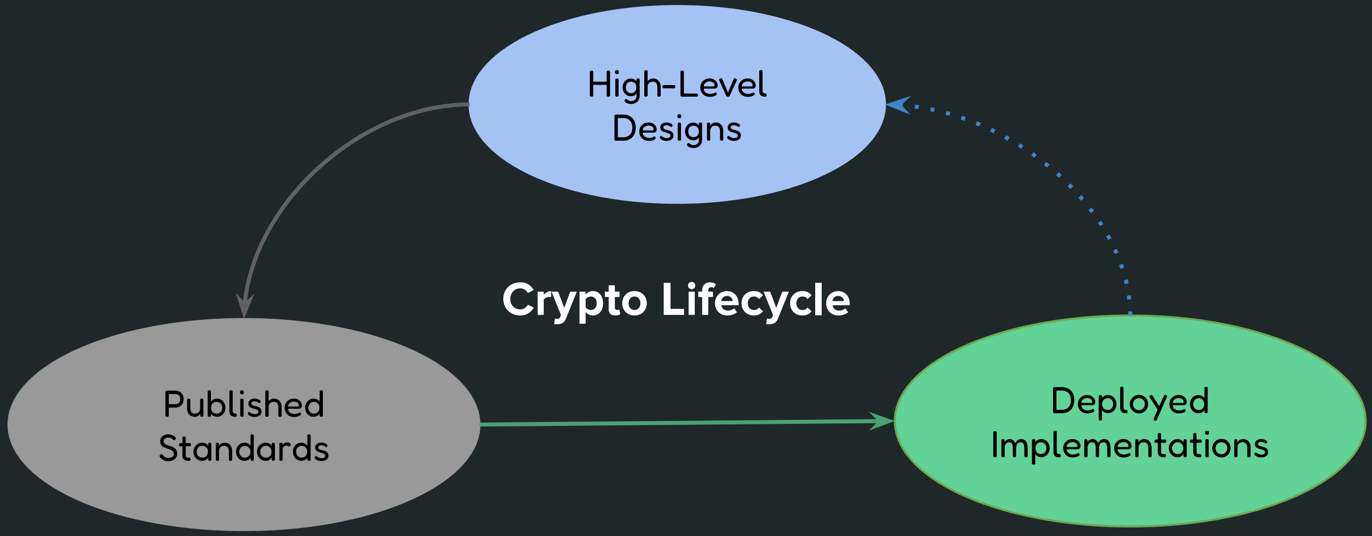
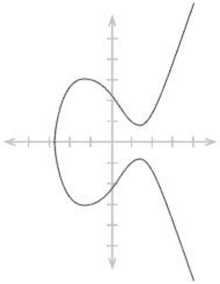
Provably Secure Protocols and High-Assurance Software for the Post-Quantum Transition

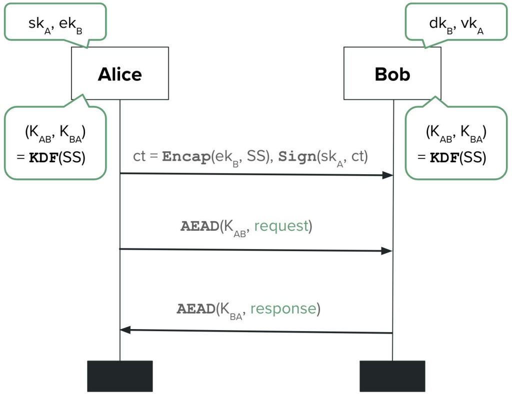
Karthikeyan Bhargavan


High-Level Designs
Crypto Lifecycle
Published Standards
Deployed Implementations

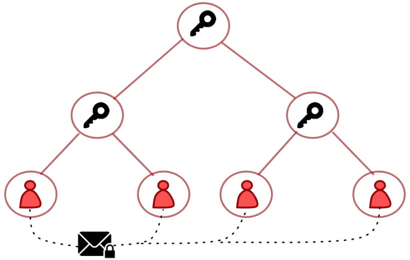
High-Level Cryptographic Design



- Novel constructions and protocols
- Abstract mathematical descriptions
- Security goals and threat models
- Is the math / proof correct?



- **Documentation:** white paper, publication
- **Evaluation**: **security proofs**, cryptanalysis
- **Examples:**
  - Elliptic Curves, **PQ KEMs**, Signatures
  - Key Exchange, Double Ratchet, TreeKEM



Published Standards

- Detailed technical specifications
- Algorithms, formats, data structures
- Enable interoperable implementations
- Is it ambiguous? Is it secure?

- **Documentation:** spec, standard
- **Evaluation**: public review, **formal analysis**
- **Examples:**
  - ECDSA, ML-KEM, HPKE
  - TLS, MLS, Signal, iMessage

FIPS 203

Federal Information Processing Standards Publication

Module-Lattice-Based Key-Encapsulation Mechanism Standard

Published:	February 2022			
ISSN:	2070-1721			
Authors:	R. Barnes	K. Bhargavan	B. Lipp	C. Wood
	Cisco	*Inria*	*Inria*	*Cloudflare*

RFC 9180

Hybrid Public Key Encryption

Signal

The PQXDH Key Agreement Protocol

Revision 3, 2023-05-24, Last Updated: 2024-01-23 [PDF]

