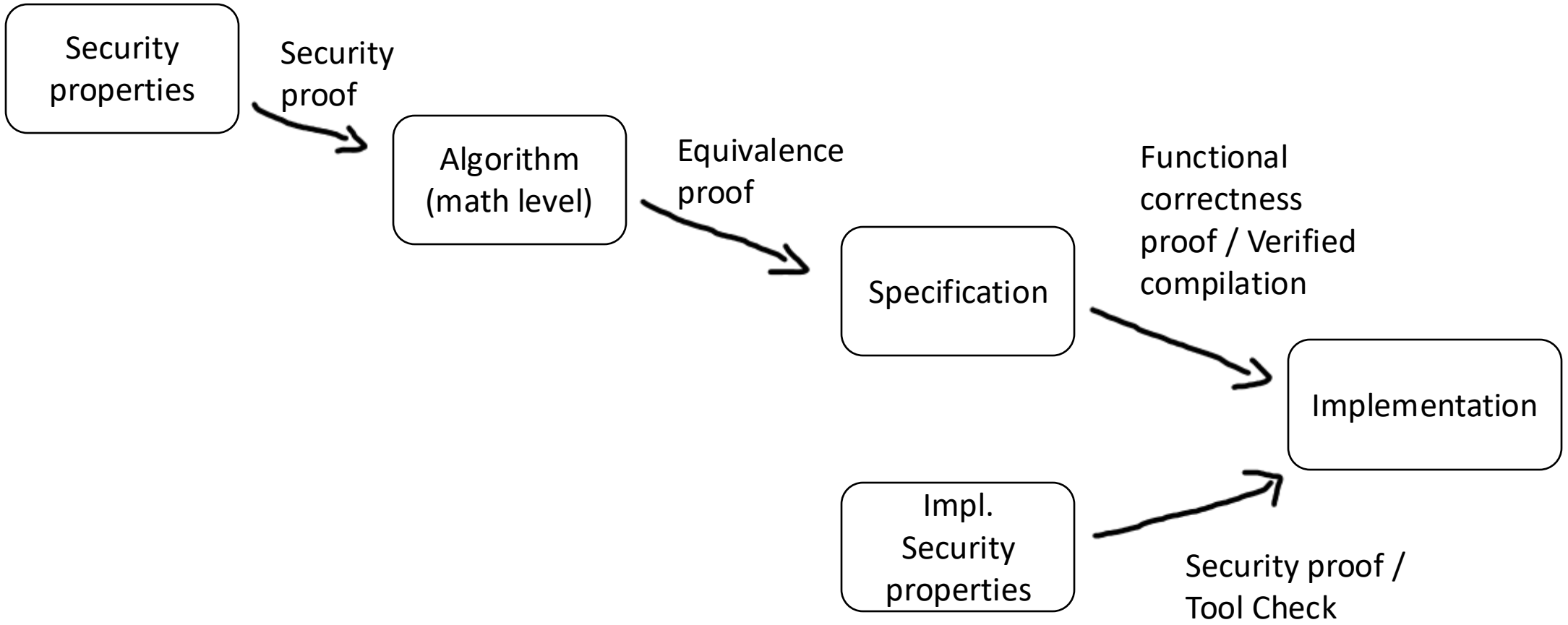
Formal methods have been used successfully to produce high-assurance implementations of cryptographic algorithms that have since been integrated into popular libraries like BoringSSL, NSS, and AWS LC. However, APIs of cryptographic libraries are often complex to use and can be error-prone. Sandwich tackles this problem.

https://cryptographycaffe.sandboxaq.com/posts/real-world-verification-of-software-for-cryptographic-application

