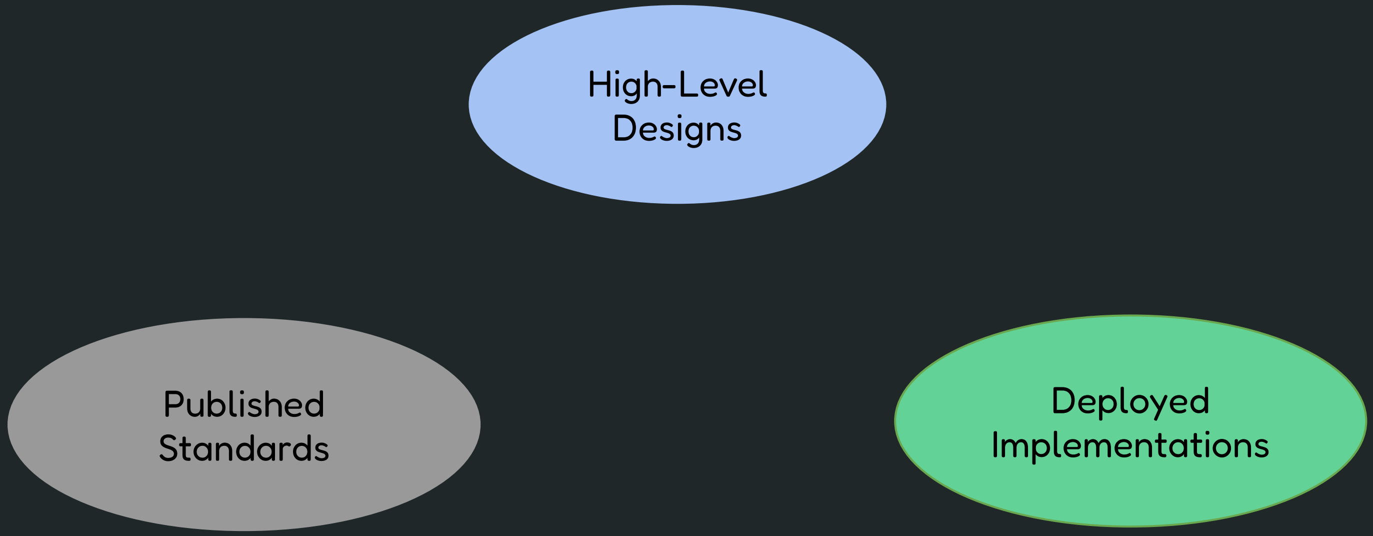
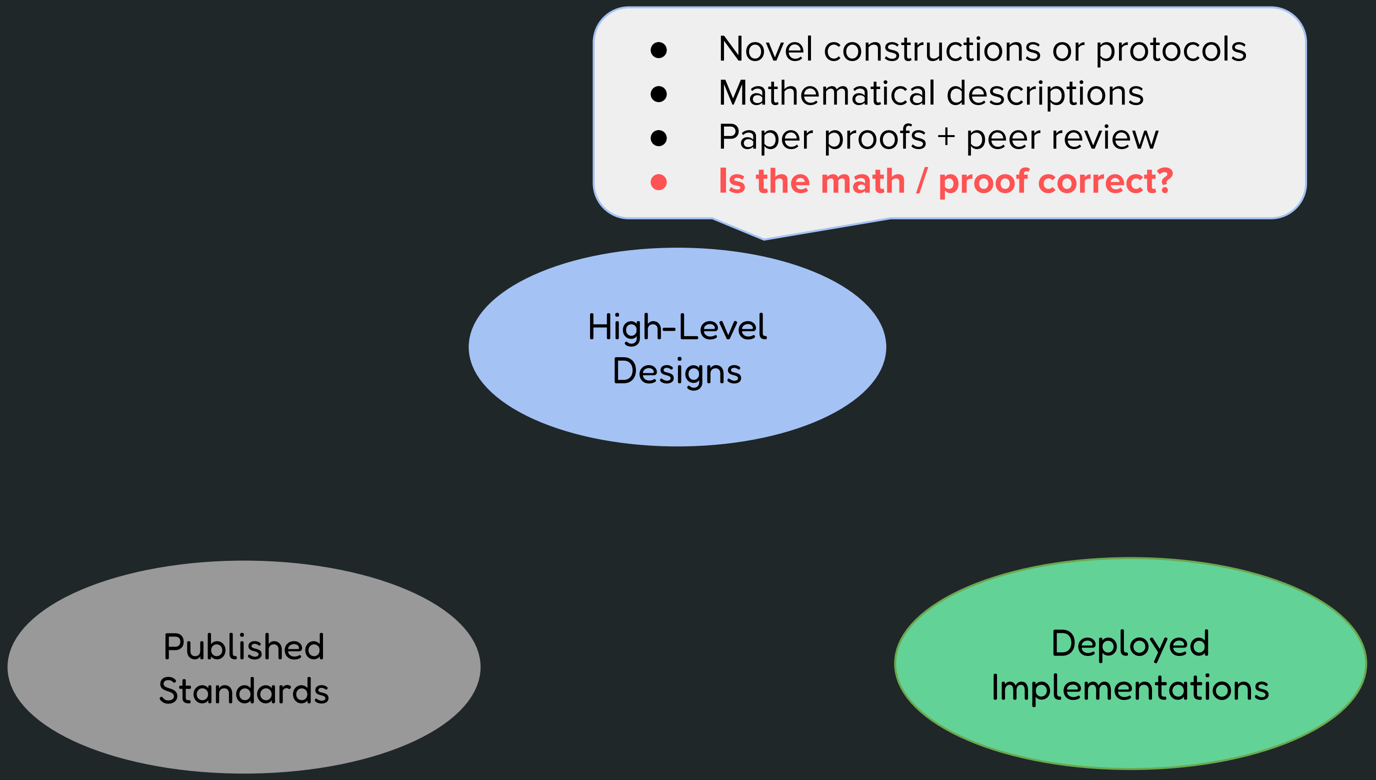
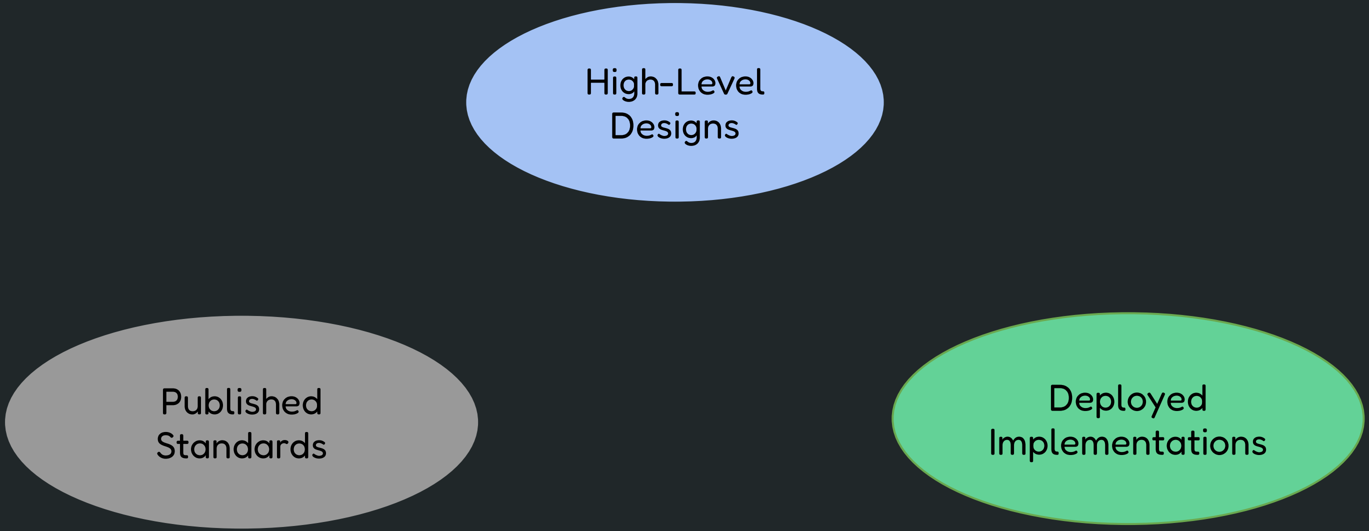
Formal Methods for Cryptography: protocols, standards, implementations

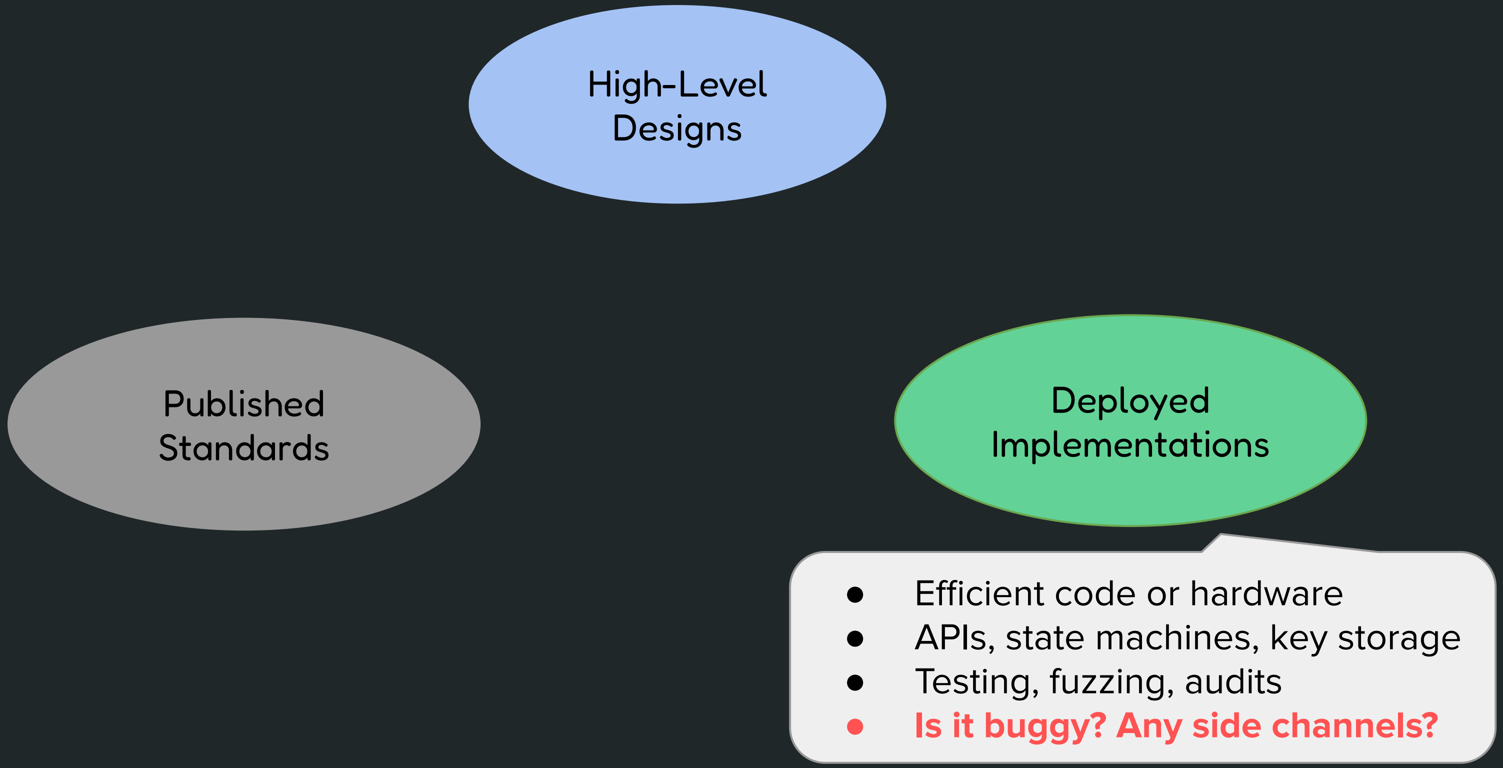
Karthikeyan Bhargavan
+ work from many many other colleagues and co-authors

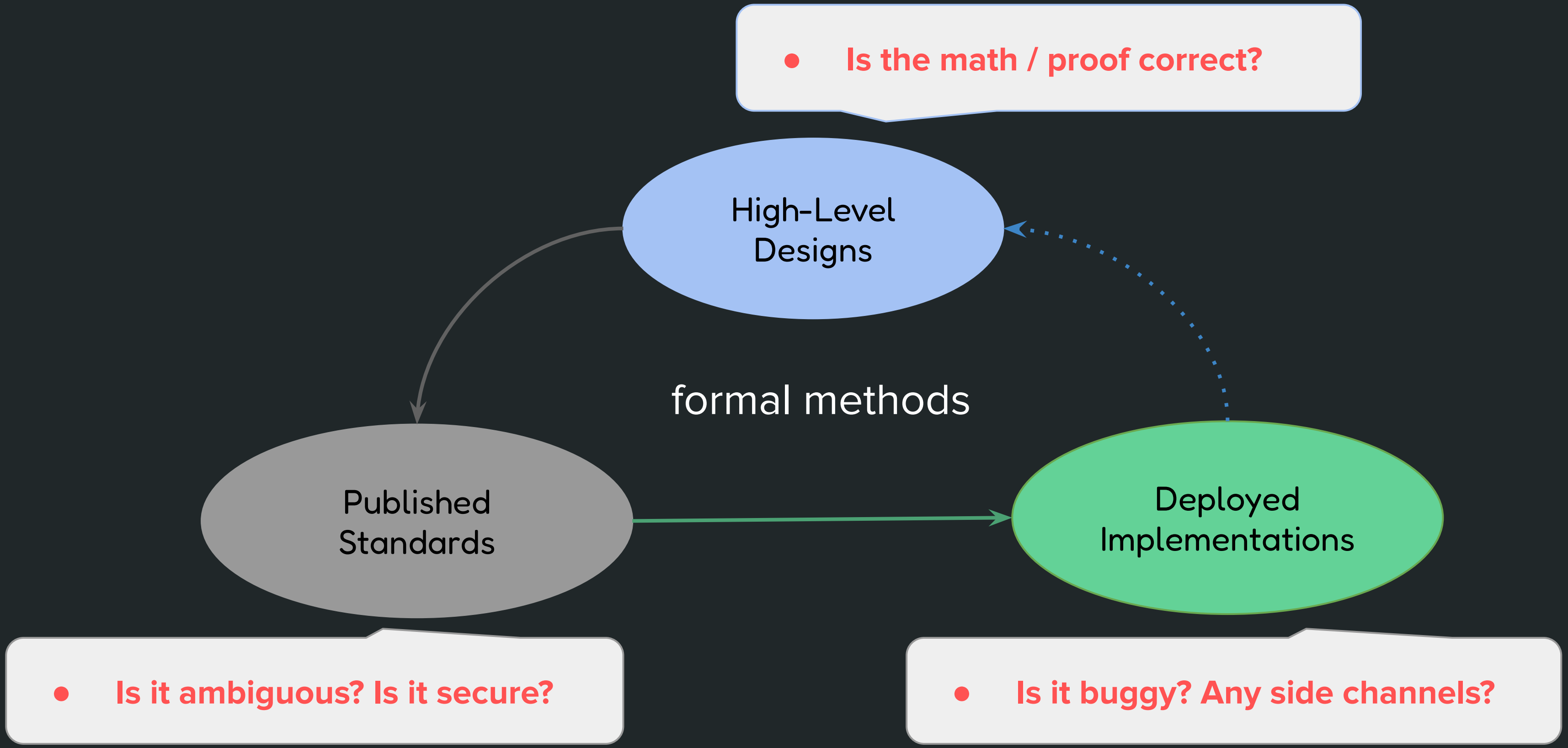

High-Level Designs
Published Standards
Deployed Implementations


Novel constructions or protocols
Mathematical descriptions
Paper proofs + peer review
Is the math / proof correct?
High-Level Designs
Published Standards
Deployed Implementations



- Detailed technical specification
- Algorithms, encodings, validation
- Community review, implementation
- **Is it ambiguous? Is it secure?**


High-Level Designs
Published Standards
Deployed Implementations
Efficient code or hardware
APIs, state machines, key storage
Testing, fuzzing, audits
Is it buggy? Any side channels?


Is the math / proof correct?
High-Level Designs
formal methods
Published Standards
Deployed Implementations
Is it ambiguous? Is it secure?
Is it buggy? Any side channels?

Formal Methods for Crypto

- Computer-Aided Cryptography, a.k.a. High Assurance Cryptography

  "Applying formal, **machine-checkable** approaches to the design, analysis, and implementation of cryptography."

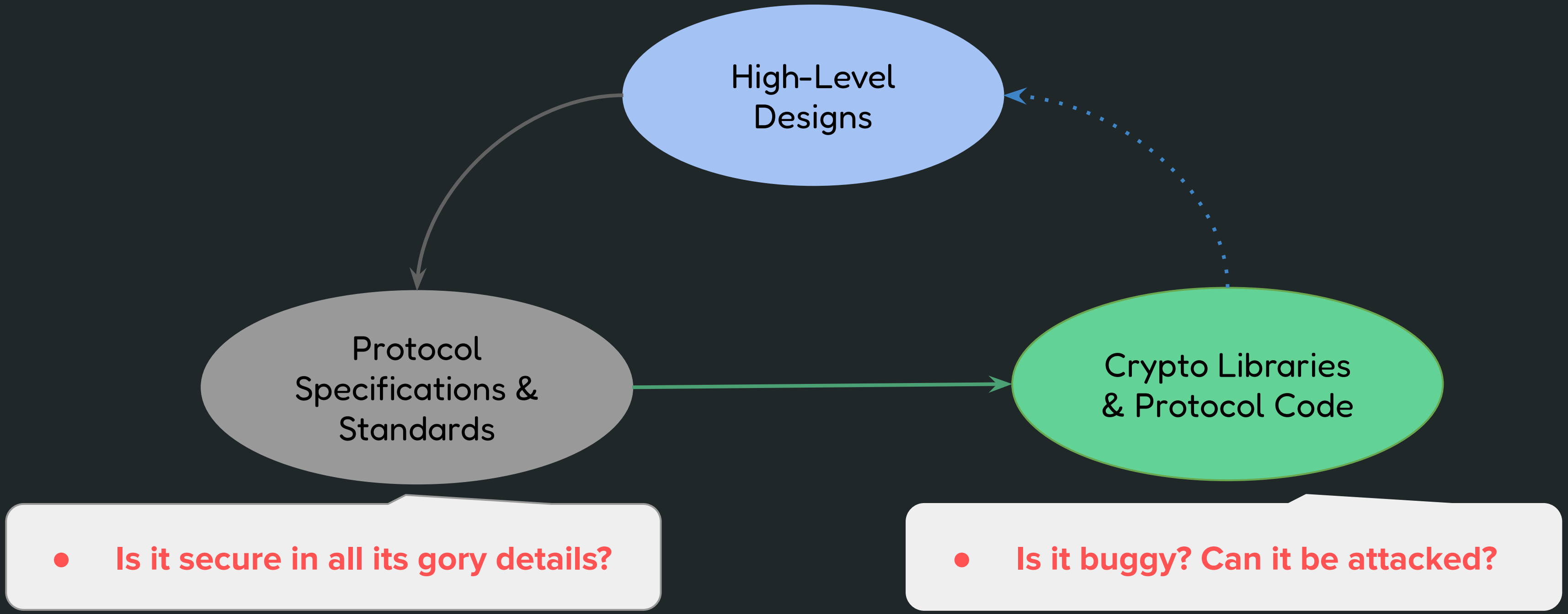
  SoK: Computer-Aided Cryptography, IEEE S&P 2021
  Barbosa, Barthe, Bhargavan, Blanchet, Cremers, Liao, Parno

- Analyze cryptographic **designs** early to find attacks or uncover assumptions
- Comprehensively analyze **specifications** and standards before publication
- Formally verify efficient **implementations** to prevent bugs and side-channels
- .... and **repeat** these steps over and over again as these artifacts evolve

Formal Methods for Crypto

- Related talks
  - EuroCrypt 2017: *Advances in computer-aided cryptography*, Gilles Barthe
  - CHES 2023: *High-assurance crypto in practice,* Peter Schwabe

- Recent papers
  - CRYPTO 2024: *Formally Verifying Kyber Episode V: Machine-checked IND-CCA security and correctness of ML-KEM in EasyCrypt* [Almeida, Olmos, Barbosa, Barthe, Dupressoir, Gregoire, Laporte, Léchenet, Low, Oliveira, Pacheco, Quaresma, Schwabe, Strub]
  - CRYPTO 2023: *Machine-Checked Security for XMSS as in RFC 8391 and SPHINCS+* [Barbosa, Dupressoir, Grégoire, Hülsing, Meijers, Strub]
  - EuroCrypt 2021: *Analysing the HPKE Standard* [Alwen, Blanchet, Hauck, Kiltz, Lipp, Riepel]
  - \+ many papers at IEEE CSF, IEEE S&P, ACM CCS, Usenix Security etc.


High-Level Designs
Protocol Specifications & Standards
Crypto Libraries & Protocol Code
Is it secure in all its gory details?
Is it buggy? Can it be attacked?

- There is a wide range of formal methods for crypto
- Using them can **speed deployment** and **clarify security** of crypto in real-world protocols

- A small effort can sometimes make a big impact
- *You* can help make the tool effective for *your work*

TLS 1.3: A (Big) Success Story

The TLS 1.3 Collaboration

- IETF RFC 8446
- 28 drafts, April 2014 - August 2018
- A complete revamp of TLS
  - Lower latency, remove weak crypto,
  - Strengthen integrity, improve privacy

Internet Engineering Task Force (IETF) E. Rescorla
Request for Comments: 8446 Mozilla
Obsoletes: 5077, 5246, 6961 August 2018
Updates: 5705, 6066
Category: Standards Track
ISSN: 2070-1721

The Transport Layer Security (TLS) Protocol Version 1.3

Abstract

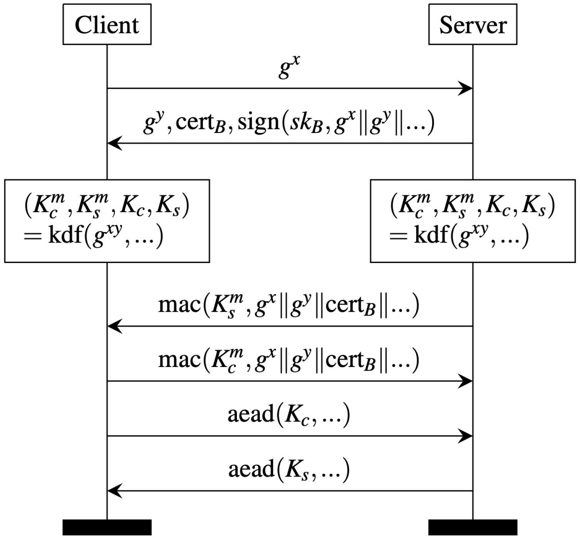
This document specifies version 1.3 of the Transport Layer Security (TLS) protocol. TLS allows client/server applications to communicate over the Internet in a way that is designed to prevent eavesdropping, tampering, and message forgery.

- Deep long-term interactions between working group and cryptographic researchers
- Many pen-and-paper proofs
- Some machine-checked proofs
  - Tamarin, ProVerif, CryptoVerif, F*

TLS 1.3: Path to standardization

2014

2018



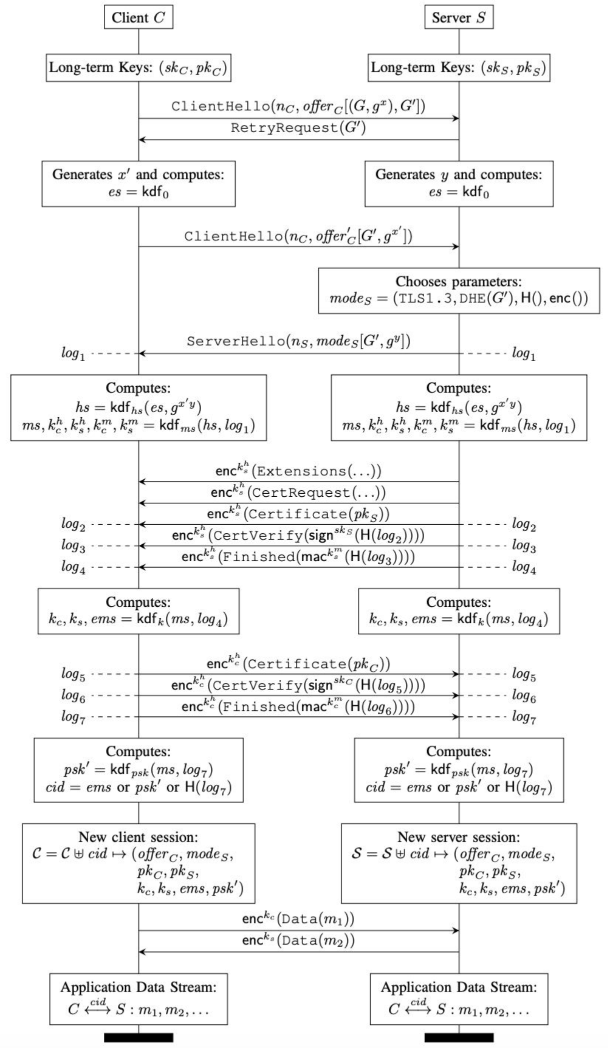
SIGMA variant
with unilateral auth

TLS 1.3: Path to standardization

2014

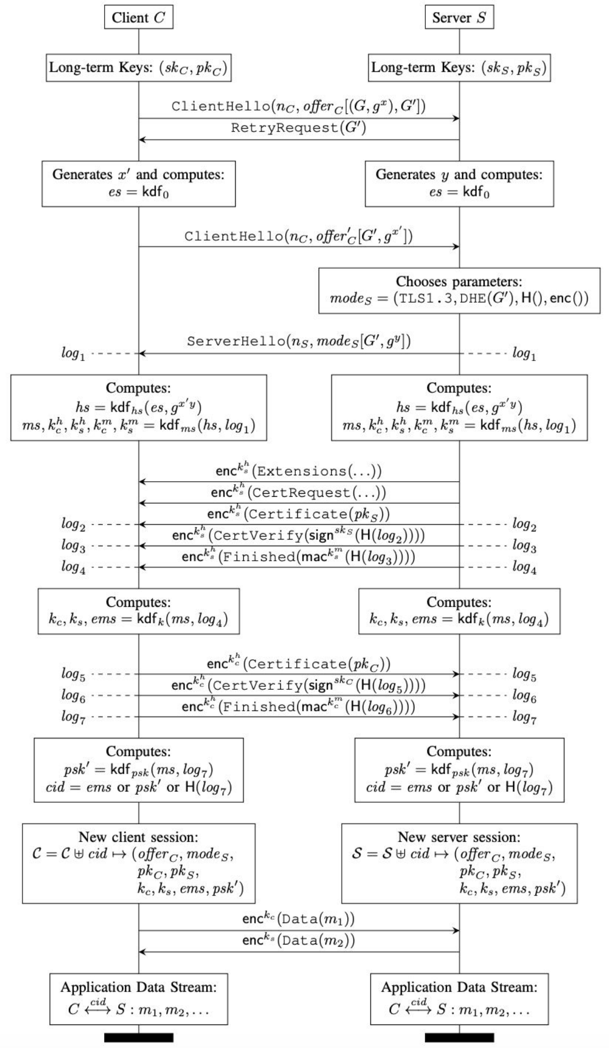
Draft 5	[Dowling, Fischlin, Günther, Stebila, 2015]
Draft 7	[Jager, Schwenk, Somorowsky, 2015]
Draft 9	[Krawczyk, Wee, 2016]
Draft 10	[Li, Xu, Zhang, Feng, Hu, 2016] [Fischlin, Günther, Schmidt, Warinschi, 2016] [Cremers, Horvat, Scott, van der Merwe, 2016]
Draft 12	[Bhargavan, Brzuska, Fournet, Green, Kohlweiss, Beguelin, 2016]
Draft 14	[Fischlin, Günther, 2017]
Draft 18	[Bhargavan, Delignat-Lavaud, Fournet, Kohlweiss, Pan, Protzenko, Rastogi, Swamy, Zanella-Béguelin, Zinzindohoué, 2017] [Bhargavan, Blanchet, Kobeissi, 2017]
Draft 21	[Cremers, Horvat, Hoyland, Scott, van der Merwe, 2017]

2018



TLS 1.3: Impact on new protocols

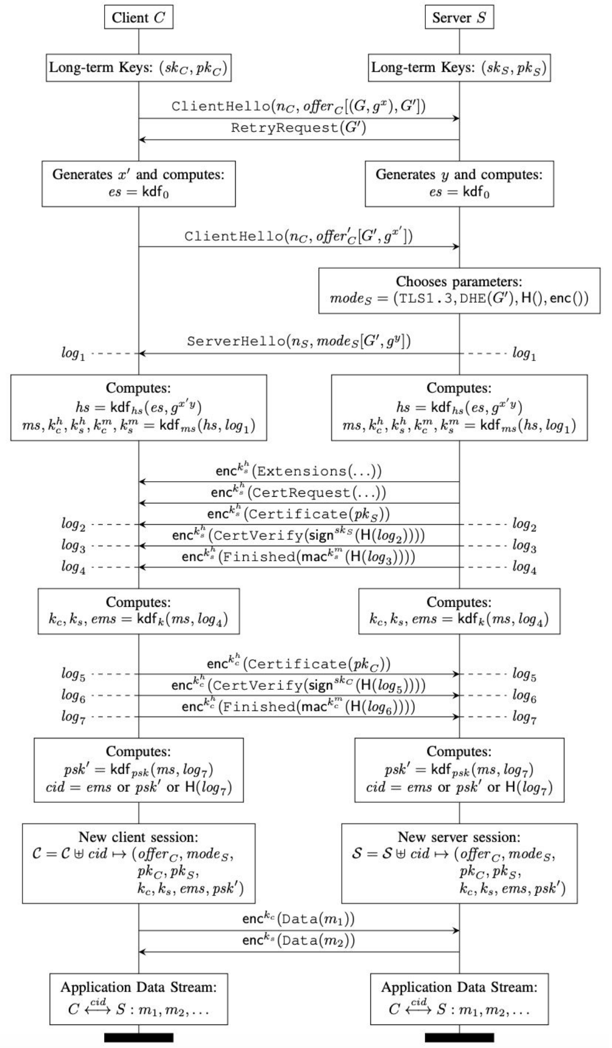
- Proofs now often required for new designs
  - Machine-checked proofs are a plus
- IETF Working Groups
  - **LAKE**: Key exchange protocol for IoT
  - **TLS**: Encrypted Client Hello, KEM-TLS
  - **MLS:** Secure group messaging
- Industrial Protocols
  - PQ3 (iMessage), PQXDH (Signal)



Lessons from TLS 1.3

- Designer ↔ Researcher exchanges are crucial
  - Design constraints ↔ Proof Limitations
- Low-level details matter for security
  - Key schedule, signature formats, ...
- Complexity arises from protocol composition
  - Multiple key exchange modes, auth modes, ...
- Prepare to answer the next question quickly
  - Months for adapting proofs is a rare luxury

For all of these, formal methods give you an edge

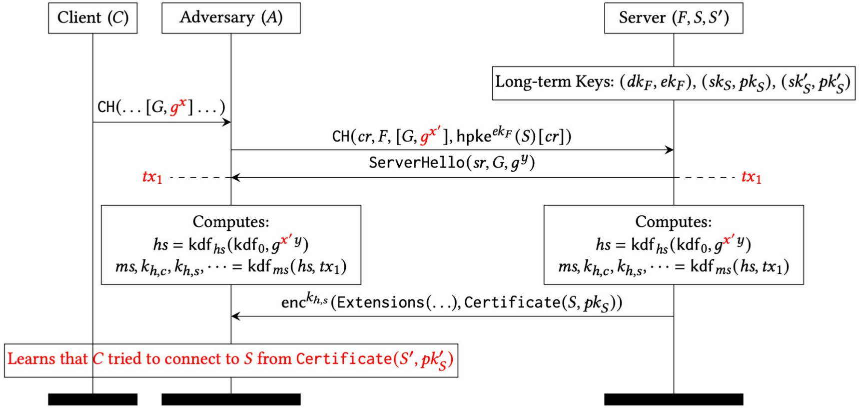


TLS 1.3+ECH: Improving privacy for TLS 1.3

- TLS 1.3 encrypts most handshake , message, but sends server name in the clear in the first message
- ECH privacy extension aims to fix this
  - many early proposals were broken
  - active network attacker could distinguish target server
- Can we prove privacy for TLS+ECH?
  - Does ECH preserve TLS 1.3 security?

Workgroup:	tls			
Internet-Draft:	draft-ietf-tls-esni-20			
Published:	5 August 2024			
Intended Status:	Standards Track			
Expires:	6 February 2025			
Authors:	E. Rescorla	K. Oku	N. Sullivan	C. A. Wood
	Independent	*Fastly*	*Cryptography Consulting LLC*	*Cloudflare*

TLS Encrypted Client Hello



TLS 1.3+ECH: A Machine-Checked Analysis

- Formally evaluate TLS 1.3+ECH
  a. Adapt ProVerif model of TLS 1.3 [Bhargavan, Blanchet, Kobeissi, 2017]
  b. Add ECH mechanisms, new threat model, new security goals
  c. Find attacks, fix, repeat
  d. Reprove ECH preserves TLS security (in the symbolic model)
  e. Prove TLS privacy theorem (under many, many, subtle conditions.)

- Hard/impossible to extend or reuse pen-and-paper proofs in this way

Workgroup:	tls
Internet-Draft:	draft-ietf-tls-esni-20
Published:	5 August 2024
Intended Status:	Standards Track
Expires:	6 February 2025
Authors:	E. Rescorla *Independent*; K. Oku *Fastly*; N. Sullivan *Cryptography Consulting LLC*; C. A. Wood *Cloudflare*

TLS Encrypted Client Hello

A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello

Karthikeyan Bhargavan
Inria Paris
Paris, France
karthikeyan.bhargavan@inria.fr

Vincent Cheval
Inria Paris
Paris, France
vincent.cheval@inria.fr

Christopher Wood
Cloudflare
San Francisco, United States
chriswood@cloudflare.com

PQXDH: Analyzing (Small) New Protocols

The (Classical) Signal Protocol

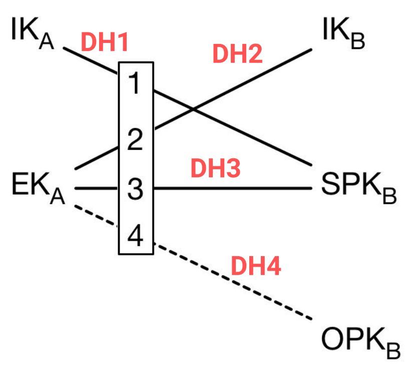
Two parts:

- **X3DH** handshake
- **Double Ratchet** for continuous key agreement

Important security guarantees:

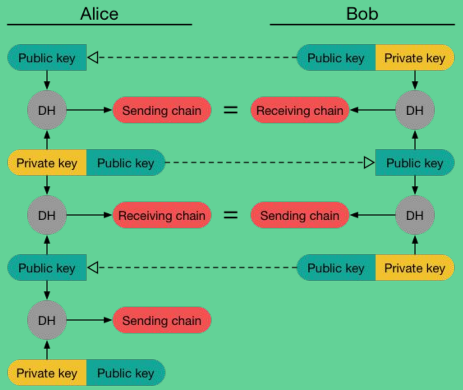
- Mutual authentication
- Post-compromise security
- Forward secrecy
- Deniability

X3DH

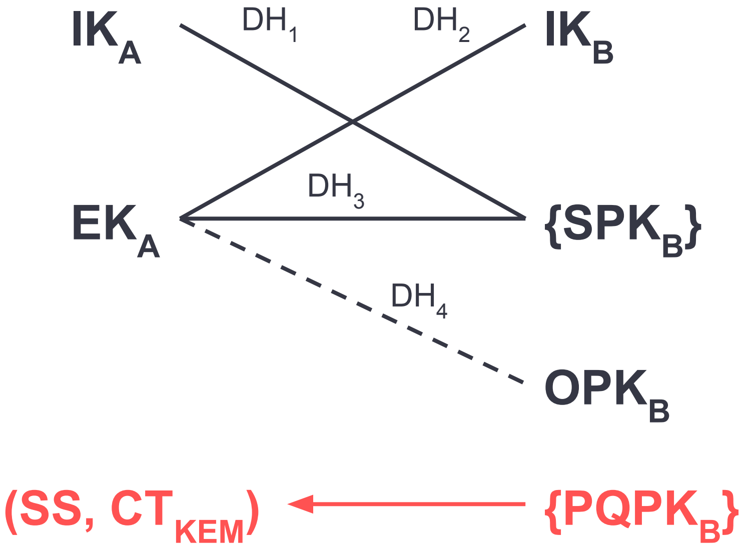


SK = KDF(**DH1** || **DH2** || **DH3** || **DH4**)

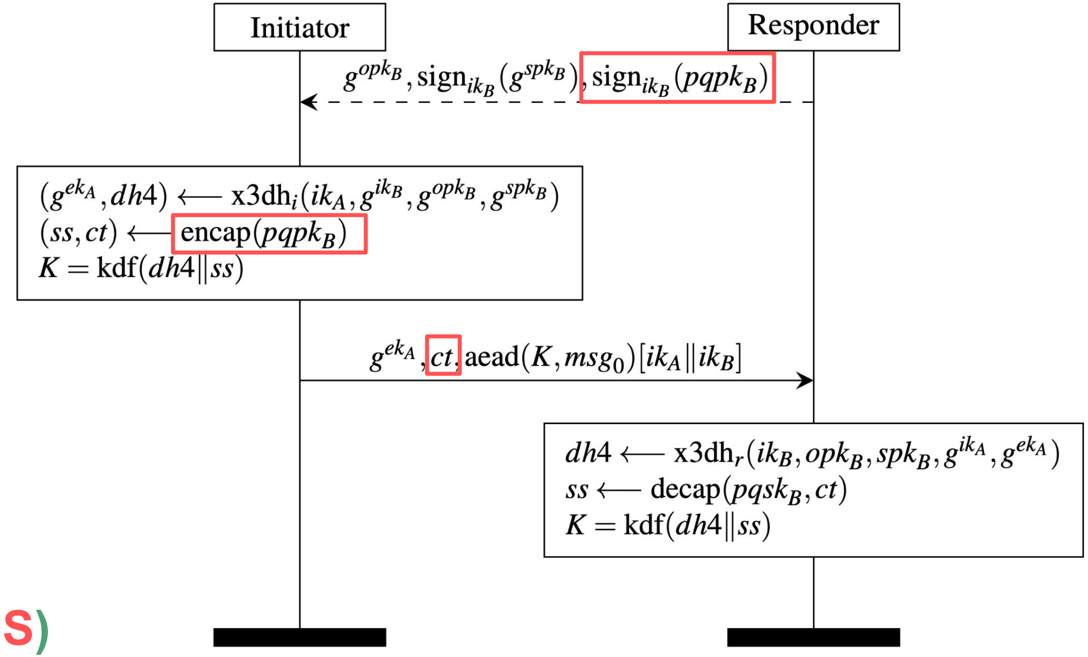
Double Ratchet



PQXDH Design: Add a PQ-KEM to X3DH



$SK = KDF(DH_1 \| DH_2 \| DH_3 \| DH_4 \| SS)$

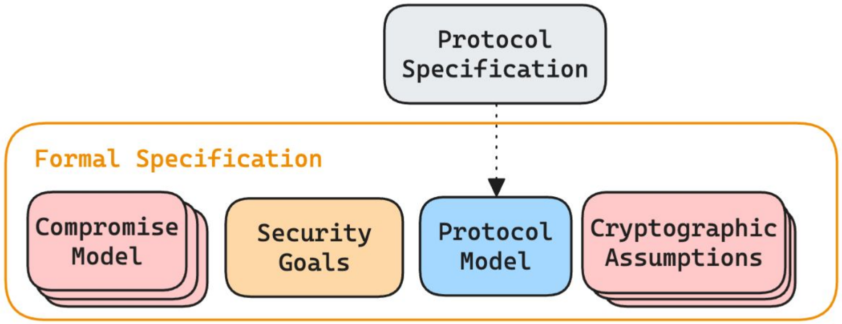


Analyzing PQXDH

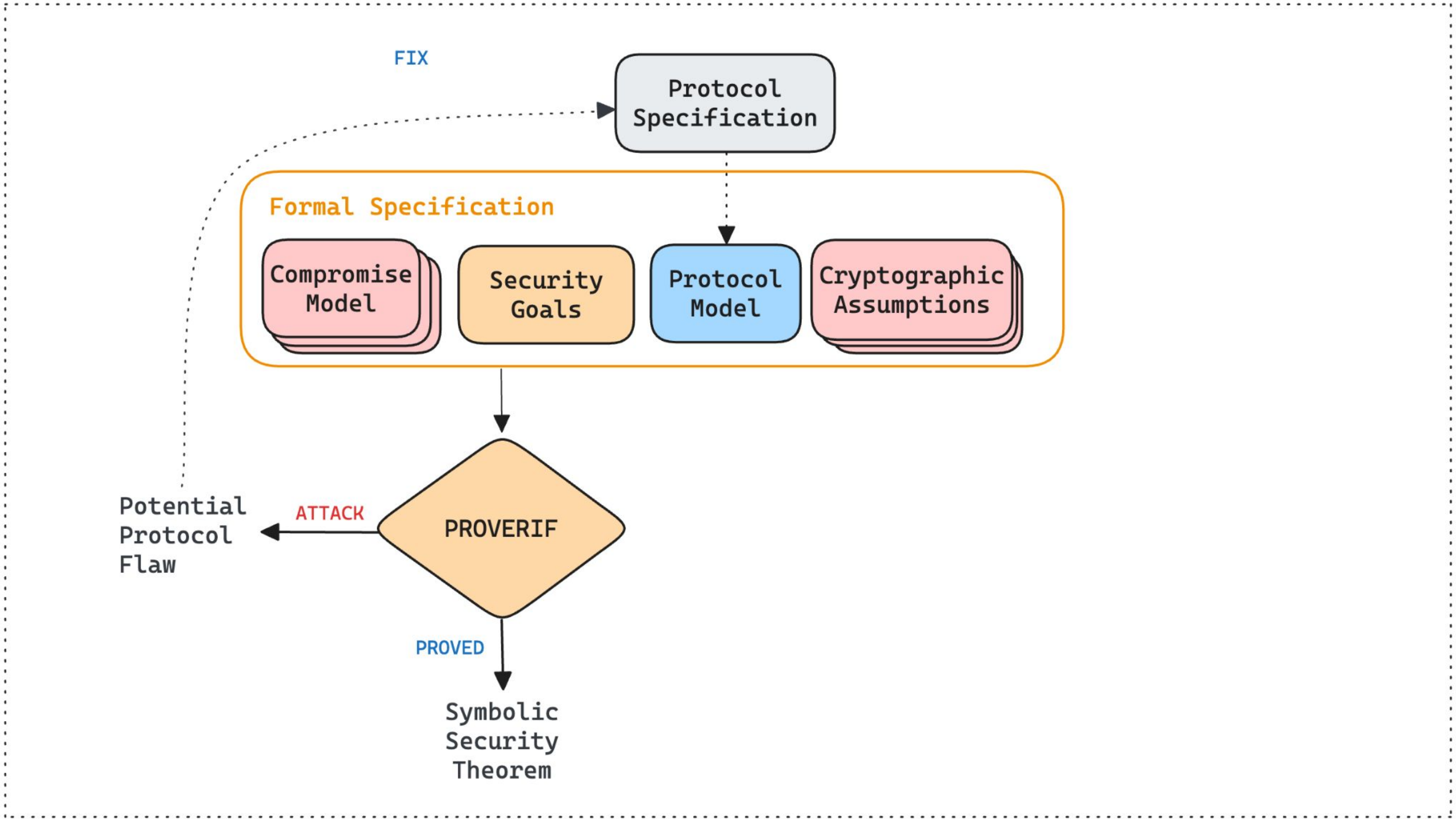
- PQXDH is a very small addition to X3DH.
- X3DH has been comprehensively analyzed in a variety of security models
  - Mutual Authentication, Confidentiality, (a form of) Forward Secrecy

- Is PQXDH as secure as X3DH?
- Is it secure against a Harvest-Now-Decrypt-Later quantum adversary?

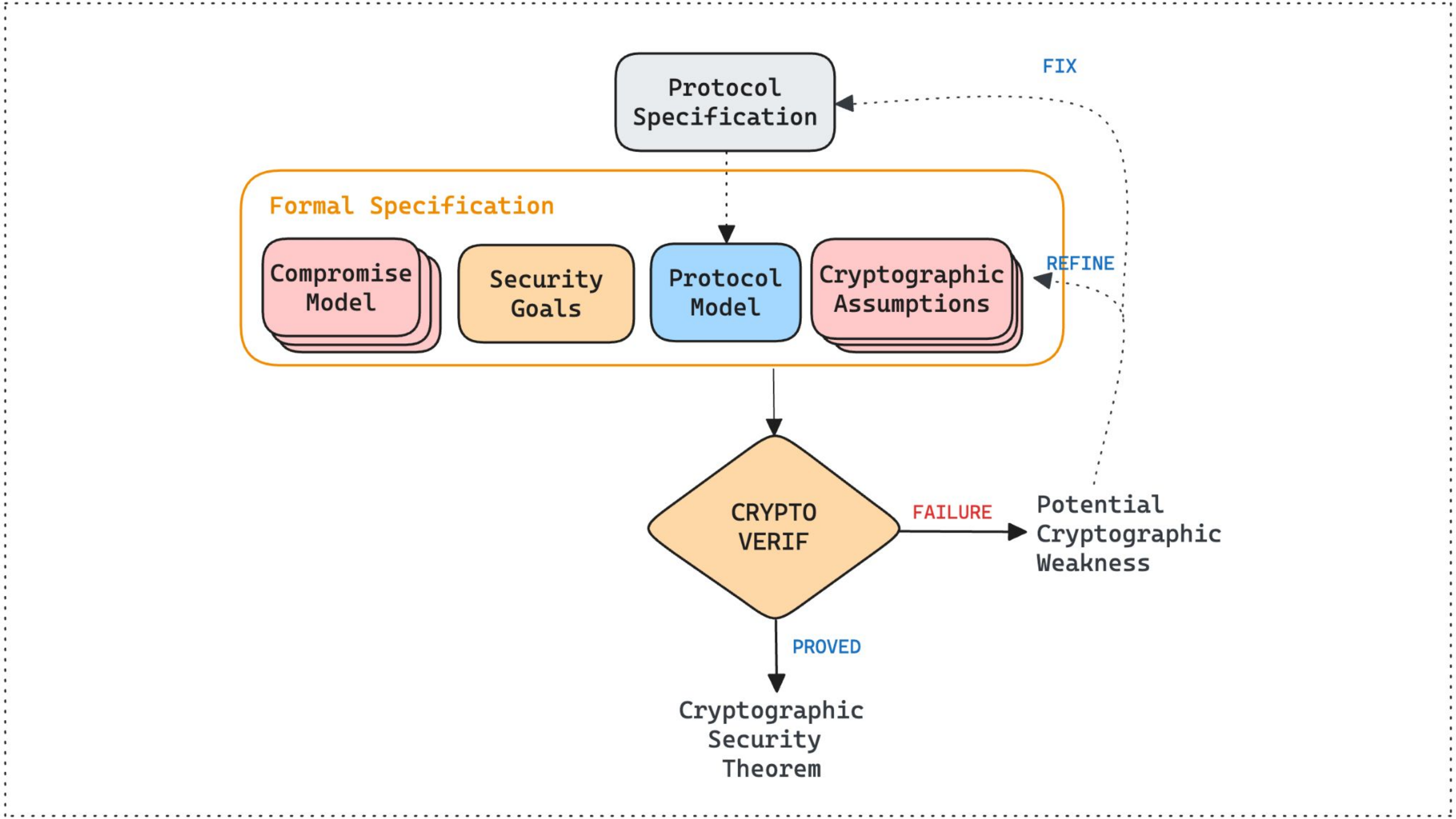
Our Formal Verification Methodology



Our Formal Verification Methodology



Our Formal Verification Methodology



Formally Specifying PQXDH

Single Message between Two Roles

- Arbitrary number of endpoints
- Any endpoint can play any role
- (Out-of-Band) Identity Key Verification
- Untrusted Key Distribution Server

Specification in Applied Pi Calculus

- Makes all computations precise.
- What is sent on the wire?
- What key encoding do we use?
- What exactly is signed/encrypted?
- How are all the keys derived?

```
let Initiator(i:client, IKA_s:scalar) =
    (* Download Responder Keys *)
    ...

    (* Verify the signatures *)
    if verify(IKB_p,encodeEC(SPKB_p),SPKB_sig) then
    if verify(IKB_p,encodeKEM(PQPKB_p),PQPKB_sig) then

    (* PQXDH Key Derivation*)
    let IKA_p = s2p(IKA_s) in
    let (CT:bitstring,SS:bitstring) =
        pqkem_enc(PQPKB_p) in (* PQ-KEM Encap *)
    new EKA_s:scalar;
    let EKA_p = s2p(EKA_s) in
    let DH1 = dh(IKA_s,SPKB_p) in
    let DH2 = dh(EKA_s,IKB_p) in
    let DH3 = dh(EKA_s,SPKB_p) in
    let DH4 = dh(EKA_s,OPKB_p) in
    let SK = kdf(concat5(DH1,DH2,DH3,DH4,SS)) in

    (* Send Message *)
    let ad = concatIK(IKA_p,IKB_p) in
    new msg_nonce: bitstring;
    let msg = app_message(i,r,msg_nonce) in
    let enc_msg = aead_enc(SK,empty_nonce,msg,ad) in

    out(server, (IKA_p,EKA_p,CT,OPKB_p,
                 SPKB_p,PQPKB_p,enc_msg))
```

Symbolic Analysis with ProVerif

Symbolic (Dolev-Yao) crypto model

- Algebraic model of crypto primitives
- User-defined equations define assumptions
- Unbounded number of sessions
- Non-deterministic (unbounded) attacker

Programmable threat model

- Can model new classes of attacks by adding new equations
- Can give attacker any internal state or keys by sending it on a public channel
- Can implement any dynamic attacker API by allowing attacker to drive protocol

```
(* Deterministic Public Key Encryption *)
fun kempk(kempriv):kempub.
fun penc(kempub,bitstring):bitstring.
fun pdec(kempriv,bitstring):bitstring

(* Equation for Decryption *)
reduc forall sk:kempriv,m:bitstring;
      pdec(sk,penc(kempk(sk),m)) = m.

(* Key Encapsulation Scheme *)
letfun pqkem_enc(pk:kempub) =
      new ss:bitstring;
      (penc(pk,ss),ss).

letfun pqkem_dec(sk:kempriv,ct:bitstring) =
      pdec(sk,ct).
```

```
process
     !in(att,i:identity); SetupInitiator(i)
   | !in(att,r:identity); SetupResponder(r)
   | !in(att,p:identity); KeyCompromise(p)
```

Symbolic Analysis with ProVerif

Security goals as queries

- Secrecy, Authentication as trace properties about protocol events & attacker knowledge
- Indistinguishability, privacy stated as equivalence properties between processes

Fully automated analysis

- Finds attacks and produces traces
- If no attack found in model, establishes a symbolic security theorem
- Might not terminate!

```
(* Post-Quantum Forward Secrecy Query *)
query A, B, spk, pqpk, sk, i, j;
      event(BlakeDone(A,B,spk,pqpk,sk))@i
            ⇒ not(attacker(sk))
              | (event(LongTermComp(A))@j & j < i)
              | (event(QuantumComp)@j & j < i)
```

Attack Trace:

1. Using the function info_x25519_sha512_kyber1024 the attacker may obtain info_x25519_sha512_kyber1024.
attacker(info_x25519_sha512_kyber1024).

2. Using the function zeroes_sha512 the attacker may obtain zeroes_sha512.
attacker(zeroes_sha512).

3. We assume as hypothesis that
attacker(a).

4. We assume as hypothesis that
attacker(b).

5. The message b that the attacker may have by 4 may be received at input {2}.
So the entry identity_pubkeys(b,SMUL(IK_s_2,G)) may be inserted in a table at i
table(identity_pubkeys(b,SMUL(IK_s_2,G))).
…

Game-Based Proofs with CryptoVerif

Computational crypto model

- Standard cryptographic assumptions
- User-defined assumptions as equivalences
- Probabilistic polynomial-time adversary

Process, security query syntax like ProVerif

- Secrecy, authentication, indistinguishability

Proofs as a sequence of game transformations

- Requires some manual guidance
- Machine-checked transformations
- Computes concrete advantage formulas
- Proof failure may indicate attack, no trace

```
proof {
crypto uf_cma_corrupt(sign) signAseed;
out_game "g1.cv" occ;

insert before "EKSecA1 <-R Z" ...
insert after "RecvOPK(" ...
out_game "g11.cv" occ;

insert after "OH_1(" ...
crypto rom(H2);
out_game "g2.cv" occ;

insert before "EKSecA1p <-R Z" ...
insert after "RecvNoOPK(" ...
out_game "g12.cv"occ;

insert after "OH(" ...
crypto rom(H1);
out_game "g3.cv";

crypto gdh(gexp_div_8) ...
crypto int_ctxt(enc) *;
crypto ind_cpa(enc) **;
out_game "g4.cv";

crypto int_ctxt_corrupt(enc) r_23;
crypto int_ctxt_corrupt(enc) r_50;
success
}
```

Modeling the Quantum Adversary

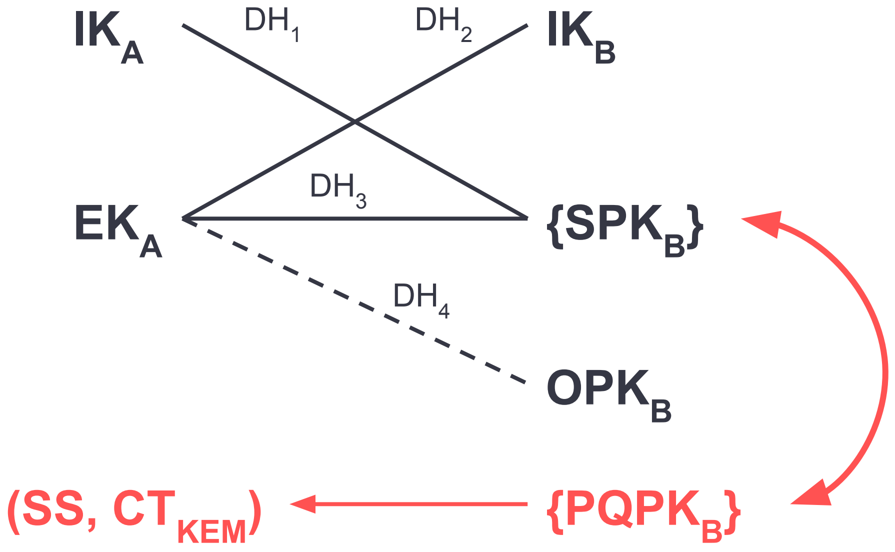
Passive Quantum Adversary Model (Harvest-Now-Decrypt-Later)

- We allow adversary to break certain crypto primitives (e.g. DH) *after* the session is over
- PQ primitives (e.g. PQ-KEM) remain secure

Symbolic and Computational Analysis

- ProVerif automatically searches for attacks that rely on broken primitives
- CryptoVerif checks that the classical game-based proof still holds against passive quantum attackers
    - *Post-quantum sound CryptoVerif and verification of hybrid TLS and SSH key-exchanges*, Blanchet, Jacomme, IEEE CSF 2024

Key Confusion Attack on PQXDH



$SK = KDF(DH_1 \| DH_2 \| DH_3 \| DH_4 \| SS)$

Attacker swaps keys and signatures to break PQ security of PQXDH

ProVerif finds this attack if:

- **the key encodings can collide, and**
- **public keys are not validated**

This is representative of a general class of cross-protocol attacks between classical modes and post-quantum crypto modes in the same protocol.

Easy Fix: Ensure all key/message/signature encodings have disjoint co-domains.

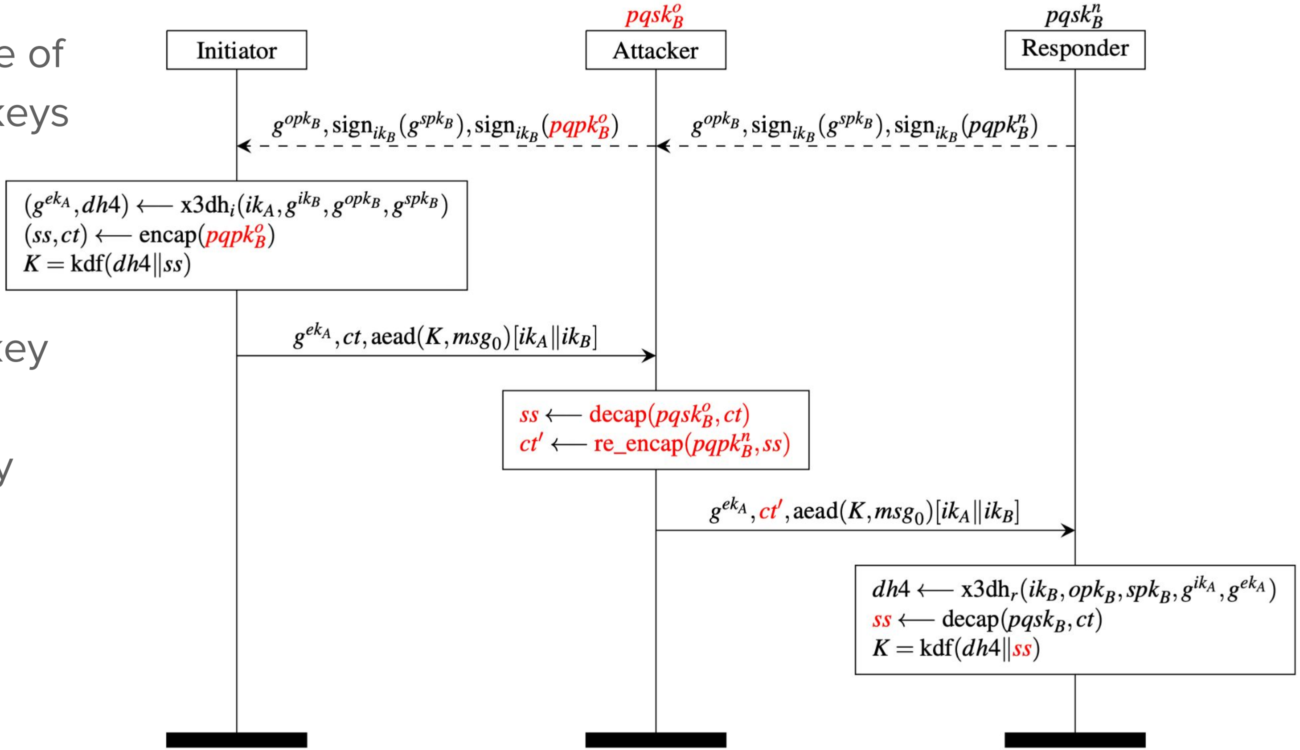
Signal implementation already does this

KEM Re-encapsulation Vulnerability in PQXDH

1. Attacker compromises one of responder's old PQ-KEM keys
2. Attacker provides old PQ-KEM key to initiator
3. Initiator encapsulates *SS* to (compromised) old key
4. Attacker re-encapsulates *SS* to responder's new key
5. Responder thinks initiator used new PQ public key

Breaks agreement query

(non-matching transcripts)



No agreement on the KEM public key.

No session independence:
A compromise of one of B’s PQSKs breaks HNDL security for A’s sessions with all other/future PQPKs of B.

Easy fix: Put the KEM public key in the AEAD associated data.

Insights on Public Key Binding in PQ-KEMS

- Kyber does not allow this attack, since it hashes the public key into SS!
- But IND-CCA is not enough to prevent KEM re-encapsulation attacks
- We formulate a new assumption: Semi-Honest Collision Resistance (SH-CR)
  - We prove that Kyber round 3 is SH-CR and
  - SH-CR is enough to ensure security of PQXDH.
- A real-world example to feed ongoing discussions on PQ-KEM bindings

$$\mathsf{Adv}^{SH\text{-}CR}_{\mathcal{A},KEM} = \Pr\left(\begin{array}{c} decaps(ct', \mathsf{sk}) = ss \wedge \\ (ct \neq ct' \vee \mathsf{pk} \neq \mathsf{pk}') \end{array} \;\middle|\; \begin{array}{l} \mathsf{sk}, \mathsf{pk} \stackrel{\$}{\leftarrow} keygen() \\ \mathsf{pk}' \stackrel{\$}{\leftarrow} \mathcal{A}(\mathsf{sk}) \\ ss, ct \stackrel{\$}{\leftarrow} encaps(\mathsf{pk}') \\ ct' \stackrel{\$}{\leftarrow} \mathcal{A}(\mathsf{sk}, ss, ct) \end{array} \right)$$

PQXDH Revision and Security Theorems

The findings and discussions with Signal team led to a new revision of the protocol:

- We required **AEAD** to be post-quantum **IND-CPA** and **INT-CTXT**
- Restricted the ranges of encodings to be disjoint
- Added **$PQPK_B$** to AD when it isn't already bound within the KEM

With these changes we can prove that PQXDH meets its classical and PQ security requirements in the symbolic, computational, and HNDL quantum models.

This whole process: spec, analysis, fix, proof, new spec took 1 calendar month.

Lessons from PQXDH

- A specification that is readable by both humans and machines is essential
  - A means of communication between developers and researchers
  - Useful for shaking out ambiguities and vulnerabilities
- Symbolic analysis can find new attacks, reconstruct suspected attacks
  - Systematically search through all possible key compromise scenarios
  - Find cross-protocol/downgrade attacks between PQXDH & X3DH
- Machine-checked game-based cryptographic proofs were not very hard
  - Gives new insights on KEM encapsulation assumptions
  - Able to handle (passive) quantum adversaries

Why PQXDH is the "easy" case

- Small protocol
  - 2 parties, 1 message, < 10 crypto applications
  - Adding the double ratchet, groups, state management etc. would make it much harder (especially for CryptoVerif)
- Well-understood crypto constructions to achieve simple security goals
  - DH, KDF, Sig, AEAD, KEM for Secrecy and Authentication
  - Analyzing Zero Knowledge, MPC, FHE for Privacy would be challenging
- Analysis at the level of a high-level design
  - Did not verify an executable specification or implementation
  - Formally proving that libsignal is secure is a much larger project

MLS: Analyzing harder protocol patterns

Messaging Layer Security

IETF secure messaging standard

- Groups with 10K+ members
- Forward secrecy
- Post-compromise security
- Efficient add, remove, update

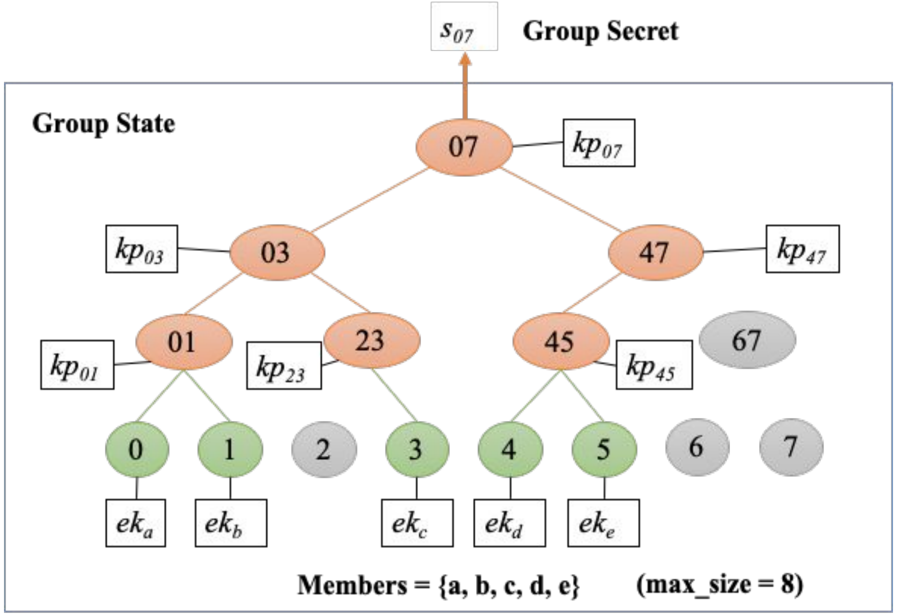
Tree-based protocol design

- Asynchronous Ratcheting Trees
  [Cohn-Gordon, Cremers, Garratt, Millican, Milner, 2018]
- TreeKEM [Bhargavan, Barnes, Rescorla, 2018]
  - Every member knows subgroup keys for the subtrees it belongs to
  - Key updates can be O(log N) PKEs rather than O(N) PKEs in Signal

Stream: Internet Engineering Task Force (IETF)
RFC: 9420
Category: Standards Track
Published: July 2023
ISSN: 2070-1721

Authors:
R. Barnes, Cisco
B. Beurdouche, Inria & Mozilla
R. Robert, Phoenix R&D
J. Millican, Meta Platforms
E. Omara
K. Cohn-Gordon, University of Oxford

RFC 9420
The Messaging Layer Security (MLS) Protocol



Formally Analyzing Messaging Layer Security

Many pen-and-paper proofs

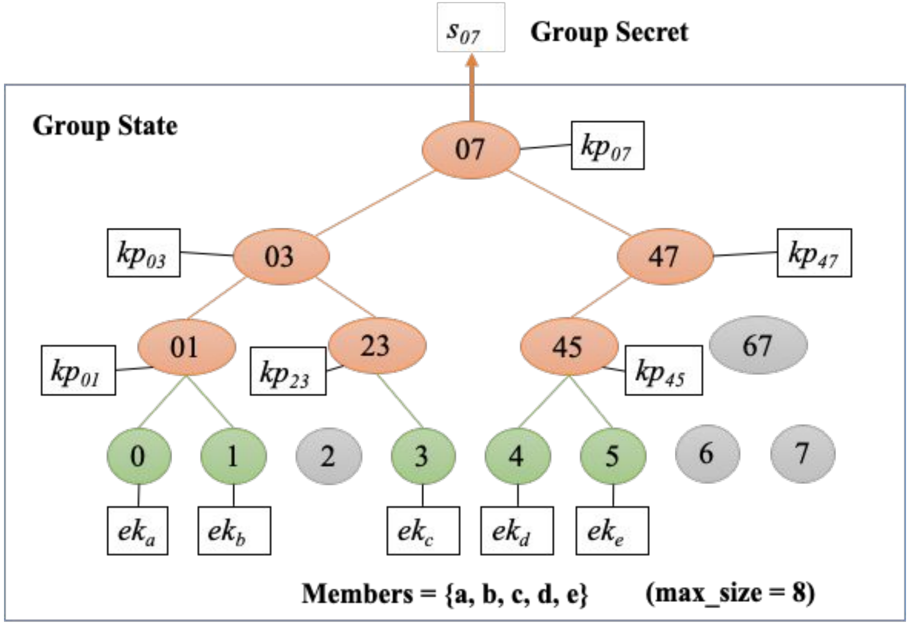
- Large proofs (30-70 pages each)
- Investigate different aspects of MLS
- Not machine-checked

Challenging for formal methods

- Recursive data structures
- All proofs need induction
- Many corner cases, asymmetries
- Beyond state of the art in 2018

Motivated a new approach: DY*

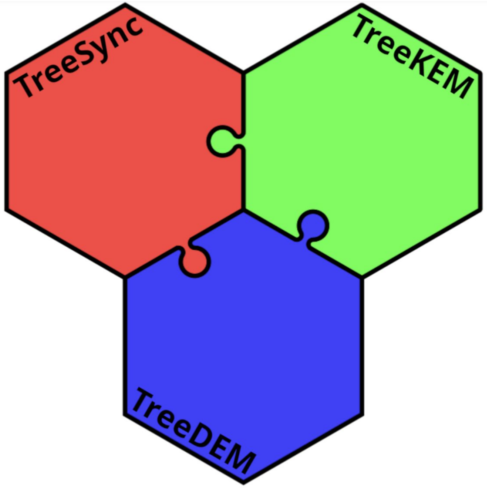
- Use a general-purpose proof assistant F* to build protocol security proofs



Formally Analyzing Messaging Layer Security

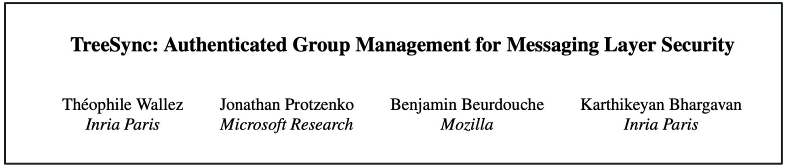
Modular executable specification in F*

- TreeSync, TreeKEM, TreeDEM
- Bit-level precision for all formats
- Passes test vectors, interoperates with other MLS implementations



Compositional proofs using DY*

- Semi-automated: needs manual guidance
- Uncovers new signature confusion attack
- Proves symbolic security theorems


TreeSync: Authenticated Group Management for Messaging Layer Security

Théophile Wallez, *Inria Paris* · Jonathan Protzenko, *Microsoft Research* · Benjamin Beurdouche, *Mozilla* · Karthikeyan Bhargavan, *Inria Paris*

Are your cryptographic formats secure?

MLS Signature Confusion Attack

- TreeSync and TreeDEM use same signature key
- Attacker can swap these signatures to break MLS
- **Issue:** signature format is ambiguous

Most analyses miss format confusion attacks

- Threema, Bitcoin, X.509, TLS 1.2, PQXDH
- Tedious to account for low-level encodings

Comparse: a systematic tool for verifying cryptographic formats [Wallez, Protzenko, Bhargavan]

- Domain-specific analysis tool for crypto formats
- Embedded within F*, compatible with DY*

Comparse: Provably Secure Formats for Cryptographic Protocols

Théophile Wallez
theophile.wallez@inria.fr
Inria
Paris, France

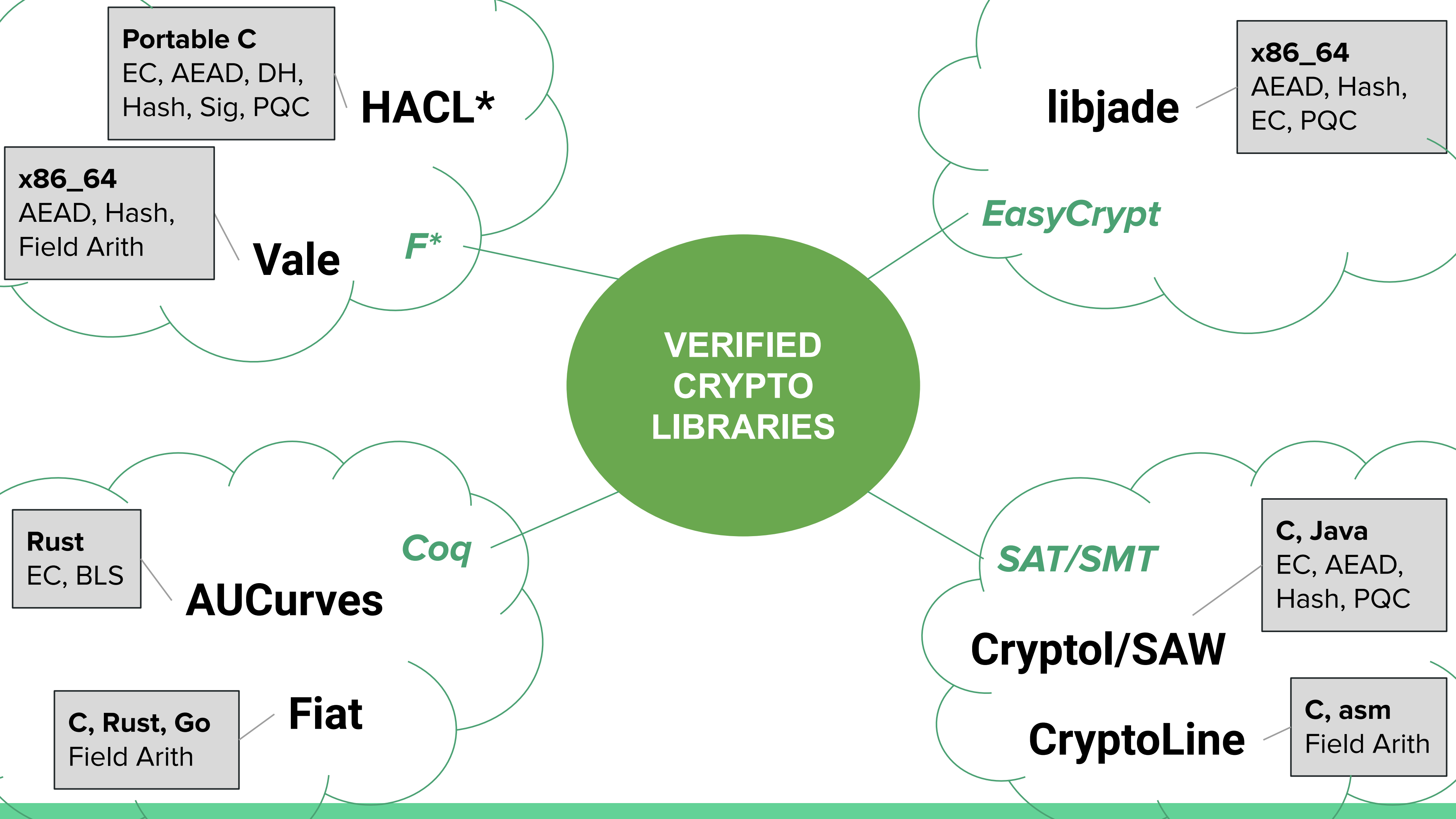
Jonathan Protzenko
protz@microsoft.com
Microsoft Research
Redmond, Washington, USA

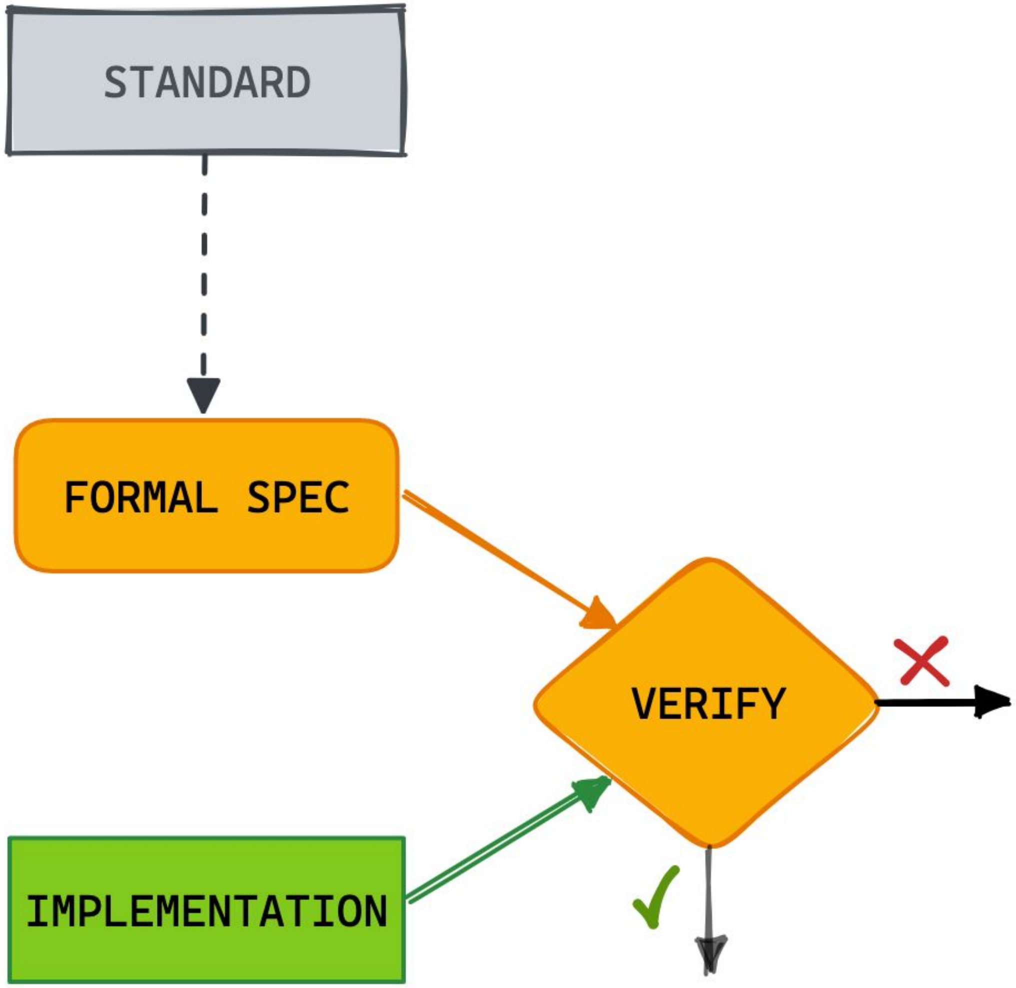
Karthikeyan Bhargavan
karthikeyan.bhargavan@inria.fr
Inria and Cryspen
Paris, France


```
struct {
    select (KeyExchangeAlgorithm) {
        case dhe_rsa, dhe_dss, // From TLS 1.2
             dhe_rsa_export, dhe_dss_export: // From TLS 1.0
            opaque client_random[32];
            opaque server_random[32];
            ServerDHParams params;
        case ec_diffie_hellman: // From TLS 1.2 ECC
            opaque client_random[32];
            opaque server_random[32];
            ServerECDHParams params;
    };
} TLS12SignatureInput;
```

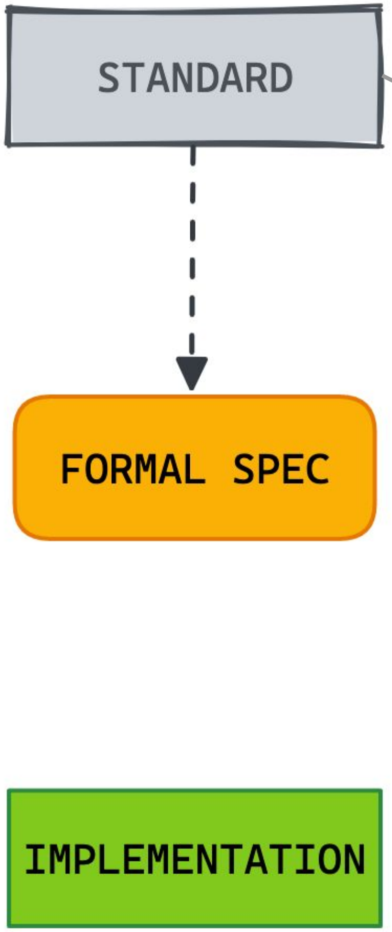
Three Lessons from Threema
Analysis of a Secure Messenger

Verifying Cryptographic Implementations


VERIFIED CRYPTO LIBRARIES
F*
HACL*
Portable C
EC, AEAD, DH, Hash, Sig, PQC
Vale
x86_64
AEAD, Hash, Field Arith
EasyCrypt
libjade
x86_64
AEAD, Hash, EC, PQC
Coq
AUCurves
Rust
EC, BLS
Fiat
C, Rust, Go
Field Arith
SAT/SMT
Cryptol/SAW
C, Java
EC, AEAD, Hash, PQC
CryptoLine
C, asm
Field Arith



Verified Cryptography Workflow




```
Internet Research Task Force (IRTF)                               Y. Nir
Request for Comments: 8439                                      Dell EMC
Obsoletes: 7539                                               A. Langley
Category: Informational                                      Google, Inc.
ISSN: 2070-1721                                                June 2018


                ChaCha20 and Poly1305 for IETF Protocols

Abstract

   This document defines the ChaCha20 stream cip
   of the Poly1305 authenticator, both as stand-
   a "combined mode", or Authenticated Encryptio
```

IETF RFC or NIST Standard

```
2.1.  The ChaCha Quarter Round

   The basic operation of the ChaCha algorithm is the quarter round.  It
   operates on four 32-bit unsigned integers, denoted a, b, c, and d.
   The operation is as follows (in C-like notation):

      a += b; d ^= a; d <<<= 16;
      c += d; b ^= c; b <<<= 12;
      a += b; d ^= a; d <<<= 8;
      c += d; b ^= c; b <<<= 7;
```

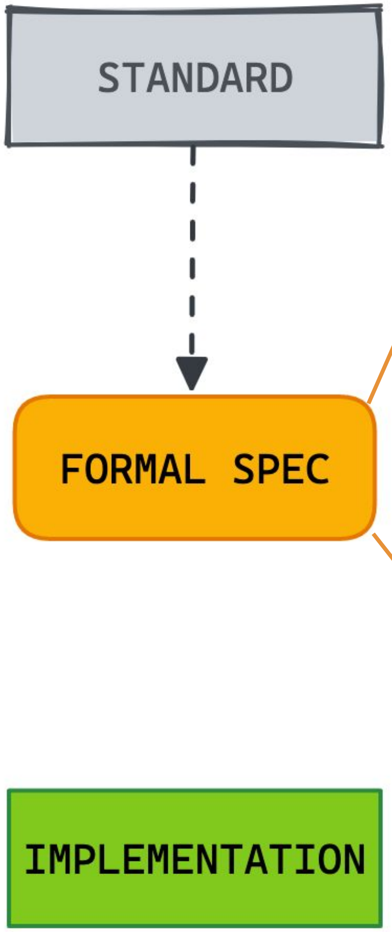
In English + Pseudocode

```
2.1.1.  Test Vector for the ChaCha Quarter Round

   For a test vector, we will use the same numbers as in the example,
   adding something random for c.

      a = 0x11111111
      b = 0x01020304
      c = 0x9b8d6f43
      d = 0x01234567
```

+ Test Vectors




```
let line (a:idx) (b:idx) (d:idx) (s:rotval U32) (m:state) : Tot state =
  let m = m.[a] ← (m.[a] +. m.[b]) in
  let m = m.[d] ← ((m.[d] ^. m.[a]) <<<. s) in m

let quarter_round a b c d : Tot shuffle =
  line a b d (size 16) @
  line c d b (size 12) @
  line a b d (size 8)  @
  line c d b (size 7)
```

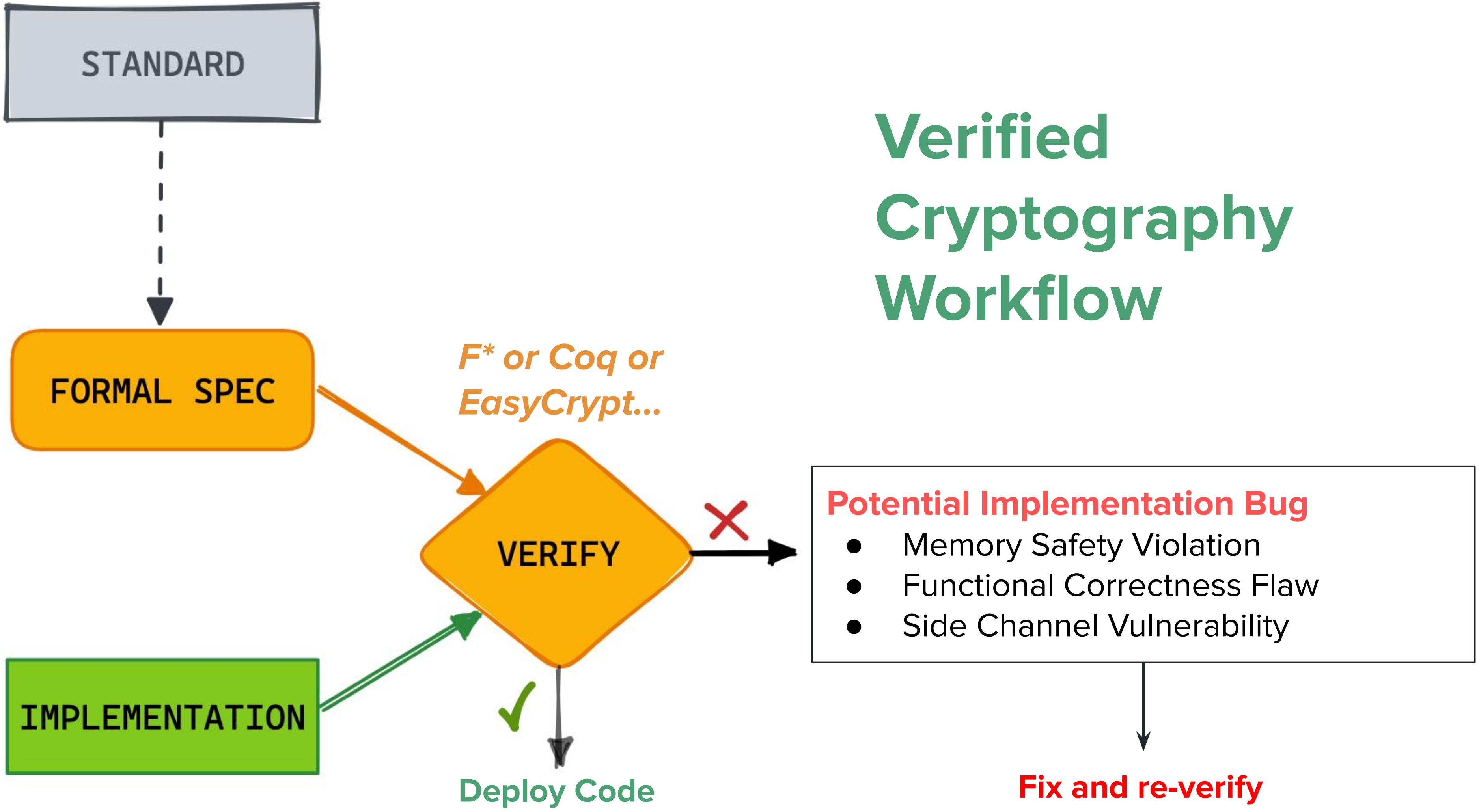
F* Spec
(HACL*)

```
proc chacha20_line(a : int, b : int, d : int, s : int, st : State) = {
  var state;
  state <- st;
  state.[a] <- ((state).[a]) + ((state).[b]);
  state.[d] <- ((state).[d]) `^` ((state).[a]);
  state.[d] <- rotate_left ((state).[d]) (s);
  return state;
}

proc chacha20_quarter_round(a : int, b : int, c : int, d : int, st : State) = {
  var state;
  state <@ chacha20_line (a, b, d, 16, st);
  state <@ chacha20_line (c, d, b, 12, state);
  state <@ chacha20_line (a, b, d, 8, state);
  state <@ chacha20_line (c, d, b, 7, state);
  return state;
}
```

EasyCrypt Spec
(libjade)

Verified Cryptography Workflow



Good news: For any modern crypto algorithm, there is probably a verified implementation

- You don’t have to sacrifice performance
- Mechanized proofs that you can run and re-run yourself
- You (mostly) don’t have to read or understand the proofs
- Formally verified crypto in NSS, BoringSSL, aws-lc, ...

HACL* and libcrux [2017-2024]

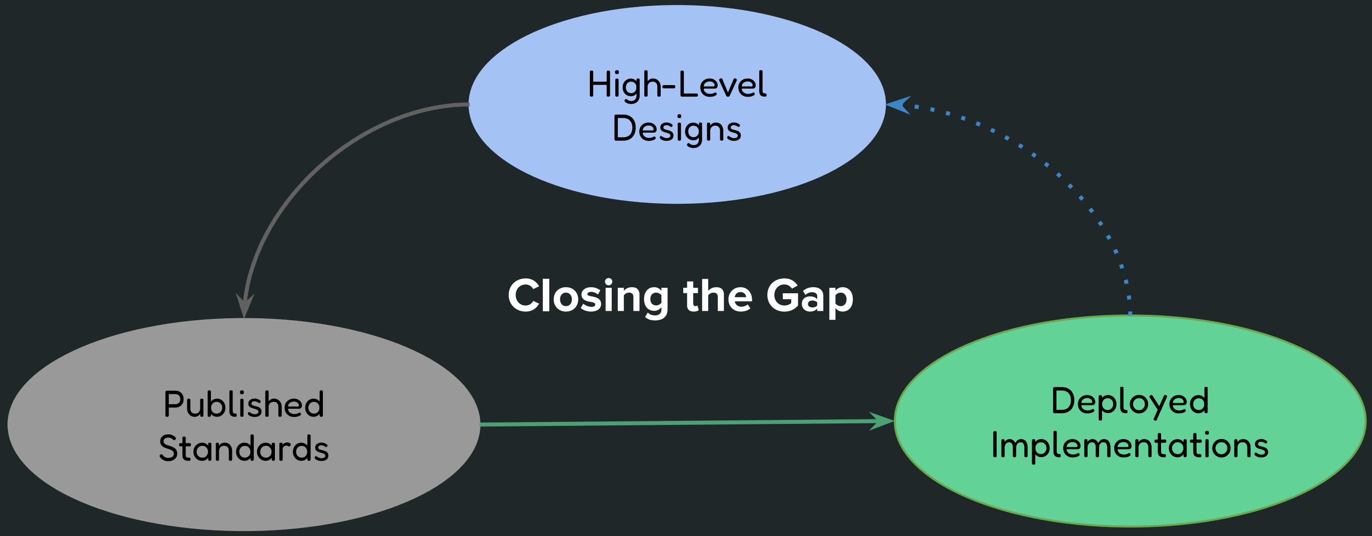
Full library of algorithms

- Entire TLS ciphersuites
- C, asm, wasm, and Rust
- Deployed: NSS, linux, python, wireguard, …

Constantly evolving

- PQ Crypto: Frodo, ML-KEM
- Constructions: XOF, HPKE
- Proofs run on CI
- Every night, every tool update, every code change, is re-verified

Algorithm	Portable C code	Arm A64 Neon	Intel x64			
			AVX	AVX2	AVX512	Vale
AEAD						
Chacha20-Poly1305	✓ [43] (+)	✓ (*)	✓ (*)	✓ (*)	✓ (*)	
AES-GCM						✓ [20]
Hashes						
SHA-224,256	✓ [43] (+)	✓ (*)	✓ (*)	✓ (*)	✓ (*)	✓ [20]
SHA-384,512	✓ [43] (+)	✓ (*)	✓ (*)	✓ (*)	✓ (*)	
Blake2s, Blake2b	✓ [34] (+)	✓ (*)	✓ (*)	✓ (*)		
SHA3-224,256,384,512	✓ [34]					
HMAC and HKDF						
HMAC (SHA-2,Blake2)	✓ [43]	✓ (*)	✓ (*)	✓ (*)	✓ (*)	
HKDF (SHA-2,Blake2)	✓ [43]	✓ (*)	✓ (*)	✓ (*)	✓ (*)	
ECC						
Curve25519	✓ [43]					✓ [34]
Ed25519	✓ [43]					
P-256	✓ [34]					
High-level APIs						
Box	✓ [43]					
HPKE	✓ (*)	✓ (*)	✓ (*)	✓ (*)	✓ (*)	✓ (*)


High-Level
Designs
Closing the Gap
Published
Standards
Deployed
Implementations

The End-to-End Dream: Linking all the proofs

Choose one proof framework and do all your proofs in it

- EasyCrypt + jasmin

  Formally Verifying Kyber Episode V: Machine-checked IND-CCA security and correctness of ML-KEM in EasyCrypt [Almeida, Olmos, Barbosa, Barthe, Dupressoir, Gregoire, Laporte, Léchenet, Low, Oliveira, Pacheco, Quaresma, Schwabe, Strub, CRYPTO 2024]

- or F* + DY* + HACL* + Comparse
- or Coq + Fiat-Crypto + SSProve

But what if different tools are better for different tasks?

Practical Formal Methods for Crypto Engineers

Do not force a premature choice of tool or language

- Use different tools for different tasks
- Derive specifications from code-like artifacts

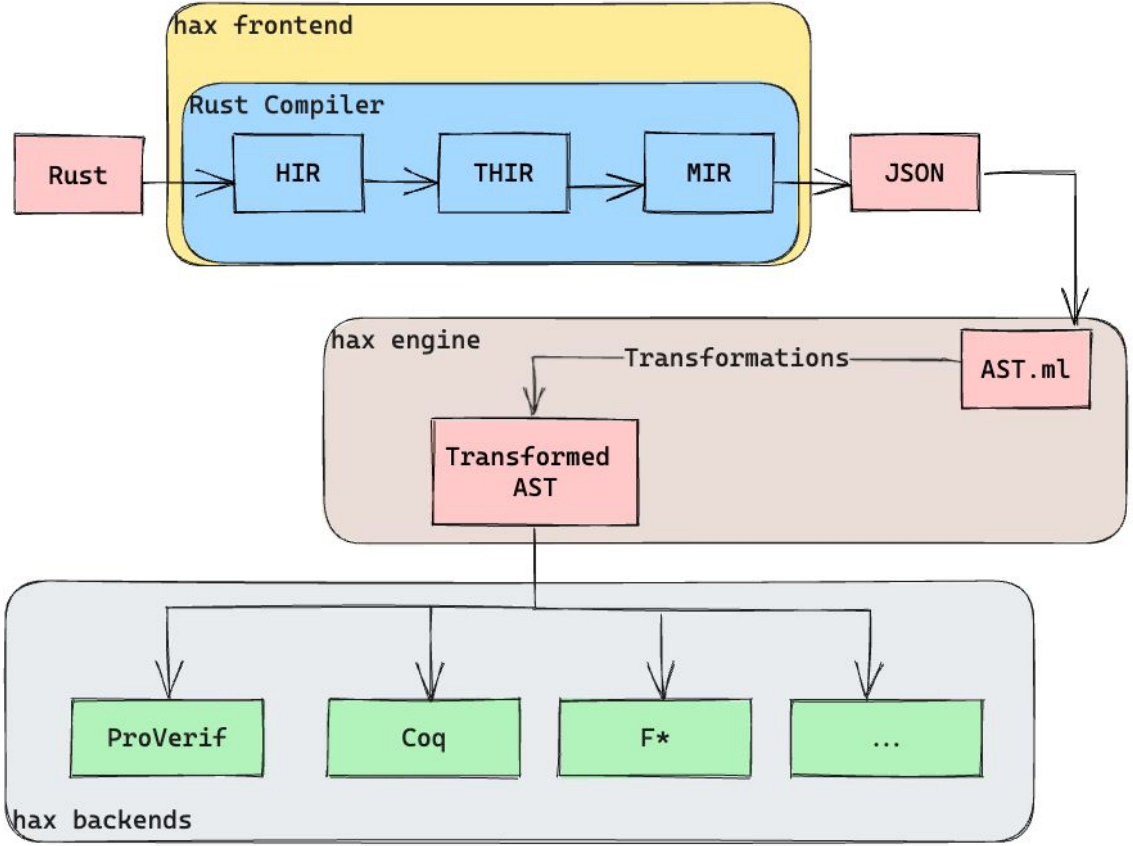
Invest in verification tool engineering

- Intuitive interfaces, good documentation
- Predictable outputs, documented limitations

Develop custom tools for each problem domain

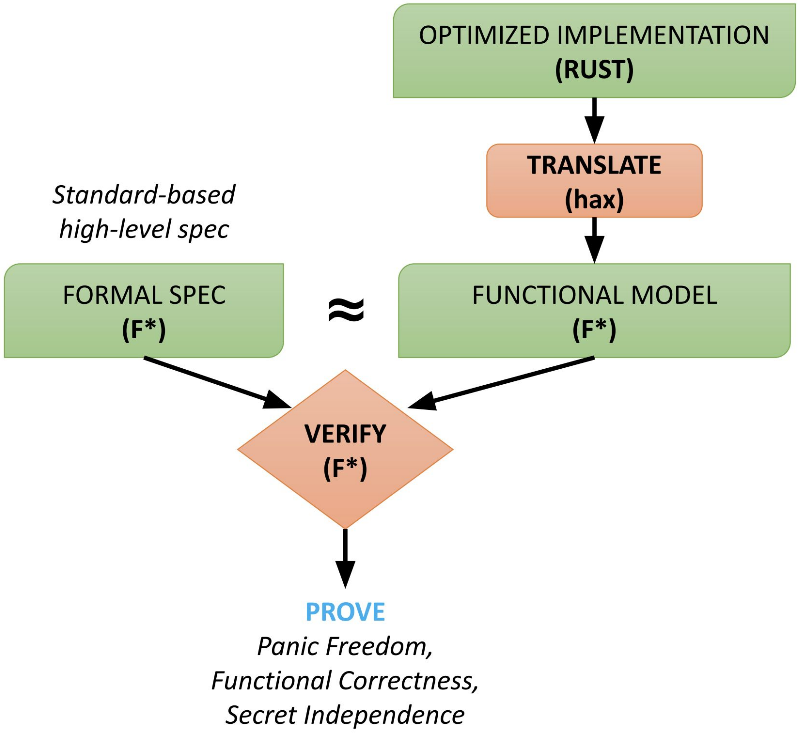
- New libraries, optimizations, proof tactics
- Easy things should be easy, hard things can take time

hax: bridging the gap between code and proof



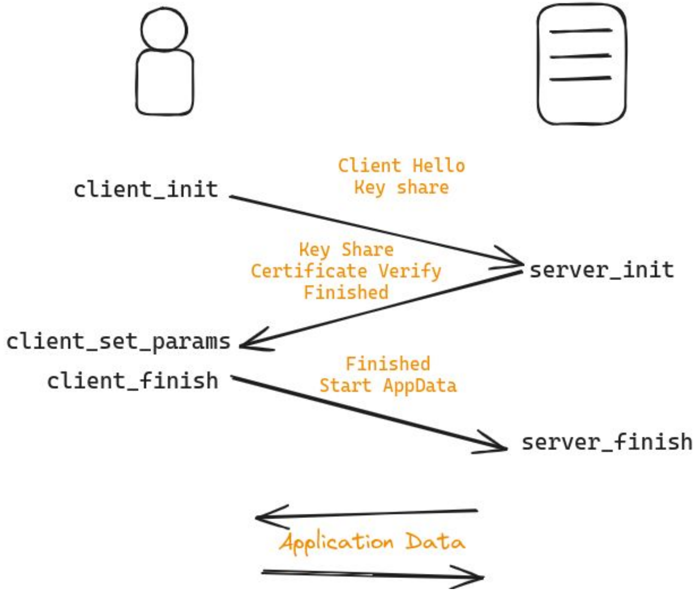
Verifying ML-KEM in Rust using hax

- A fast implementation of ML-KEM
  - Portable, AVX2, Neon
- Compile Rust code to F* with hax
  - Prove Panic Freedom
  - Prove Functional Correctness
  - Prove Secret Independence
  - Found KyberSlash bug
- Compile Rust code to C
  - Deployment in NSS (Mozilla), BoringSSL (Google), ...



Verifying a TLS 1.3 implementation using hax

- A compact implementation of TLS 1.3
  - messaging functions, state machines, message formats, ...
- Protocol code compiled to ProVerif
  - Add compromise assumptions
  - Add security goals
  - Verify to find security bugs in the code
- Messaging code compiled to F*
  - Verify for correctness, ambiguities



Exciting New Directions

Symbolic Protocol Verification: ProVerif, Tamarin, DY*, ...

- More precise attacker models to find new classes of attacks
- More automation for induction, support for recursive data structures
- Symbolic verification directly for protocol implementations

Machine-checked Crypto Proofs: CryptoVerif, EasyCrypt, Squirrel, ...

- Proofs against quantum adversaries in the QROM
- **Squirrel:** symbolic methods for computational crypto proofs
  - Based on the Computationally Complete Symbolic Attacker approach

    The Squirrel Prover and its Logic, ACM SIGLOG, 2024

Exciting New Directions

Cryptographic Code Verification: HACL*, libjade, Fiat-Crypto, aws-lc, ...

- Formal verification becomes the default for modern crypto libraries
- New side channel analyses for low-level cryptography

Protocol Code Verification: hax/ProVerif, SSProve/Coq, Verus/Owl, ...

- Scale code verification to large protocol implementations in Rust/C
- Analyze state machines, formats, APIs, information flow, protocol security

Lightweight Formal Methods: hacspec, ProofFrog ...

- Make it easier to write formal specs, check manual crypto proofs

Conclusions

- Deploying new crypto requires a lot of social communication
  - Precise, readable, well-documented specifications help
  - Be able to explain the security argument/theorem to non-experts
  - Designs keep evolving, so maintain your proofs like software

- Formal methods can help answer questions about crypto artifacts
  - Symbolic and computational proofs, software verification
  - Systematic tool-based analyses can help head off issues early
  - Invest in learning a proof tool, customize it for your use case!

Questions?

- *SoK: Computer-Aided Cryptography*
  [Barbosa, Barthe, Bhargavan, Blanchet, Cremers, Liao, Parno, IEEE S&P 2021]
- *A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello*
  [Bhargavan, Cheval, Wood, ACM CCS 2022]
- *Formal verification of the PQXDH Post-Quantum key agreement protocol for end-to-end secure messaging*
  [Bhargavan, Jacomme, Kiefer, Schmidt, Usenix Security 2024]
- *DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code.*
  [Bhargavan, Bichhawat, Do, Hosseyni, Küsters, Schmitz, Würtele, Euro S&P 2021]
- *TreeSync: Authenticated Group Management for Messaging Layer Security*
  [Wallez, Protzenko, Beurdouche, Bhargavan, Usenix Security 2023]
- *Comparse: Provably Secure Formats for Cryptographic Protocols*
  [Wallez, Protzenko, Bhargavan, ACM CCS 2023]

CRYSPEN

https://cryspen.com