Ehren Kret, Rolfe Schmidt

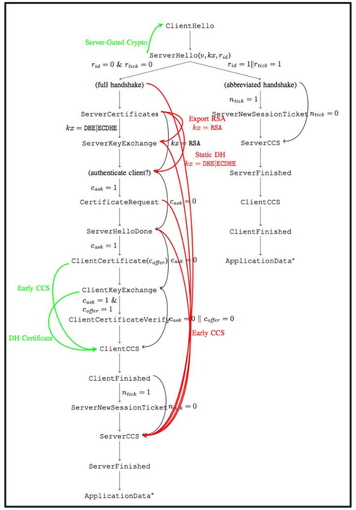
Deployed Implementations

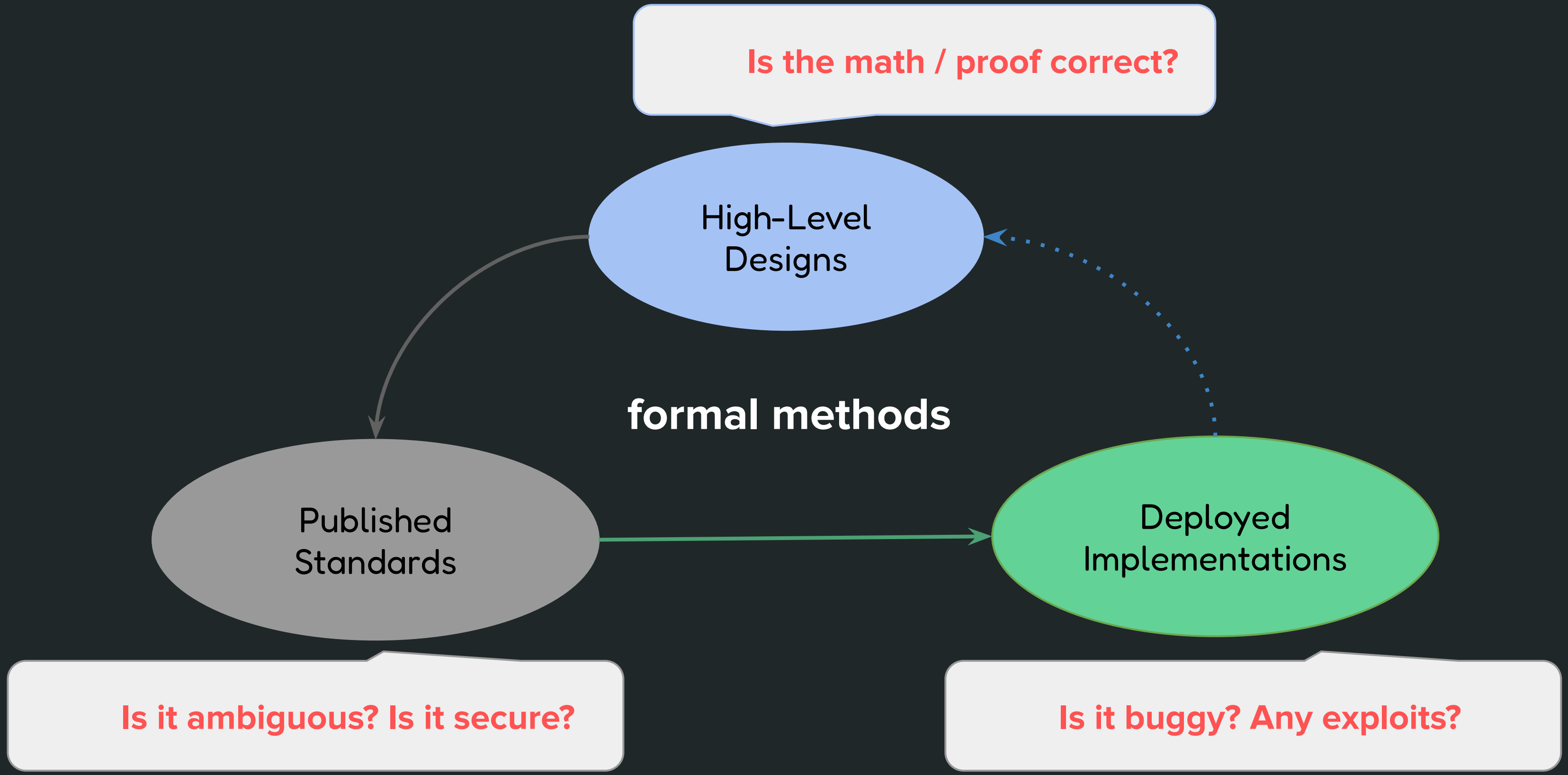
OpenSSL
Cryptography and SSL/TLS Toolkit

- Efficient code and hardware
- Usable APIs, side channel protections
- Integrate with real-world applications
- Is it buggy? Any exploitable attacks?

- **Documentation:** code, documentation
- **Evaluation**: testing, **software verification**
- **Examples:**
  - OpenSSL, boringssl, NSS, aws-lc, **libcrux**
  - Libsignal, miTLS, OpenMLS, **bertie**

```
for(i=0;i<KYBER_N/8;i++) {
  for(j=0;j<8;j++) {
    // map to positive standard representatives
    u  = a->coeffs[8*i+j];
    u += (u >> 15) & KYBER_Q;
    t[j] = ((((uint16_t)u << 4) + KYBER_Q/2)/KYBER_Q) & 15;
  }

  r[0] = t[0] | (t[1] << 4);
  r[1] = t[2] | (t[3] << 4);
  r[2] = t[4] | (t[5] << 4);
  r[3] = t[6] | (t[7] << 4);
  r += 4;
}
```




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

Formal Methods for Crypto

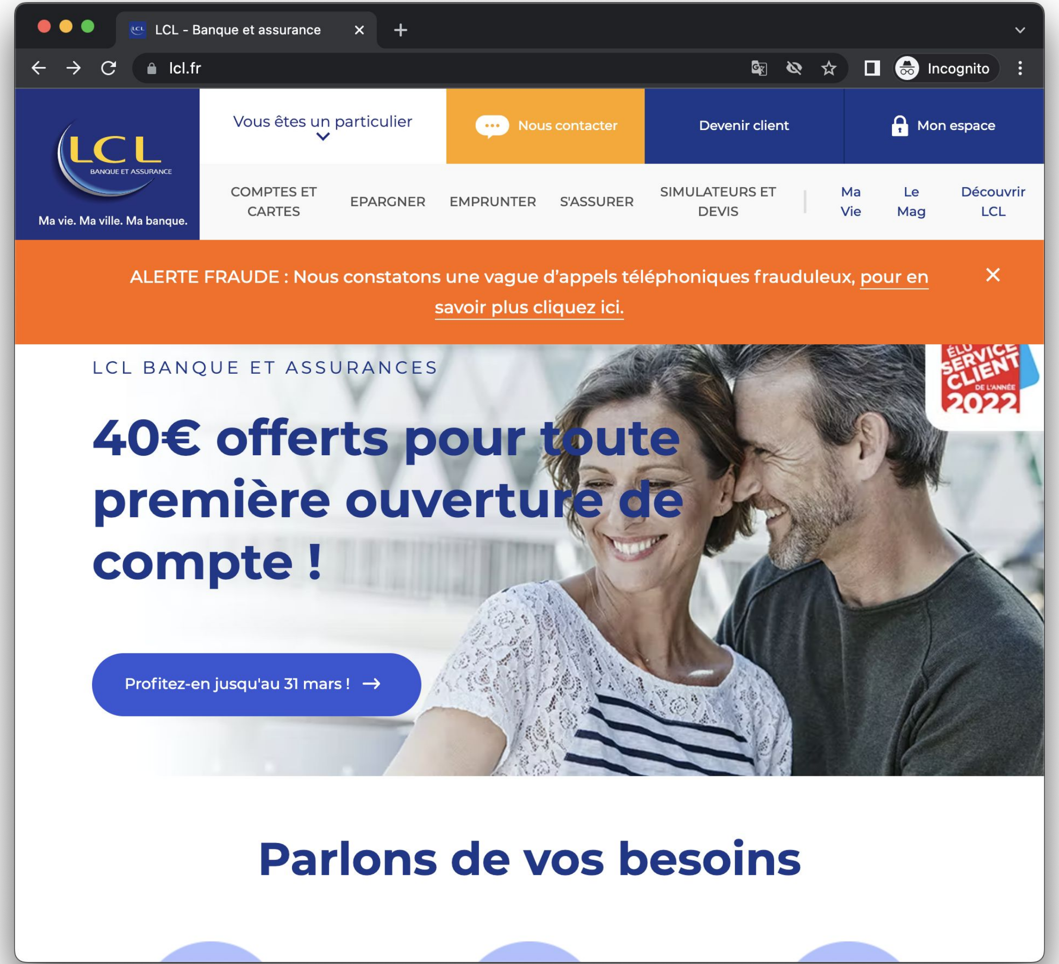
- Computer-Aided Cryptography, a.k.a. High Assurance Cryptography