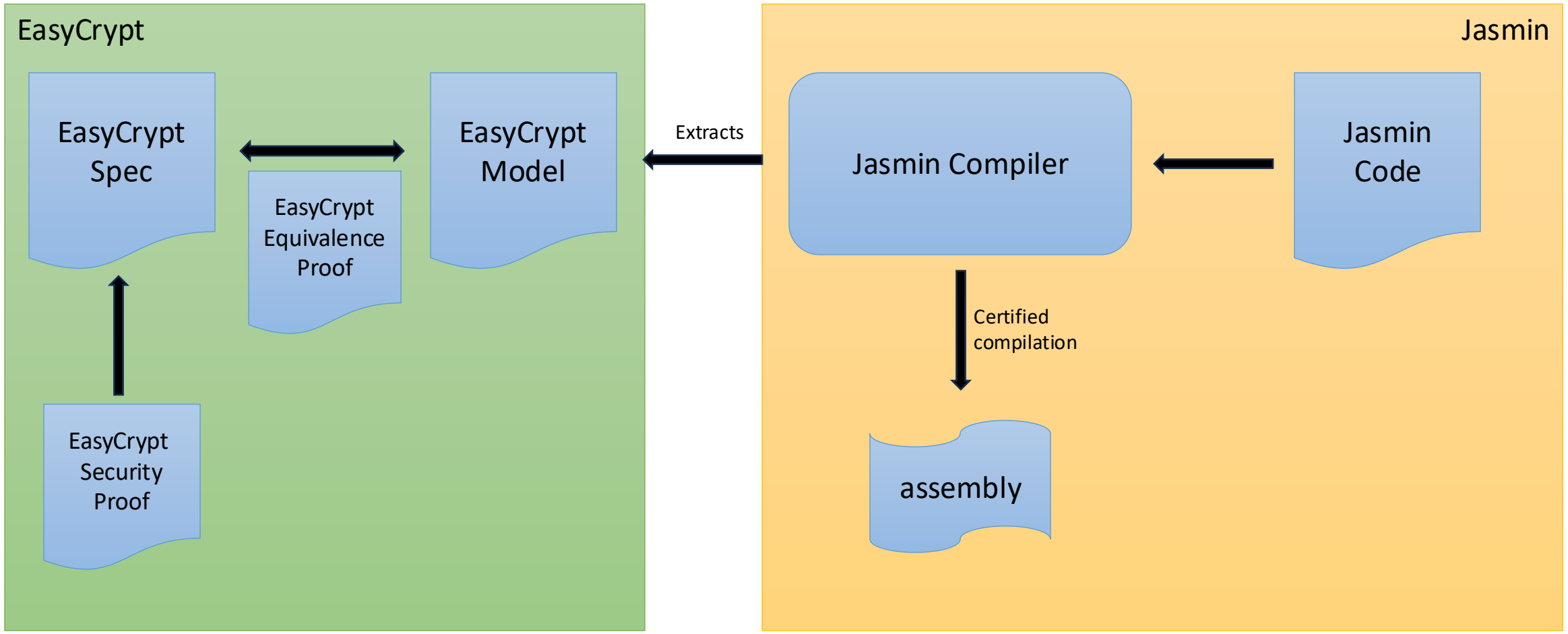
Act II: Fun for academics

(the how & who)

Components related to crypto code (primitive level)



Formosa view



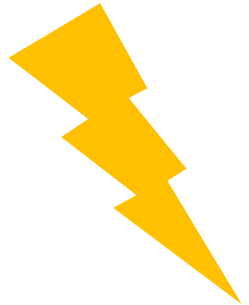
What Formosa cannot do, yet

- Automated proofs of security or equivalence
  - In progress, improving
- Verify your existing code (except if it is Jasmin code)
  - Possible way out: Executable EasyCrypt
- Analyze complex protocols
  - Possible in theory, complexity hard to manage in practice
- Analyze arbitrary quantum algorithms
  - EasyPQC: Support for basic QROM arguments