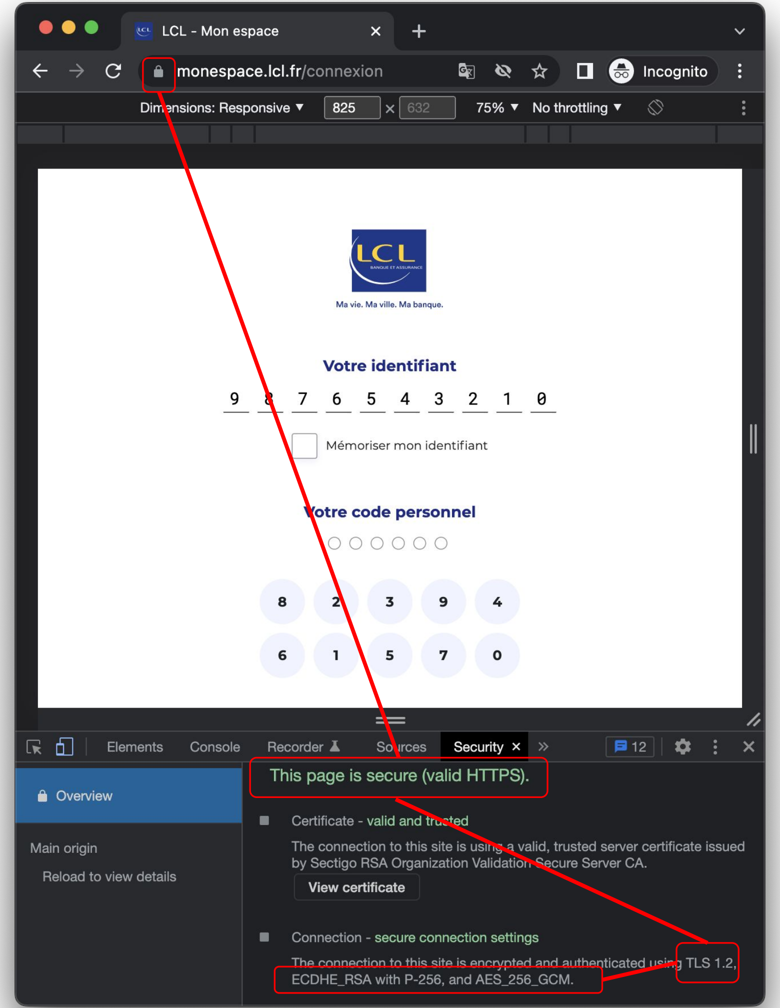
  "Applying formal, **machine-checkable** approaches to the design, analysis, and implementation of cryptography."

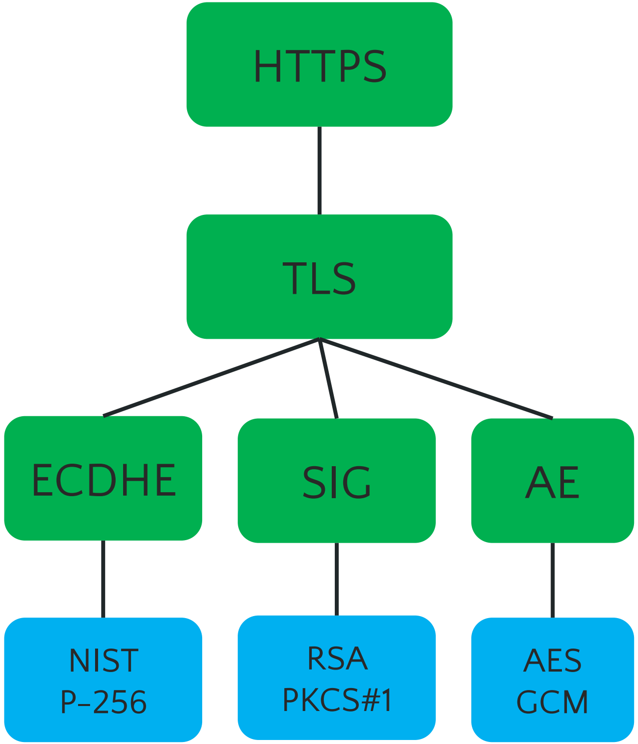
  SoK: Computer-Aided Cryptography, IEEE S&P 2021
  Barbosa, Barthe, Bhargavan, Blanchet, Cremers, Liao, Parno

- Analyze cryptographic **designs** early to find vulnerabilities
- Comprehensively analyze **specifications** before publication
- Formally verify **implementations** to prevent bugs and attacks
- .... and **repeat** these steps over and over again as these artifacts evolve

TLS 1.3: An Instructive Example


LCL - Banque et assurance
lcl.fr
Incognito
LCL
BANQUE ET ASSURANCE
Ma vie. Ma ville. Ma banque.
Vous êtes un particulier
Nous contacter
Devenir client
Mon espace
COMPTES ET CARTES
EPARGNER
EMPRUNTER
S'ASSURER
SIMULATEURS ET DEVIS
Ma Vie
Le Mag
Découvrir LCL
ALERTE FRAUDE : Nous constatons une vague d'appels téléphoniques frauduleux, pour en savoir plus cliquez ici.
LCL BANQUE ET ASSURANCES
40€ offerts pour toute première ouverture de compte !
Profitez-en jusqu'au 31 mars !
ÉLU SERVICE CLIENT DE L'ANNÉE 2022
Parlons de vos besoins





- Well-understood since 1994!
- So, is this a solved problem?

Many Practical Attacks on TLS (until TLS 1.3)

• BEAST	CBC predictable IVs	[Sep’11]
• CRIME	Compression before Encryption	[Sep’12]
• RC4	Keystream biases	[Mar’13]
• Lucky 13	MAC-Encode-Encrypt CBC	[May’13]
• 3Shake	Insecure resumption	[Apr’14]
• POODLE	SSLv3 MAC-Encode-Encrypt	[Dec’14]
• FREAK	Export-grade 512-bit RSA	[Mar’15]
• LOGJAM	Export-grade 512-bit DH	[May’15]
• SLOTH	RSA-MD5 signatures	[Jan’16]
• DROWN	SSLv2 RSA-PKCS#1v1.5	[Mar’16]
• SWEET32	3DES collisions	[May’16]
• …		

What goes wrong in protocols like TLS?

Crypto Weaknesses

- RC4, 3DES, MD5, PKCS#1 v1.5

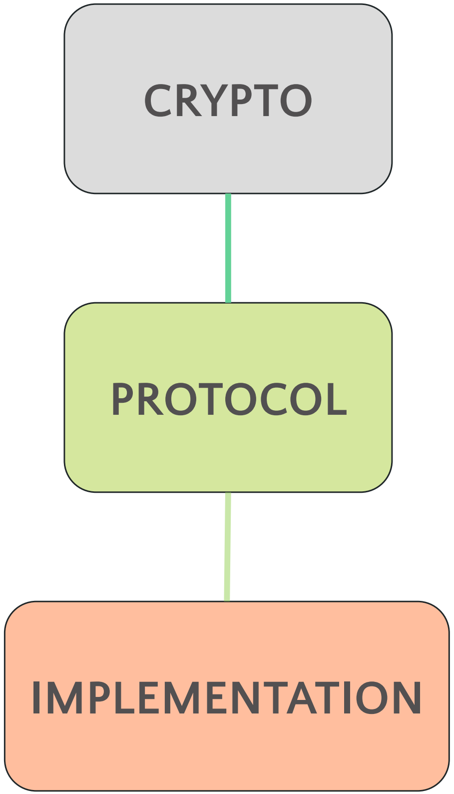
Protocol Design Flaws

- Downgrade attacks, Transcript collisions

Implementation Bugs

- Parsing, State machines, Memory safety

Often, a mix of all the above!

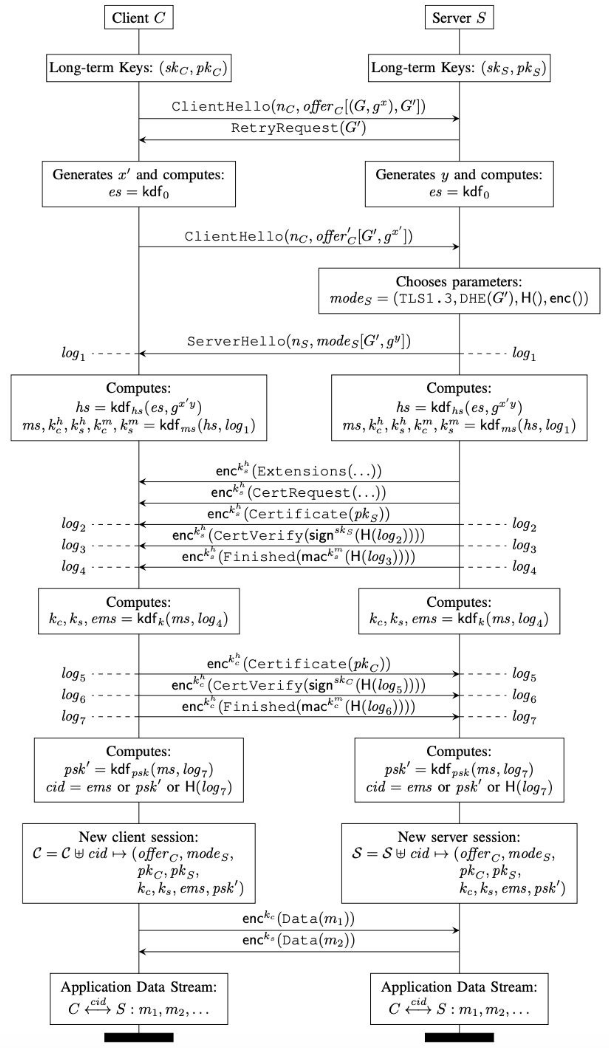


The TLS 1.3 Standardization Project

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

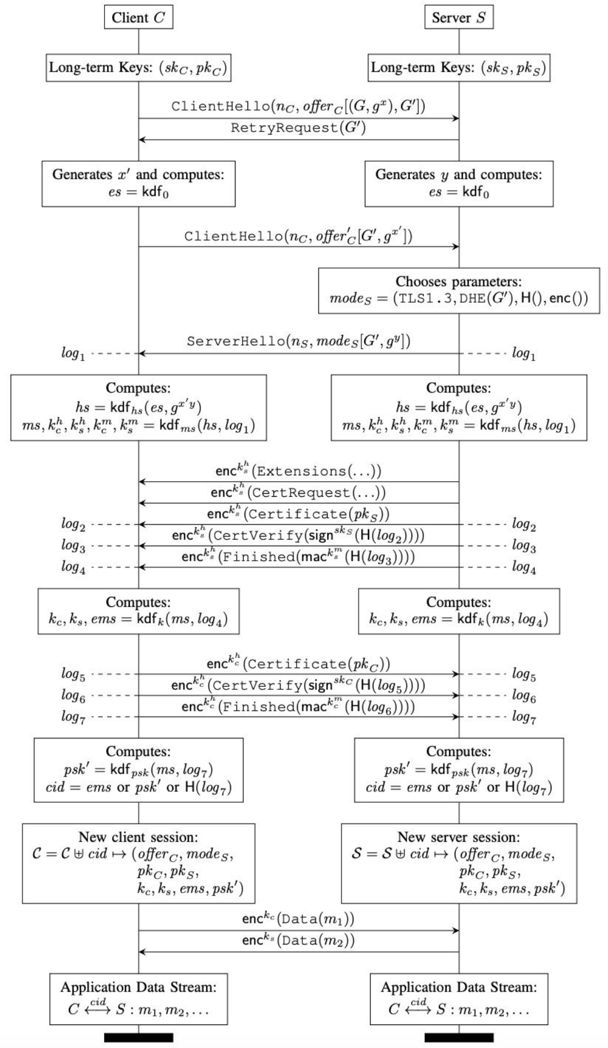


TLS 1.3: Impact on new protocols

- Proofs now often required for new designs
  - **Machine-checked proofs** preferred

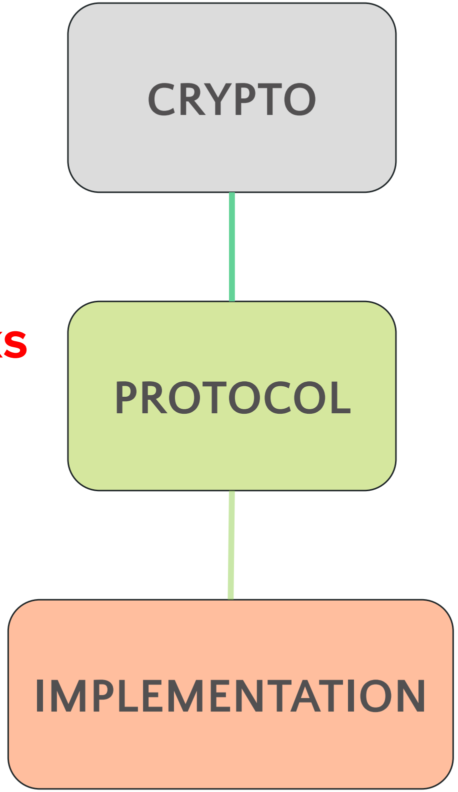
- IETF Working Groups
  - **LAKE**: Key exchange protocol for IoT
  - **TLS**: Encrypted Client Hello, KEM-TLS
  - **MLS:** Secure group messaging

- Industrial Protocols
  - PQ3 (iMessage), PQXDH (Signal)



Lessons from TLS 1.3

- Tight **Design** ↔ **Analysis** exchanges are crucial
- It's not just crypto, **low-level protocol details** matter
- Beware of composition flaws and **downgrade attacks**
- Prepare to address the **next threat** quickly
- **Formal methods** can give you an edge



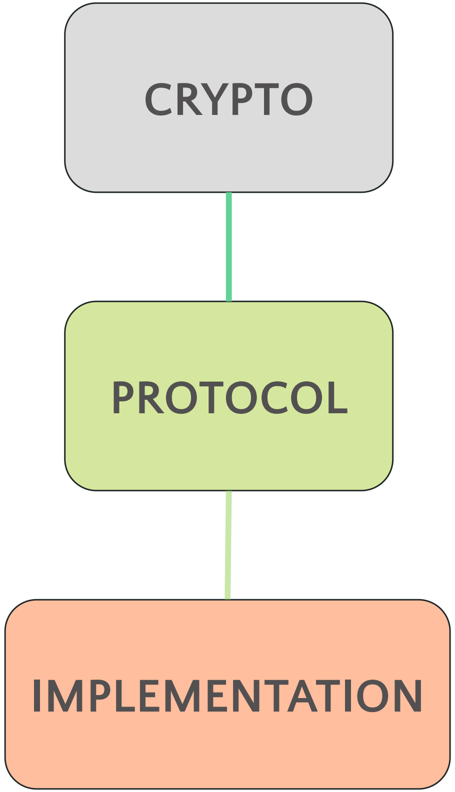
How can we apply these lessons to the PQC Migration?

Formal Methods for Post-Quantum Cryptography

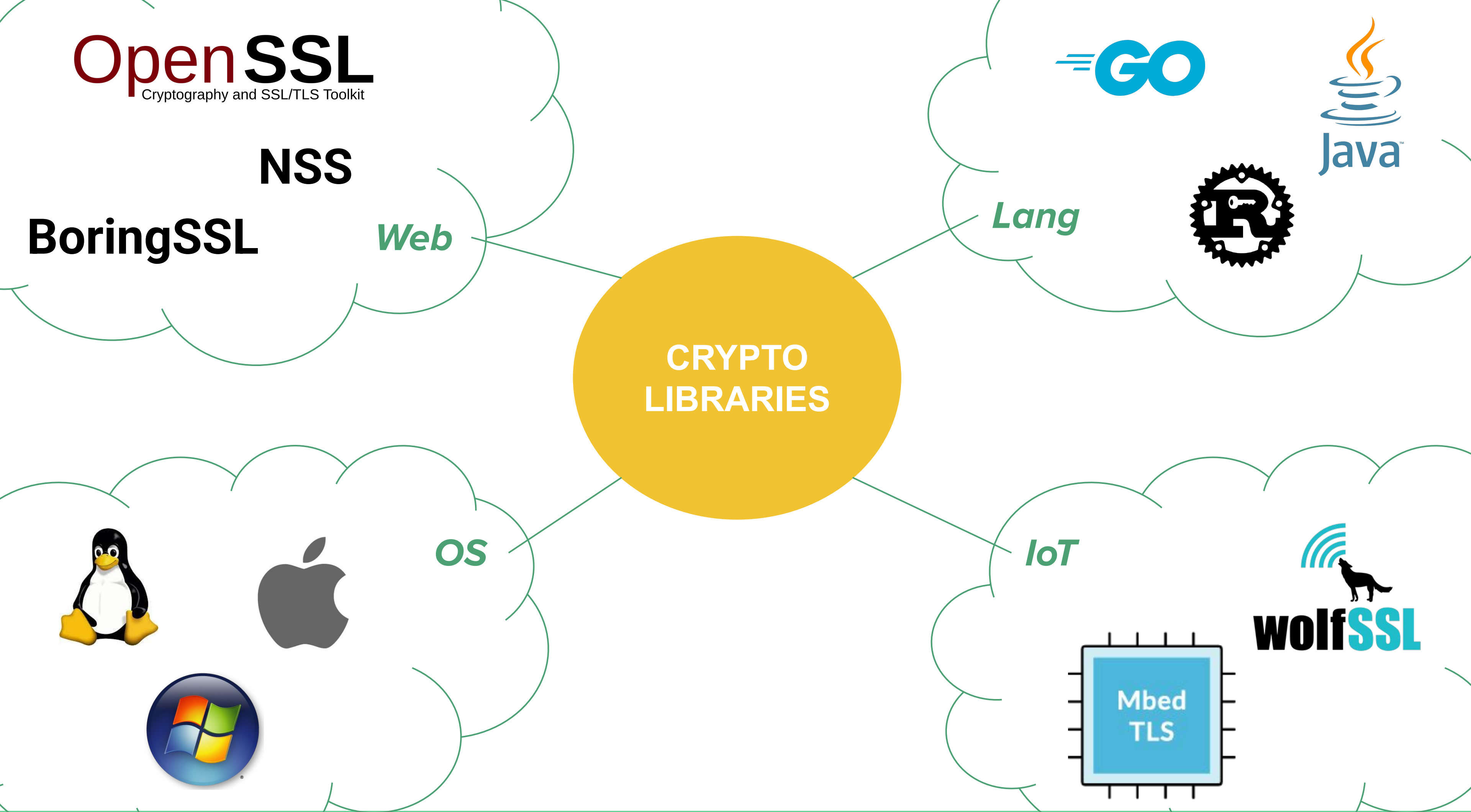
- Formally verifying PQC constructions and software
    - In this session: **Manuel Barbosa**

- Formally verifying real-world PQC protocols
    - In this session: **Rolfe Schmidt**

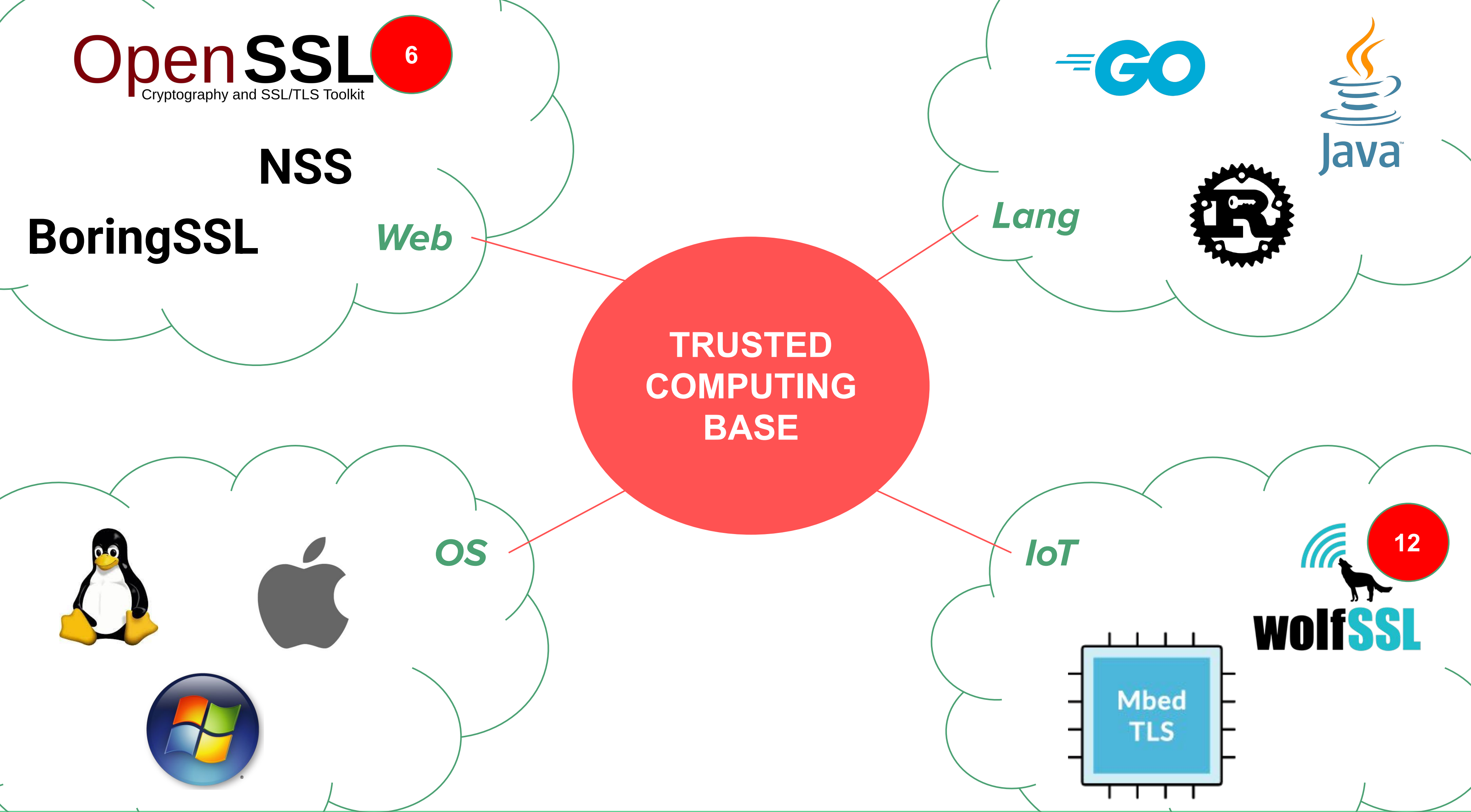
Rest of this talk: Verifying PQC software in Rust



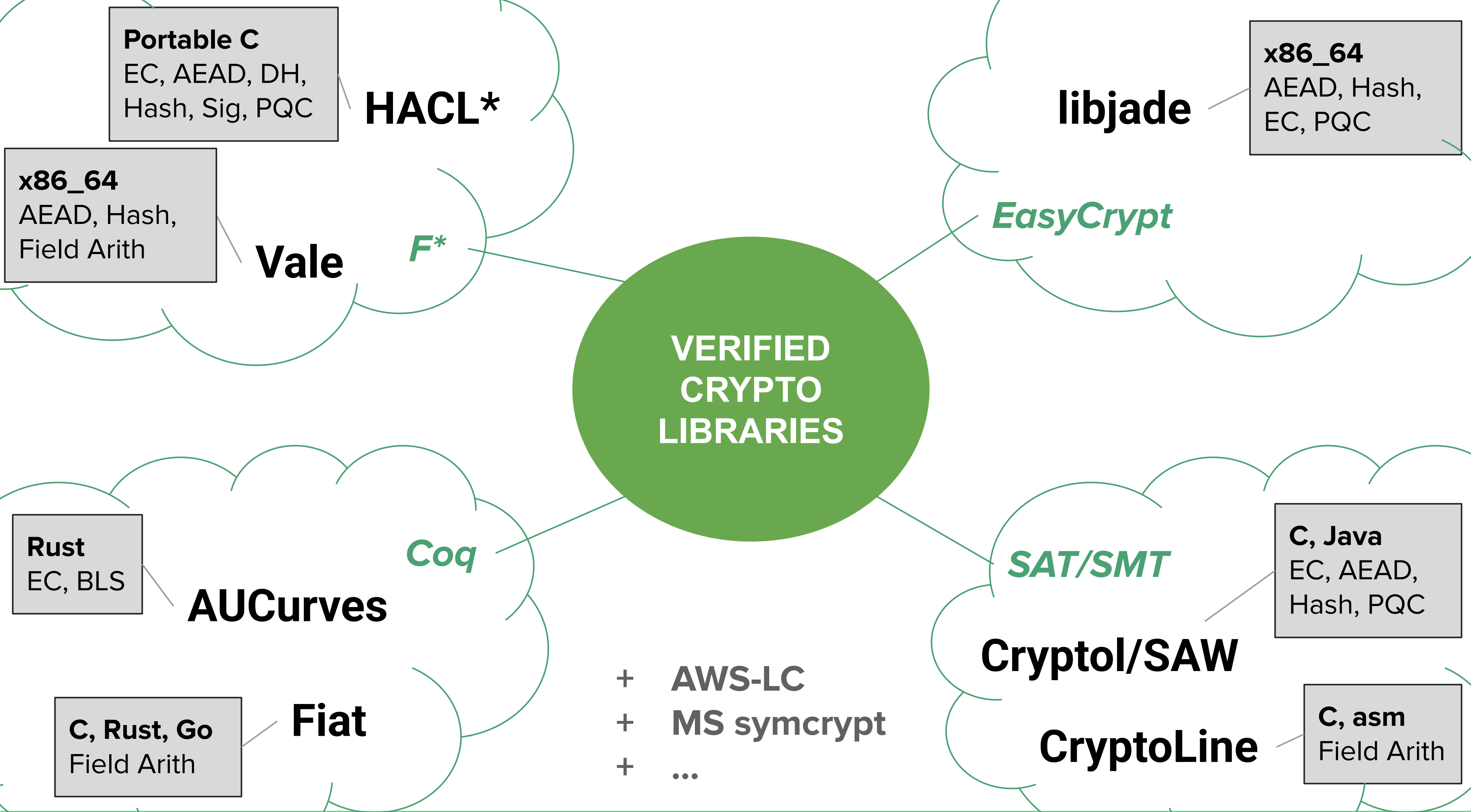
Verifying Post-Quantum Crypto Libraries



CRYPTO LIBRARIES
Web
OpenSSL
Cryptography and SSL/TLS Toolkit
NSS
BoringSSL
Lang
GO
Java
OS
IoT
wolfSSL
Mbed TLS



TRUSTED COMPUTING BASE
Web
OpenSSL
Cryptography and SSL/TLS Toolkit
6
NSS
BoringSSL
Lang
GO
Java
OS
IoT
12
wolfSSL
Mbed TLS



VERIFIED CRYPTO LIBRARIES
HACL*
Portable C
EC, AEAD, DH, Hash, Sig, PQC
Vale
x86_64
AEAD, Hash, Field Arith
F*
libjade
x86_64
AEAD, Hash, EC, PQC
EasyCrypt
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
+ AWS-LC
+ MS symcrypt
+ ...

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
- Formally verified crypto in NSS, BoringSSL, aws-lc, ...

FIPS 203

Federal Information Processing Standards Publication

Module-Lattice-Based Key-Encapsulation Mechanism Standard

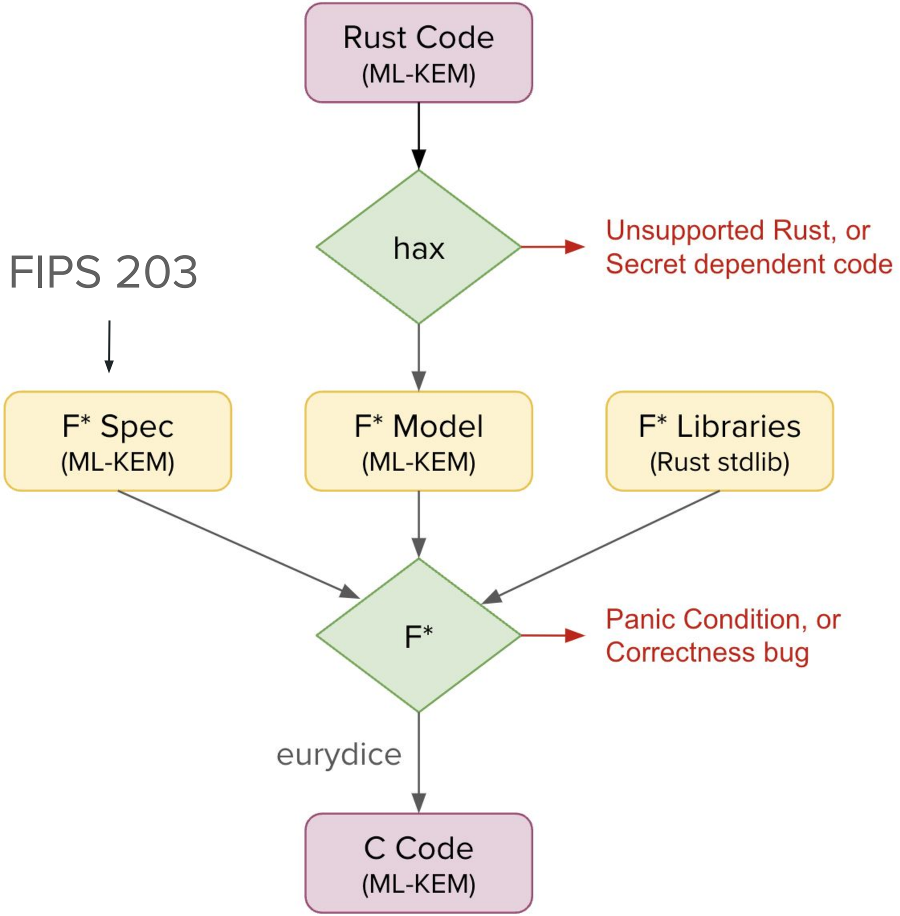
Category: Computer Security **Subcategory: Cryptography**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
https://doi.org/10.6028/NIST.FIPS.203

Published August 13, 2024

Verifying crypto code written in Rust using hax and F*



Writing Crypto Code in Rust

```rust
fn barrett_reduce(value: i32) -> i32 {
    let t = i64::from(value) * BARRETT_MULTIPLIER + (BARRETT_R >> 1);
    let quotient = (t >> BARRETT_SHIFT) as i32;
    let remainder = value - quotient * FIELD_MODULUS;
    remainder
}
```

Barrett Reduction Algorithm

(in constant time, so cannot directly use modulus operator)

Potential Panics in Rust Code

```rust
fn barrett_reduce(value: i32) -> i32 {
    let t = i64::from(value) * BARRETT_MULTIPLIER + (BARRETT_R >> 1);
    let quotient = (t >> BARRETT_SHIFT) as i32;
    let remainder = value - quotient * FIELD_MODULUS;
    remainder
}
```

These arithmetic operations may overflow or underflow causing the code to crash or compute wrong results

Proving Panic Freedom and Functional Correctness

```
#[requires(value >= -BARRETT_R && value <= BARRETT_R)]
#[ensures(|result| result > -FIELD_MODULUS && result < FIELD_MODULUS
                && result % FIELD_MODULUS == value % FIELD_MODULUS)]
fn barrett_reduce(value: i32) -> i32 {
    ...
    remainder
}
```

Specify pre-condition and post-condition
(Automatically verified using F*)

Enforcing Secret Independence

Type-based static analysis to avoid certain side channel attacks

- **arithmetic operations** with input-dependent timing (e.g. division) over secret integers
- **comparison** over secret values
- **branching** over secret values
- **array** or vector **accesses** at secret indices

Prevents timing bugs at Rust source level.

Does not prevent compiler-induced leaks, micro-architectural attacks, ...

KyberSlash: a new timing vulnerability

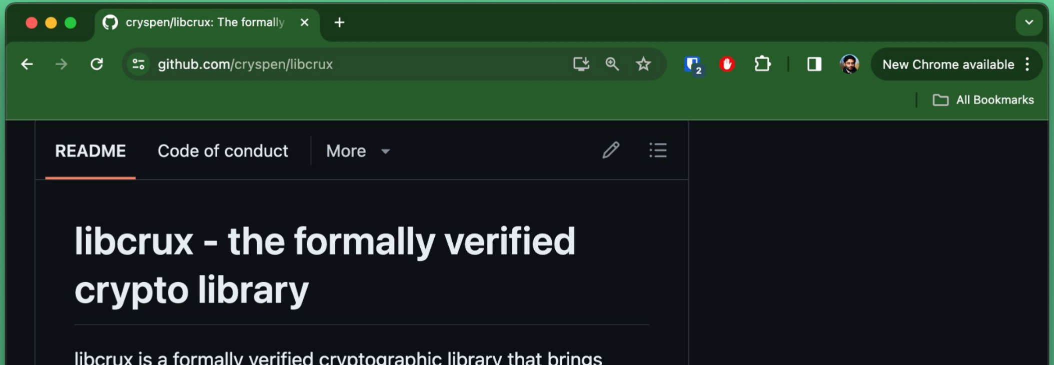
```c
void poly_tomsg(uint8_t msg[KYBER_INDCPA_MSGBYTES], const poly *a)
{
  unsigned int i,j;
  uint16_t t;

  for(i=0;i<KYBER_N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t  = a->coeffs[8*i+j];
      t += ((int16_t)t >> 15) & KYBER_Q;
      t  = (((t << 1) + KYBER_Q/2)/KYBER_Q) & 1;
      msg[i] |= t << j;
    }
  }
}
```

Bug present in PQ-Crystals, PQ-Clean, ...

Bug found during Formal Verification of our Rust code!

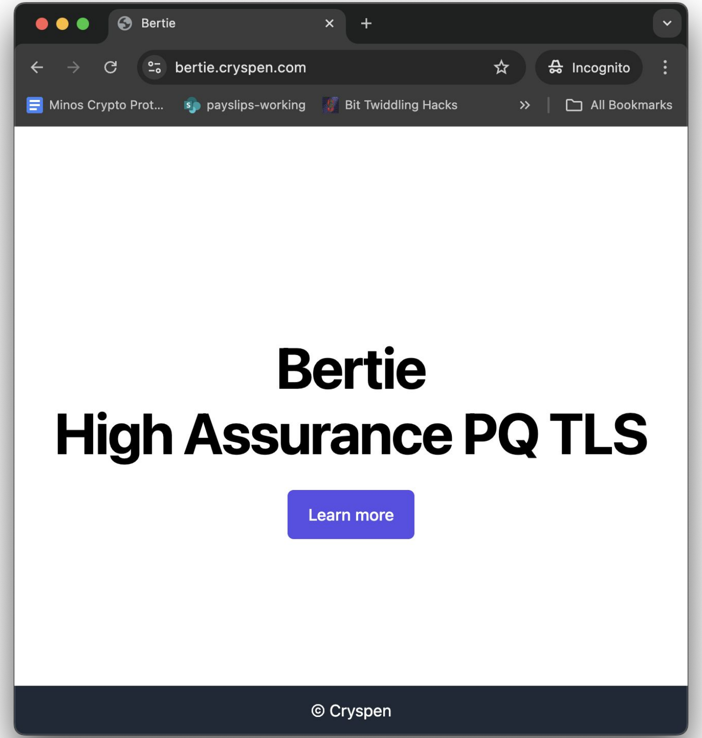
KyberSlash: Exploiting secret-dependent division timings in Kyber implementations. CHES 2025.
Bernstein, Bhargavan, Bhasin, Chattopadhyay, Chia, Kannwischer, Kiefer, Paiva, Ravi, Tamvada.

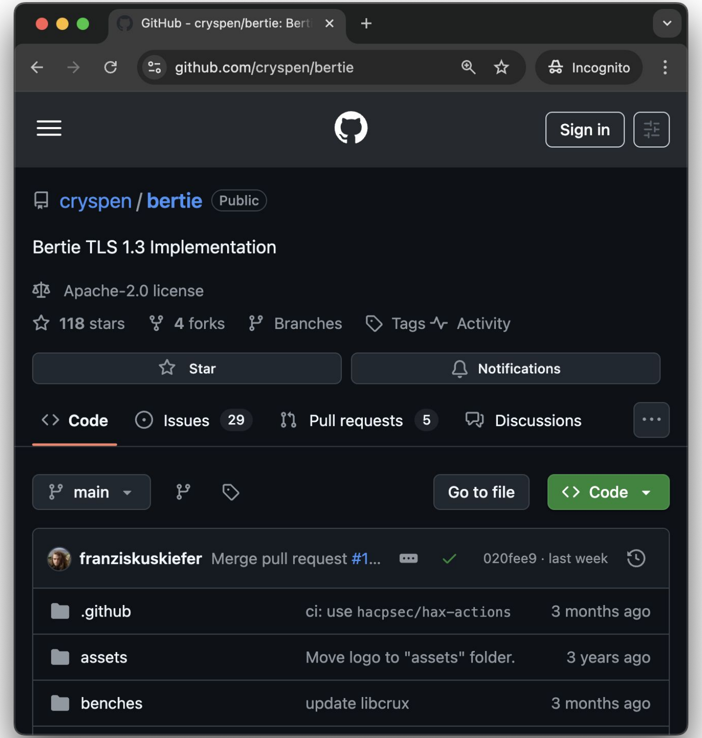
Libcrux has an **optimized**, **portable**, formally **verified** implementation of ML-KEM and ML-DSA in **Rust and C**. Our ML-KEM code is now deployed in Firefox, OpenSSH, Signal, ...



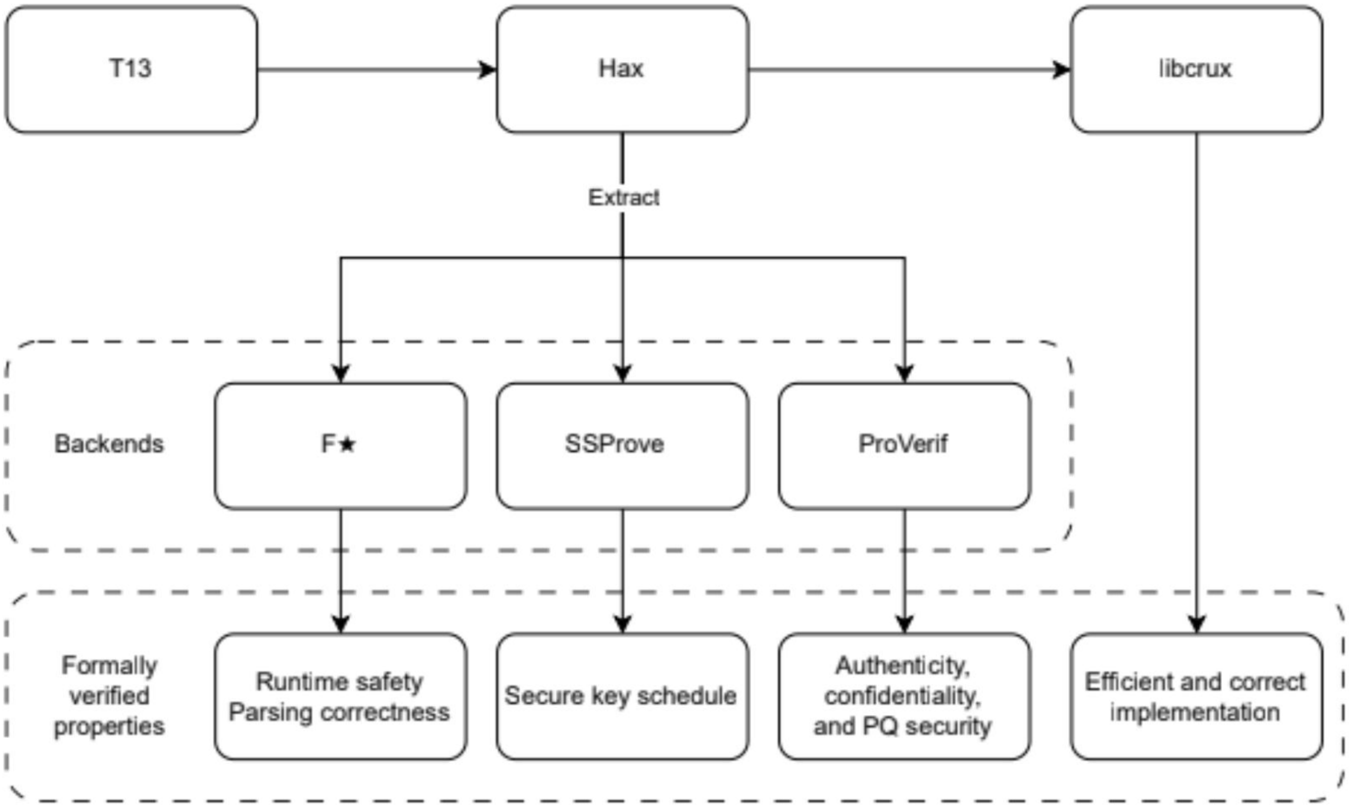
Verifying Post-Quantum Protocol Code

Bertie: Verified PQ-TLS 1.3 in Rust





Verifying Bertie for Security and Correctness



Verifying Bertie using hax

- We use the Rust type system to enforce state machine invariants
- We use libcrux for verified cryptography
- We use F* for properties like parsing correctness and panic-freedom
- We use ProVerif to prove protocol security for the TLS 1.3 implementation
- We use SSProve to cryptographically verify the key schedule implementation

Table 2: Formal Verification Results for Bert13

Backend Prover	Rust Modules	Rust LoC	Translated LoC	Properties Proven	Time Taken for Proofs (s)
SSProve	1	425	815	Core Key Schedule Security	11m17s
ProVerif	3	1723	5980	Forward Secrecy, Authentication	20s
				HNDL Post-Quantum Security	
F★	8	3264	10964	Runtime Safety, Unambiguous Formats	1m21s

Formal Security and Functional Verification of Cryptographic Protocol Implementations in Rust.
ACM CCS 2025. Bhargavan, Hansen, Kiefer, Schneider-Bensch, Spitters.

Conclusion

Transitioning to PQC is about more than just switching crypto algorithms

- New security models, new attacks, new software bugs
- We should **require machine-checked proofs** for all new protocols

Fast, verified post-quantum crypto is available and already deployed

- See Signal, Formosa, NSS, BoringSSL, AWS-LC, MS Symcrypt, …
- We should **require software verification** for all PQC libraries

PQC is a unique opportunity to upgrade the security of the Internet

- Provably improve security for long-lived hardware and software
- Leverage cutting-edge EU research on formal security verification

Questions?

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

- Recent Papers: https://cryspen.com/research/
- Code:
  - Hax: https://github.com/cryspen/hax
  - Libcrux: https://github.com/cryspen/libcrux
  - Bertie: https://github.com/cryspen/bertie
-

CRYSPEN

https://cryspen.com