Viewpoints

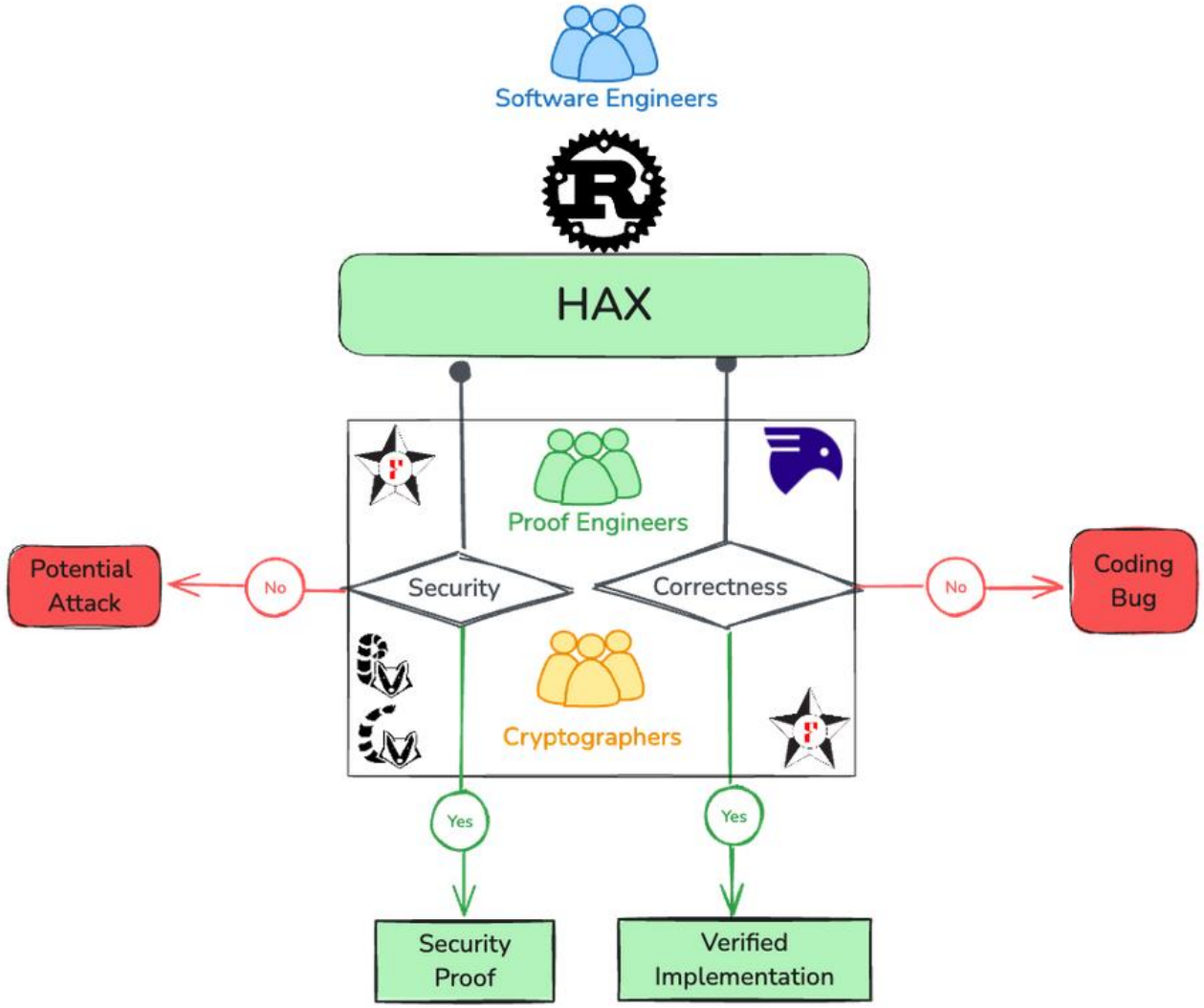
Me@Formosa

- We provide you with a tool and a programming language that allows you to check your software and proofs



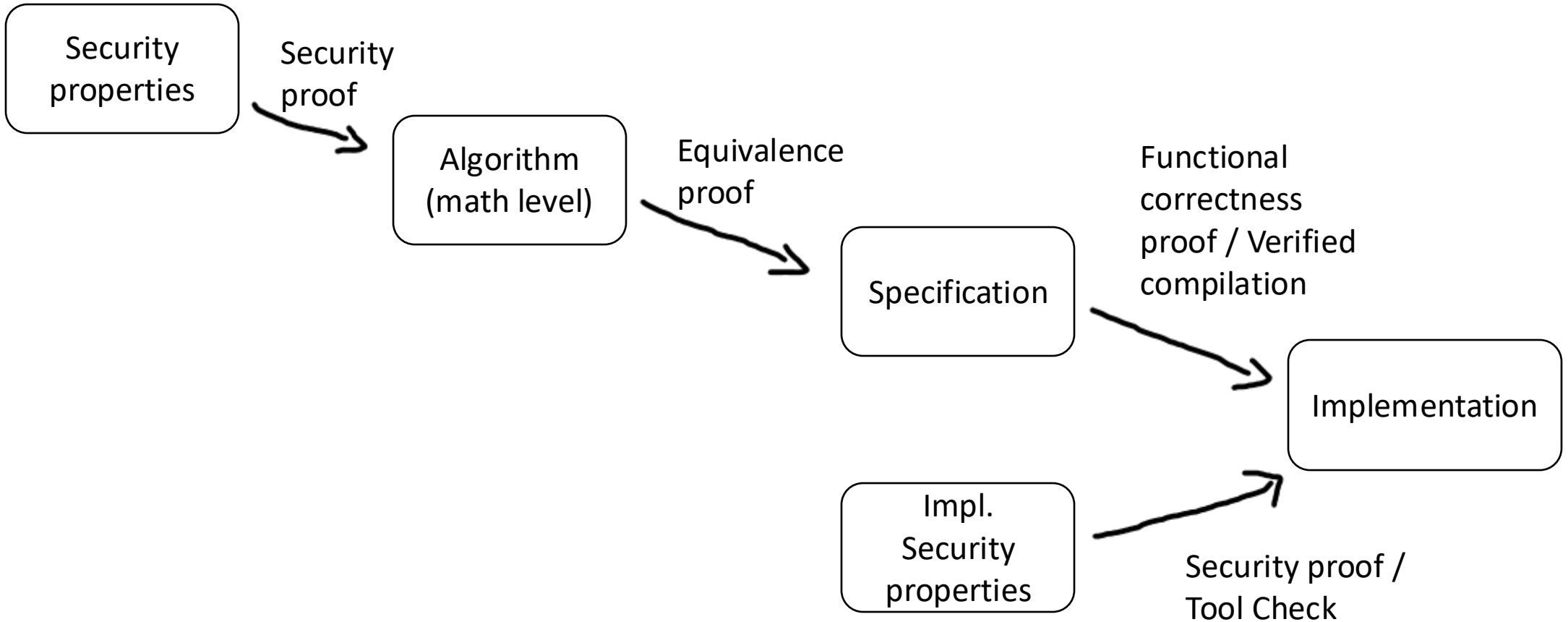
Me@SandboxAQ

- We have this code that we would like to verify



https://hax.cryspen.com/

Components related to crypto code (primitive level)



HAX

- Can verify RUST code (a subset)
- Can verify functional correctness (with respect to a F* spec)
- Can verify security claims in F* or other proof-assistants (Rocq, SSProve, ProVerif)
- More automation for simple tasks
- Maintenance of bridges tricky

One way out: formally verified libraries

Libjade

Libjade

Libjade is a formally verified cryptographic library written in the jasmin programming language with computer-verified proofs in EasyCrypt. Libjade is part of the Formosa-Crypto project.

The primary focus is on offering high-assurance implementations of post-quantum crypto (PQC) primitives to support the migration to the next generation of asymmetric cryptography. The library additionally contains implementations of various symmetric primitives and—to enable hybrid deployment of PQC—also widely used elliptic-curve-based schemes.

Information for users

This section contains information for anybody who would like to integrate code from Libjade into a higher-level cryptographic library or application. If you would like to compile Libjade yourself, run tests, or reproduce proofs, please see information for developers below.

Supported platforms

The jasmin compiler produces assembly, so all code in Libjade is platform specific. In the latest release of Libjade, the only supported architecture is AMD64 (aka x86_64 or x64). All code in the latest release is in AT&T assembly format and is following the System V AMD64 ABI, which is used by, e.g., Linux, FreeBSD, and (Intel-based) macOS.

Primitives available in Libjade

The latest release of Libjade includes implementations of the following primitives (asymmetric post-quantum primitives in boldface):

- Hash functions: SHA-256, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512
- Extendable output functions (XOFs): SHAKE-128, SHAKE-256
- One-time authenticators: Poly1305
- Stream ciphers: ChaCha12, ChaCha20, Salsa20, XSalsa20
- Authenticated encryption ("secretbox"): XSalsa20Poly1305
- Scalar multiplication: Curve25519
- Key-encapsulation: **Kyber-512**, **Kyber-768**
- Signatures: **Falcon512** (verification only)

https://github.com/formosa-crypto/libjade

Libcrux

https://github.com/cryspen/libcrux

libcrux - the formally verified crypto library

libcrux is a formally verified cryptographic library in Rust.

Minimum Supported Rust Version (MSRV)

The default feature set has a MSRV of `1.78.0`. `no_std` environments are supported starting from Rust version `1.81.0`.

Randomness

libcrux provides a DRBG implementation that can be used standalone (`drbg::Drbg`) or through the `Rng` traits.

`no_std` support

`libcrux` and the individual primitive crates it depends on support `no_std` environments given a global allocator for the target platform.

Verification status

As a quick indicator of overall verification status, subcrates in this workspace include the following badges:

- PRE VERIFICATION to indicate that most (or all) of the code that is contained in default features of that crate is not (yet) verified.
- VERIFIED (HACL-RS) to indicate that algorithms in a crate have been verified and extracted to Rust as part of the HACL project. Top-level APIs in these crates accessing the code from HACL may not be verified.
- VERIFIED to indicate that most (or all) of the code that is contained in the default feature set is verified.

In every case, please refer to the more detailed notes on verification in each sub-crate to learn more about what has (or has not) been verified in the particular case.

Symbolic vs Computational

Computational model (EasyCrypt, Coq, SSProve, CryptoVerif, ...)

- Computational adversaries
  - Only limitation: Interaction with honest parties
  - "Black-listing" adversary actions
- Proof -> Strong statement
- Limited automation
- Proof fails -> ??? Good luck

Symbolic model (Tamarin, ProVerif, ...)

- Dolev-Yao adversary
  - Idealized model
  - Adversary limited to defined actions
  - "White-listing" adversary actions
- Proof -> ???
  - "Proves absence of in model attacks"
- Mostly automated
- No proof? -> Attack

Limitations

Results are great but

- Full workflow for Kyber took more than 3 years of many, many people! (Still not fully published!)
- Tools are "Expert Tools"
- New proofs often need help of tool developer
- Little automation or weaker results
- Little integration between tools
- **Statements must be verified!**

Summary

- We have the tools, we can achieve great results (see Kyber)
- Verifying proofs is still research
- Usability still needs improvement
- There are many different tools for different use-cases

- We are working on a fully verified PQC library!
- Check out the Formosa project (https://formosa-crypto.org/)

- Effort to formally verify crypto
- Goal: verified PQC ready for deployment
- Three main projects:
  - EasyCrypt proof assistant
  - Jasmin programming language
  - Libjade (PQ-)crypto library
- Core community of ≈ 30–40 people
- Discussion forum with >180 people

University of BRISTOL

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY

institute IMDEA software

U.PORTO

INESCTEC

Radboud University

Inria INVENTEURS DU MONDE NUMÉRIQUE

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY