Exploring post-quantum cryptography in Internet protocols

Douglas Stebila

UNIVERSITY OF WATERLOO

NSERC CRSNG

https://eprint.iacr.org/2019/858

https://eprint.iacr.org/2019/1356

https://eprint.iacr.org/2019/1447

https://tools.ietf.org/html/draft-stebila-tls-hybrid-design-01

https://openquantumsafe.org/

https://github.com/open-quantum-safe/

https://www.douglas.stebila.ca/

Univ. Grenoble Alpes • 2019-12-17

UNIVERSITY OF WATERLOO

IQC Institute for Quantum Computing

CYBER SECURITY AND PRIVACY INSTITUTE
UNIVERSITY OF WATERLOO

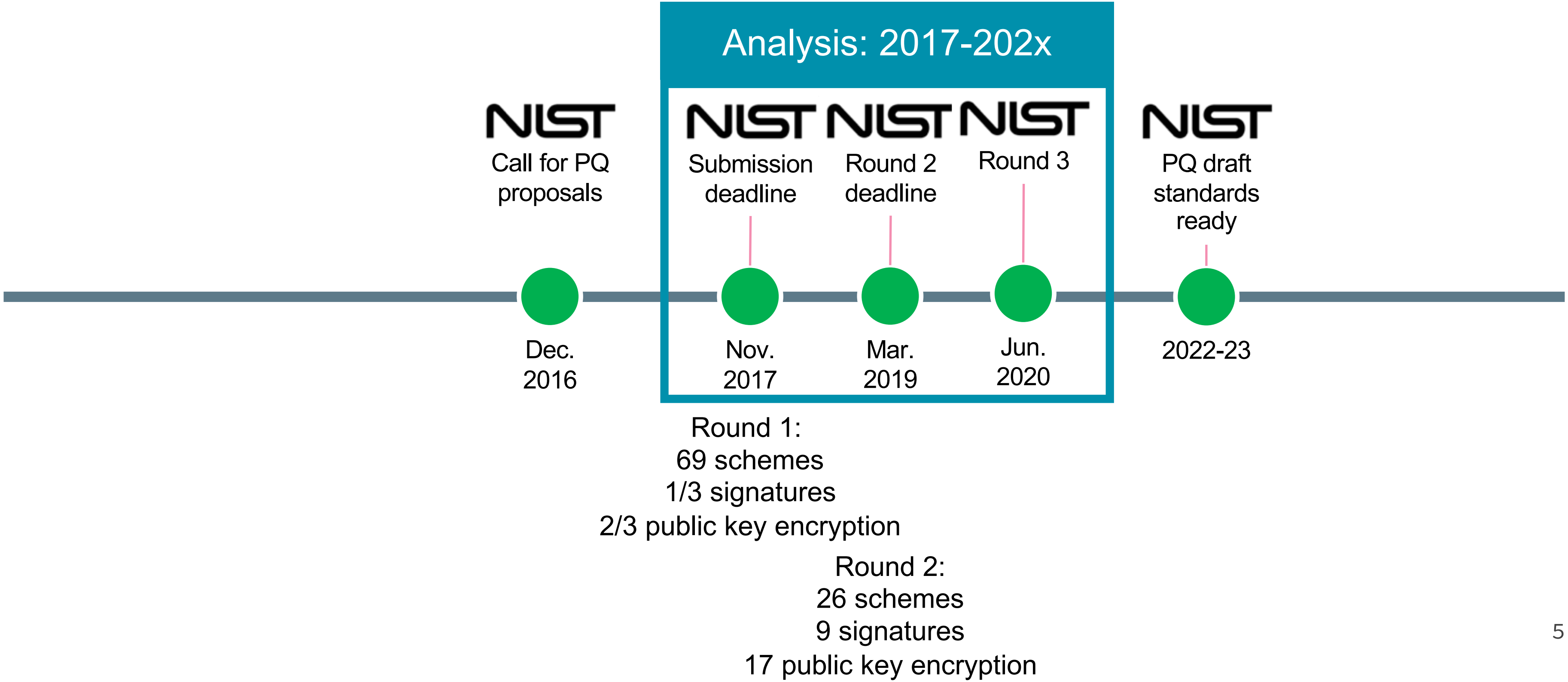
Quantum-resistant crypto @ Waterloo

- UW involved in 6 NIST Round 2 submissions:
  - CRYSTALS-Kyber, FrodoKEM, NewHope, NTRU, SIKE; qTESLA
- Large team led by David Jao working on isogeny-based crypto
- Quantum cryptanalysis led by Michele Mosca
- Quantum key distribution theory (Lütkenhaus) and experiments (Jennewein, Reimer)
- CryptoWorks21 training program for quantum-resistant cryptography

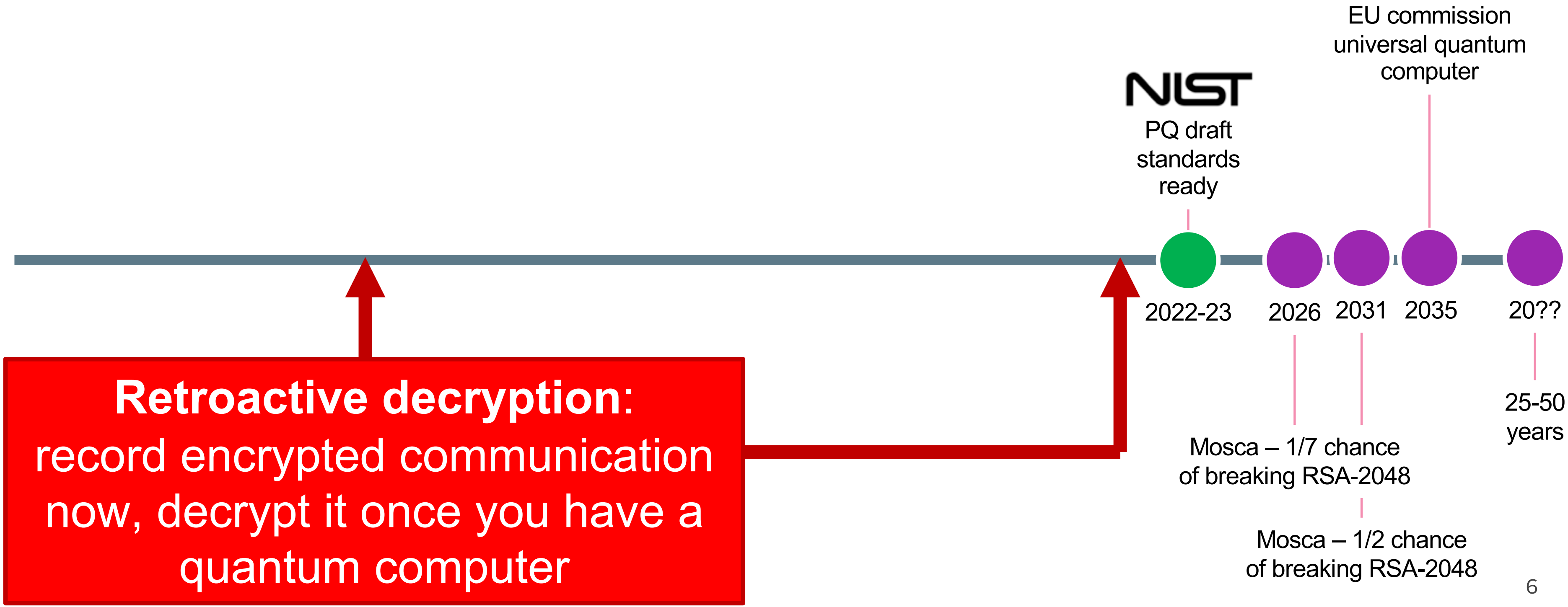
Motivating post-quantum cryptography

NIST Post-quantum Crypto Project timeline

http://www.nist.gov/pqcrypto

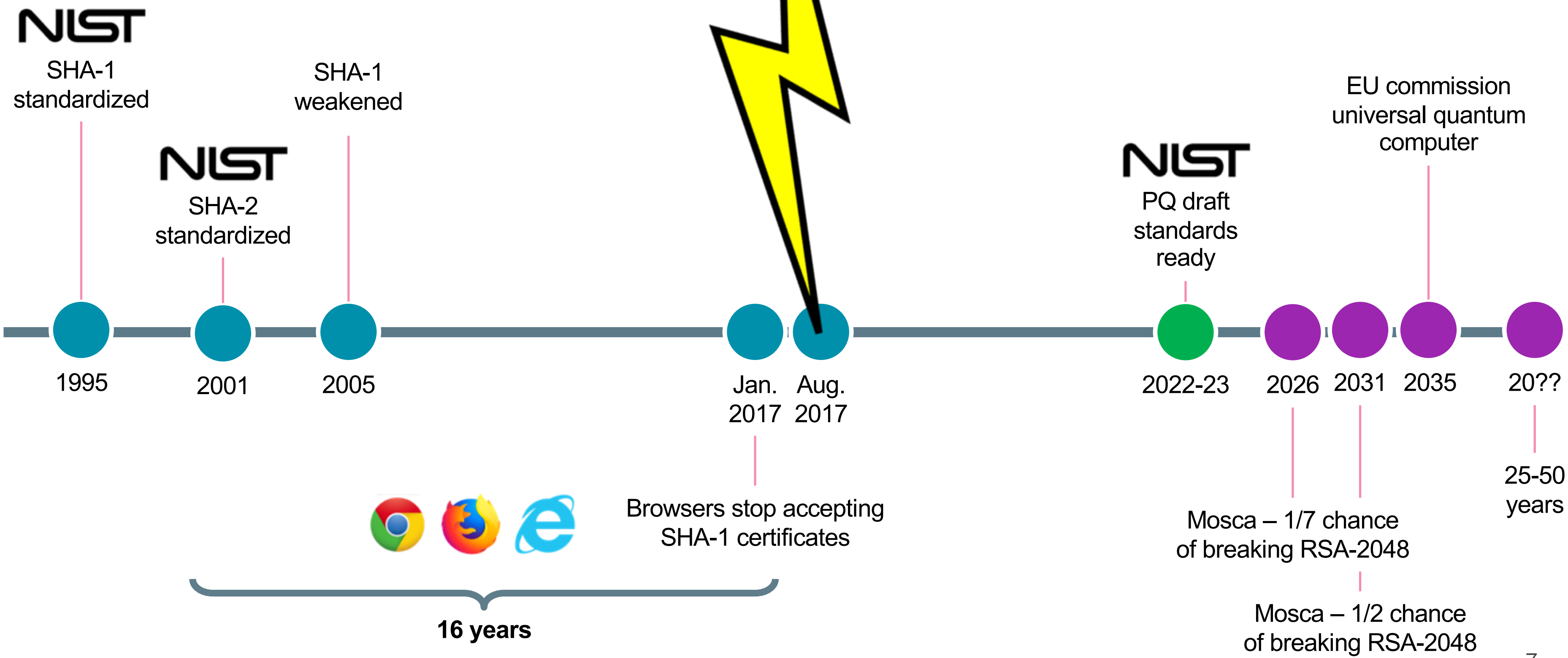


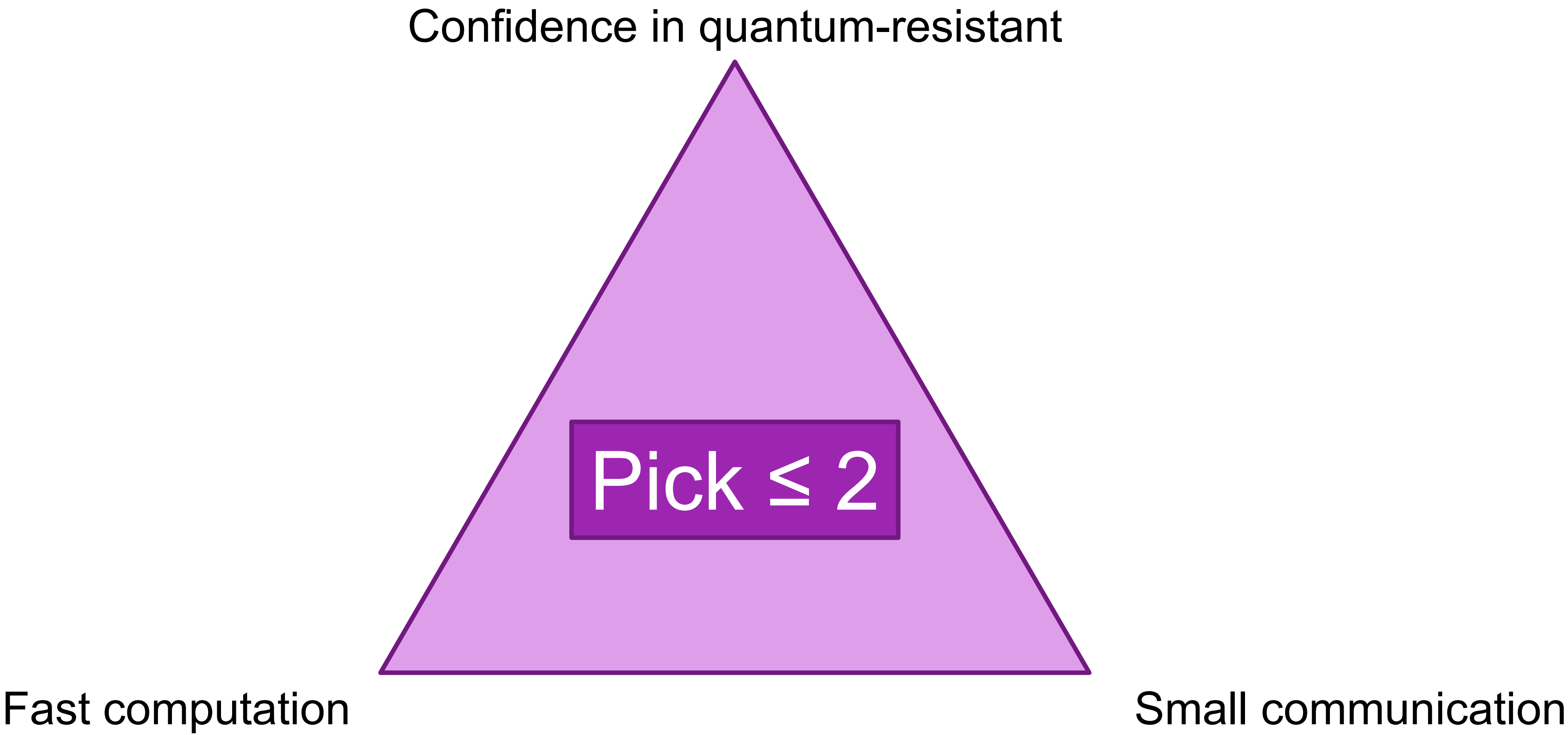
NIST Post-quantum Crypto Project timeline
http://www.nist.gov/pqcrypto



NIST Post-quantum Crypto Project timeline

http://www.nist.gov/pqcrypto




Confidence in quantum-resistant
Pick ≤ 2
Fast computation
Small communication

“Hybrid”

“Hybrid” or “composite” or “dual” or “multi-algorithm” cryptography

- Use pre-quantum and post-quantum algorithms together
- Secure if either one remains unbroken

Why hybrid?

- Potential post-quantum security for early adopters
- Maintain compliance with older standards (e.g. FIPS)
- Reduce risk from uncertainty on PQ assumptions/parameters

Hybrid ciphersuites

	Key exchange	Authentication
1	Hybrid traditional + PQ	Single traditional
2	Hybrid traditional + PQ	Hybrid traditional + PQ
3	Single PQ	Single traditional
4	Single PQ	Single PQ

Likely focus
for next 5-10 years

- Need PQ key exchange before we need PQ authentication because future quantum computers could retroactively decrypt, but not retroactively impersonate

Hybrid key exchange and authentication to date

- Hybrid key exchange Internet-Drafts at IETF:
  - TLS 1.2: Schanck, Whyte, Zhang 2016; Amazon 2019
  - TLS 1.3: Schanck, Stebila 2017; Whyte, Zhang, Fluhrer, Garcia-Morchon 2017; Kiefer, Kwiatkowski 2018; Stebila, Fluhrer, Gueron 2019
  - IPsec / IKEv2: Tjhai, Thomlinson, Bartlet, Fluhrer, Geest, Garcia-Morchon, Smyslov 2019
- Hybrid key exchange experimental implementations:
  - Google CECPQ1, CECPQ2; Open Quantum Safe; CECPQ2b; …
- Hybrid X.509 certificates:
  - Truskovsky, Van Geest, Fluhrer, Kampanakis, Ounsworth, Mister 2018

Design issues for hybrid key exchange in TLS 1.3

Douglas Stebila, Scott Fluhrer, Shay Gueron. **Design issues for hybrid key exchange in TLS 1.3**. **Internet-Draft**. Internet Engineering Task Force, July 2019. https://tools.ietf.org/html/draft-stebila-tls-hybrid-design-01

Goals for hybridization

1. Backwards compatibility
    - Hybrid-aware client, hybrid-aware server
    - Hybrid-aware client, non-hybrid-aware server
    - Non-hybrid-aware client, hybrid-aware server
2. Low computational overhead
3. Low latency
4. No extra round trips
5. No duplicate information

Design options

- How to negotiate algorithms
- How to convey cryptographic data (public keys / ciphertexts)
- How to combine keying material

Negotiation: How many algorithms?

2

≥ 2

Negotiation: How to indicate which algorithms to use

Negotiate each algorithm individually

1. Standardize a name for each algorithm
2. Provide a data structure for conveying supported algorithms
3. Implement logic negotiating which combination

Negotiate pre-defined combinations of algorithms

1. Standardize a name for each desired combination

- Can use existing negotiation data structures and logic

Which option is preferred may depend on how many algorithms are ultimately standardized.

Conveying cryptographic data (public keys / ciphertexts)

1) Separate public keys

- For each supported algorithm, send each public key / ciphertext in its own parseable data structure

2) Concatenate public keys

- For each supported combination, concatenate its public keys / ciphertext into an opaque data structure

#1 requires protocol and implementation changes

#2 abstracts combinations into "just another single algorithm"

But #2 can also lead to sending duplicate values

- nistp256+bike1l1
- nistp256+sikep403
- nistp256+frodo640aes
- sikep403+frodo640aes

3x nistp256,
2x sikep403,
2x frodo640aes
public keys

Combining keying material

Top requirement: needs to provide "robust" security:

- Final session key should be secure as long as at least one of the ingredient keys is unbroken
- (Most obvious techniques are fine, though with some subtleties; see Giacon, Heuer, Poettering PKC'18, Bindel et al. PQCrypto 2019, … .)

- XOR keys
- Concatenate keys and use directly
- Concatenate keys then apply a hash function / KDF
- Extend the protocol's "key schedule" with new stages for each key
- Insert the 2nd key into an unused spot in the protocol's key schedule

Draft-00 @ IETF 104

draft-stebila-tls-hybrid-design-00

Contained a “menu” of design options along several axes

1. How to negotiate which algorithms?
2. How many algorithms?
3. How to transmit public key shares?
4. How to combine secrets?

Feedback from working group:

- Avoid changes to key schedule
- Present one or two instantiations
- Specific feedback on some aspects

Draft-01 @ IETF 105

draft-stebila-tls-hybrid-design-01

Kept menu of design choices

Constructed two candidate instantiations from menu for discussion

1. Directly negotiate each hybrid algorithm; separate key shares
2. Code points for pre-defined combinations; concatenated key shares

Additional KDF-based options for combining keys

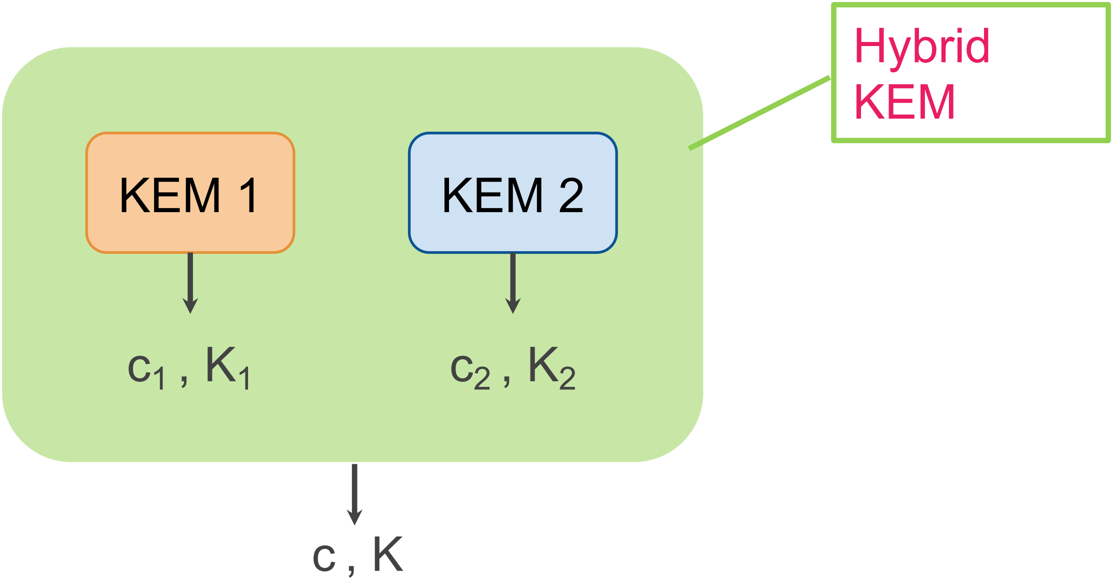
Emerging consensus?

- **Combining keying material**:
  - Consensus: (unambiguously) concatenate keys then apply hash function / KDF
- **Number of algorithms**: 2 vs ≥ 2:
  - TLS working group leaning to 2
- **Negotiation**: negotiate algorithms separately versus in combination:
  - All(?) implementations to date have negotiated pre-defined combinations
  - TLS working group leaning to “in combination”
- **Conveying public keys**: separately versus concatenated:
  - All(?) implementations to date have used concatenation
  - TLS working group leaning to (unambiguous) concatenation

Hybrid key encapsulation mechanisms and authenticated key exchange

Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, Douglas Stebila. **Hybrid key encapsulation mechanisms and authenticated key exchange**. In Jintai Ding, Rainer Steinwandt, editors, *Proc. 10th International Conference on Post-Quantum Cryptography (PQCrypto) 2019*, *LNCS*. Springer, May 2019. https://eprint.iacr.org/2019/858

Safely combining KEMs



- How to safely combine into single KEM such that this hybrid preserves security, as long as one of the two input schemes remains secure

Existing options

- XOR
  - K = K1 XOR K2
  - Preserves IND-CPA security but not IND-CCA security (mix and match attack)
- XOR with transcript (Giacon et al. PKC 2018)
  - K = H(K1 XOR K2, C1 || C2)
  - Preserves IND-CCA security if H is a random oracle
- Concatenation (Giacon et al. PKC 2018)
  - K = H(K1 || K2, C1 || C2)
  - Preserves IND-CCA security if H is a random oracle

The XOR-then-MAC Combiner

- Add MAC $\tau = \text{MAC}(c)$

$$K \,||\, K_{MAC} \leftarrow K_1 \text{ XOR } K_2$$

$$c = (c_1, c_2, \tau)$$

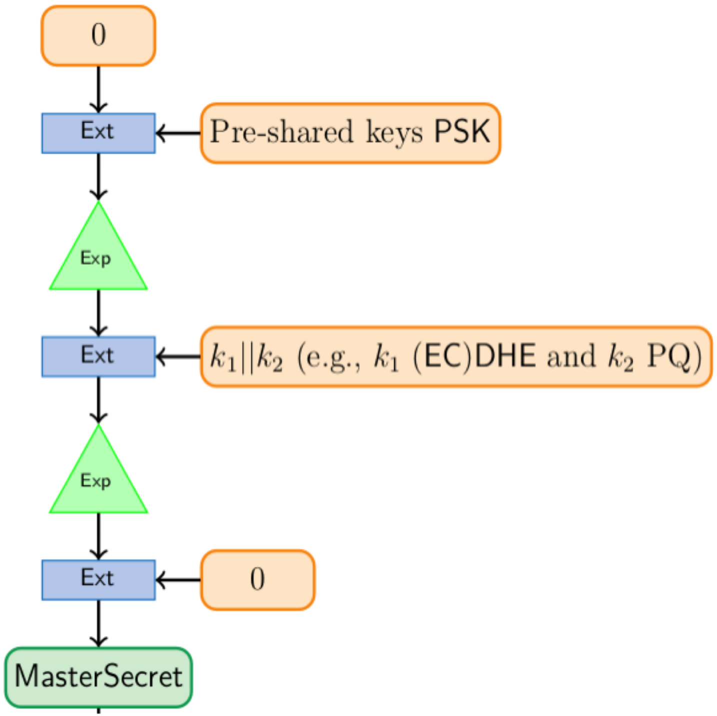
- Preserves IND-CCA security under the **standard model** assumption that MAC is secure
- Protocols (e.g. TLS) often compute MAC over transcript anyways (may replace the MAC here)

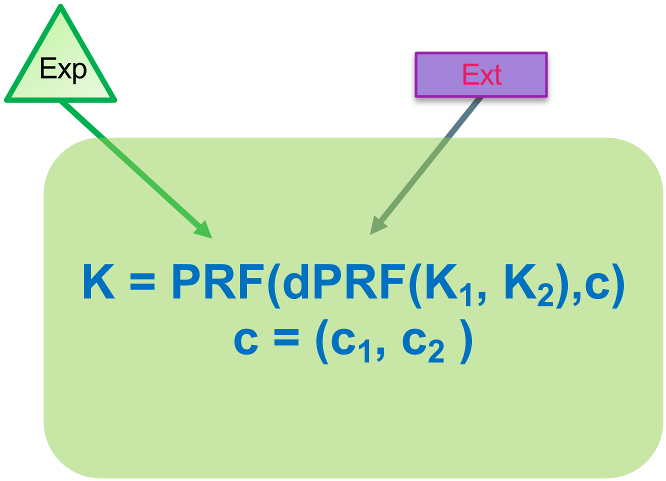
dualPRF Combiner

- **dualPRF Security**: both dPRF(k,·) and dPRF(·,x) are pseudorandom functions
- Models concatenation-based TLS 1.3 hybrid drafts
- HKDF is a dual PRF

$$K = PRF(dPRF(K_1, K_2), c)$$
$$c = (c_1, c_2)$$

dualPRF Combiner



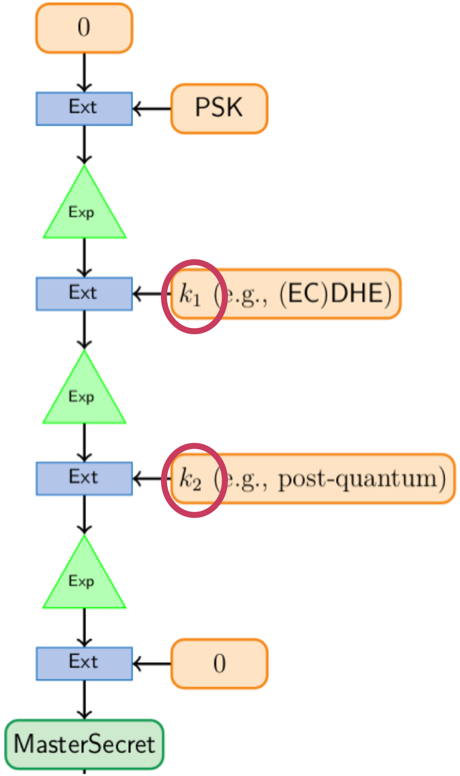


Nested dualPRF Combiner

- dualPRF combiner with additional preprocessing step
- Inspired by the TLS 1.3 key schedule
  - Models TLS 1.3 hybrid draft by Schanck and Stebila

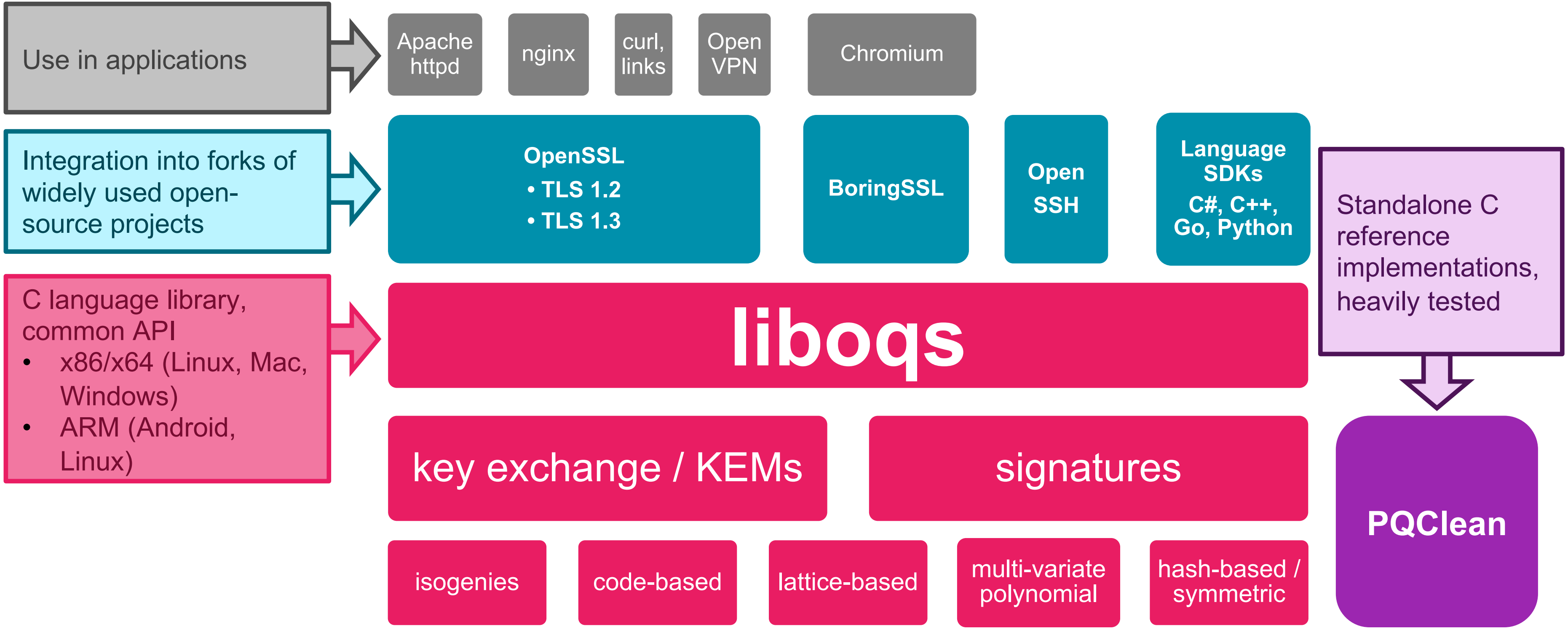
$$K_e = Ext(0, K_1)$$

$$K = PRF(dPRF(K_e, K_2),c)$$



https://openquantumsafe.org/ • https://github.com/open-quantum-safe/

Open Quantum Safe Project



OQS team

- Project leads
  - Douglas Stebila (Waterloo)
  - Michele Mosca (Waterloo)
- Industry collaborators
  - Amazon Web Services
  - Cisco Systems
  - evolutionQ
  - IBM Research
  - Microsoft Research
- Individual contributors

- Financial support
  - Government of Canada
    - NSERC Discoverry
    - Tutte Institute
  - Amazon Web Services

- In-kind contributions of developer time from industry collaborators

liboqs

- C library with common API for post-quantum signature schemes and key encapsulation mechanisms
- MIT License
- Builds on Windows, macOS, Linux; x86_64, ARM v8
- 43 key encapsulation mechanisms from 7 NIST Round 2 candidates
- 52 signature schemes from 5 NIST Round 2 candidates

List of algorithms

Key encapsulation mechanisms

- **BIKE**: BIKE1-L1-CPA, BIKE1-L3-CPA, BIKE1-L1-FO, BIKE1-L3-FO
- **FrodoKEM**: FrodoKEM-640-AES, FrodoKEM-640-SHAKE, FrodoKEM-976-AES, FrodoKEM-976-SHAKE, FrodoKEM-1344-AES, FrodoKEM-1344-SHAKE
- **Kyber**: Kyber512, Kyber768, Kyber1024, Kyber512-90s, Kyber768-90s, Kyber1024-90s
- **NewHope**: NewHope-512-CCA, NewHope-1024-CCA
- **NTRU**: NTRU-HPS-2048-509, NTRU-HPS-2048-677, NTRU-HPS-4096-821, NTRU-HRSS-701
- **SABER**: LightSaber-KEM, Saber-KEM, FireSaber-KEM
- **SIKE**: SIDH-p434, SIDH-p503, SIDH-p610, SIDH-p751, SIKE-p434, SIKE-p503, SIKE-p610, SIKE-p751, SIDH-p434-compressed, SIDH-p503-compressed, SIDH-p610-compressed, SIDH-p751-compressed, SIKE-p434-compressed, SIKE-p503-compressed, SIKE-p610-compressed, SIKE-p751-compressed

Signature schemes

- **Dilithium**: Dilithium2, Dilithium3, Dilithium4
- **MQDSS**: MQDSS-31-48, MQDSS-31-64
- **Picnic**: Picnic-L1-FS, Picnic-L1-UR, Picnic-L3-FS, Picnic-L3-UR, Picnic-L5-FS, Picnic-L5-UR, Picnic2-L1-FS, Picnic2-L3-FS, Picnic2-L5-FS
- **qTesla**: qTesla-p-I, qTesla-p-III
- **SPHINCS+-Haraka**: SPHINCS+-Haraka-128f-robust, SPHINCS+-Haraka-128f-simple, SPHINCS+-Haraka-128s-robust, SPHINCS+-Haraka-128s-simple, SPHINCS+-Haraka-192f-robust, SPHINCS+-Haraka-192f-simple, SPHINCS+-Haraka-192s-robust, SPHINCS+-Haraka-192s-simple, SPHINCS+-Haraka-256f-robust, SPHINCS+-Haraka-256f-simple, SPHINCS+-Haraka-256s-robust, SPHINCS+-Haraka-256s-simple
- **SPHINCS+-SHA256**: SPHINCS+-SHA256-128f-robust, SPHINCS+-SHA256-128f-simple, SPHINCS+-SHA256-128s-robust, SPHINCS+-SHA256-128s-simple, SPHINCS+-SHA256-192f-robust, SPHINCS+-SHA256-192f-simple, SPHINCS+-SHA256-192s-robust, SPHINCS+-SHA256-192s-simple, SPHINCS+-SHA256-256f-robust, SPHINCS+-SHA256-256f-simple, SPHINCS+-SHA256-256s-robust, SPHINCS+-SHA256-256s-simple
- **SPHINCS+-SHAKE256**: SPHINCS+-SHAKE256-128f-robust, SPHINCS+-SHAKE256-128f-simple, SPHINCS+-SHAKE256-128s-robust, SPHINCS+-SHAKE256-128s-simple, SPHINCS+-SHAKE256-192f-robust, SPHINCS+-SHAKE256-192f-simple, SPHINCS+-SHAKE256-192s-robust, SPHINCS+-SHAKE256-192s-simple, SPHINCS+-SHAKE256-256f-robust, SPHINCS+-SHAKE256-256f-simple, SPHINCS+-SHAKE256-256s-robust, SPHINCS+-SHAKE256-256s-simple

PQClean

- New, sister project to OQS
- Goal: standalone, high-quality C reference implementations of PQ algorithms
  - Lots of automated code analysis and continuous integration testing
  - Builds tested on little-endian and big-endian
- MIT License and public domain
- Not a library, but easy to pull out code that can be incorporated into a library
  - liboqs consumes implementations from PQClean
- In collaboration with Peter Schwabe and team at Radboud University, Netherlands

https://github.com/PQClean/PQClean

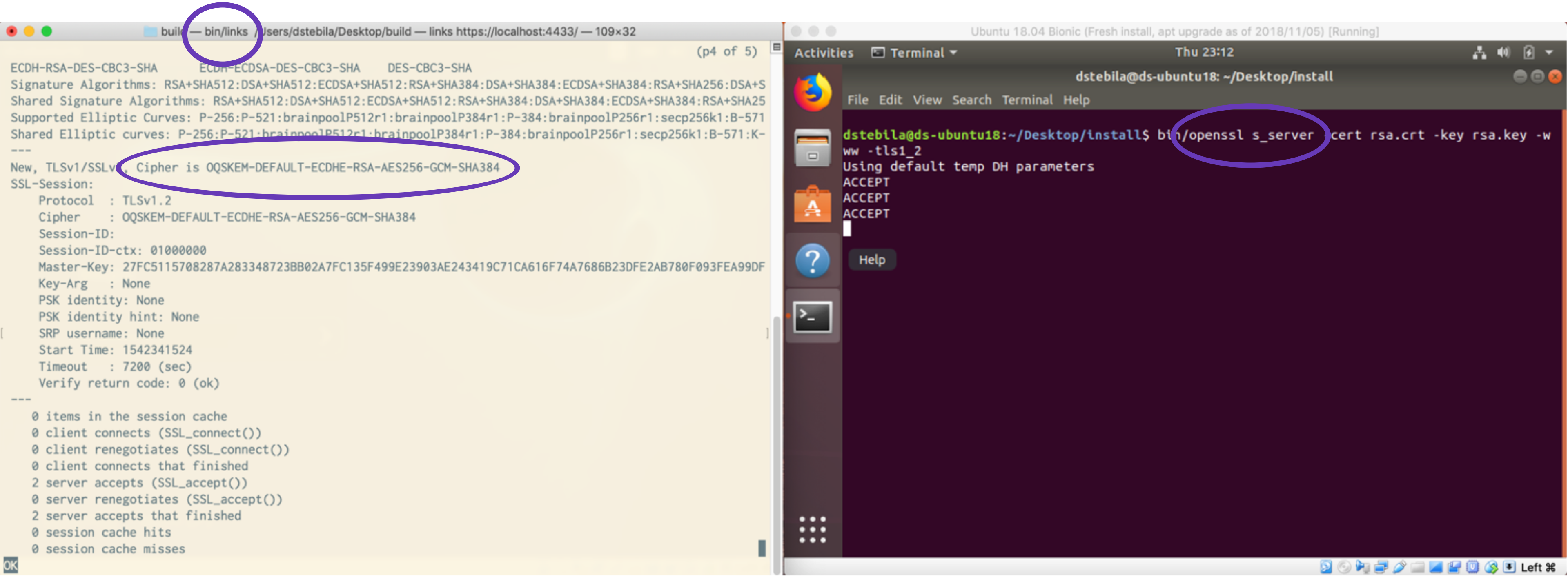
OpenSSL

- OQS fork of OpenSSL 1.0.2
  - PQ and hybrid key exchange in TLS 1.2

- OQS fork of OpenSSL 1.1.1
  - PQ and hybrid key exchange in TLS 1.3
  - PQ and hybrid certificates and signature authentication in TLS 1.3

- Can be readily used with applications that rely on OpenSSL with few/no modifications

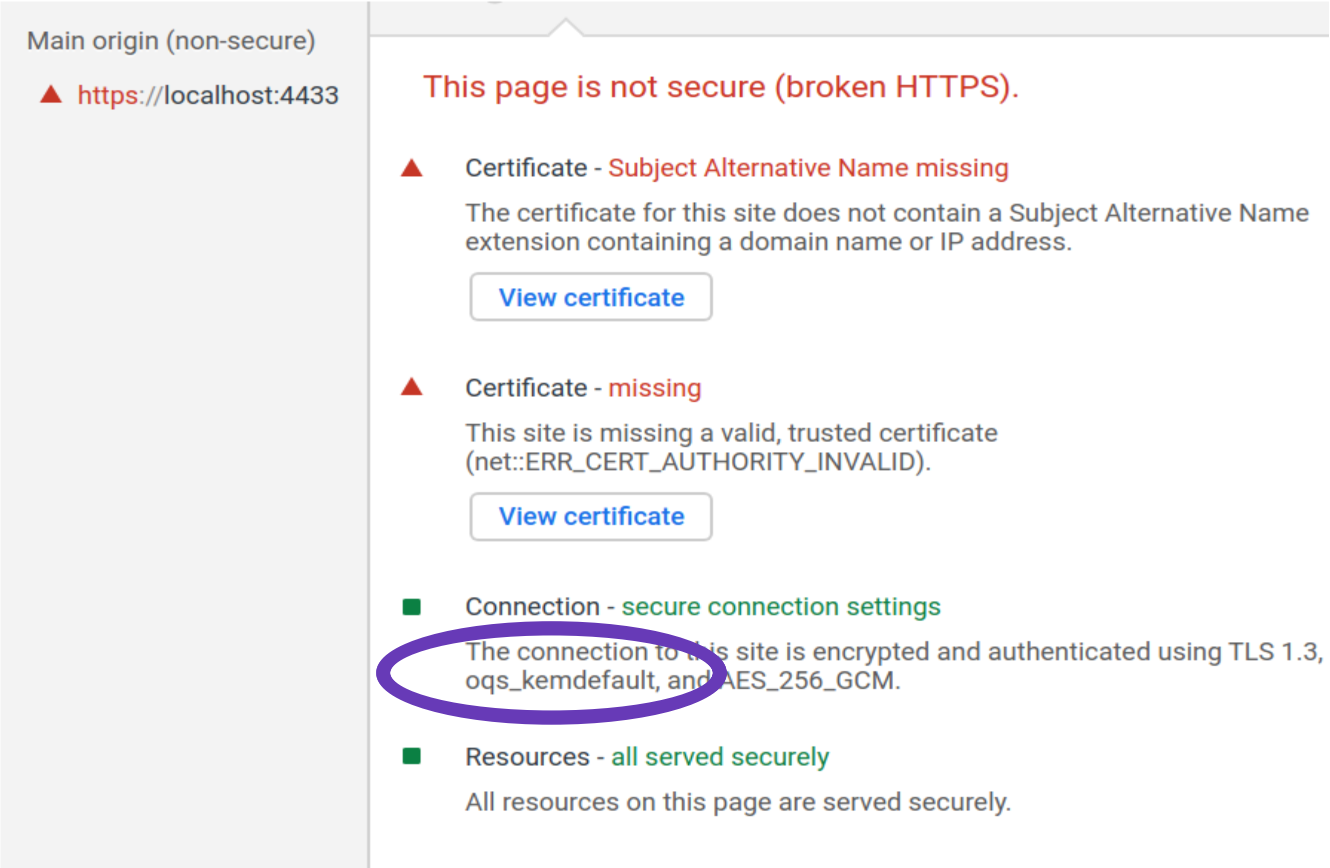
OQS demo: OpenSSL



BoringSSL

- OQS fork of BoringSSL (which is a fork of OpenSSL)
  - PQ and hybrid key exchange in TLS 1.3
- After a few modifications, can be used with Chromium!

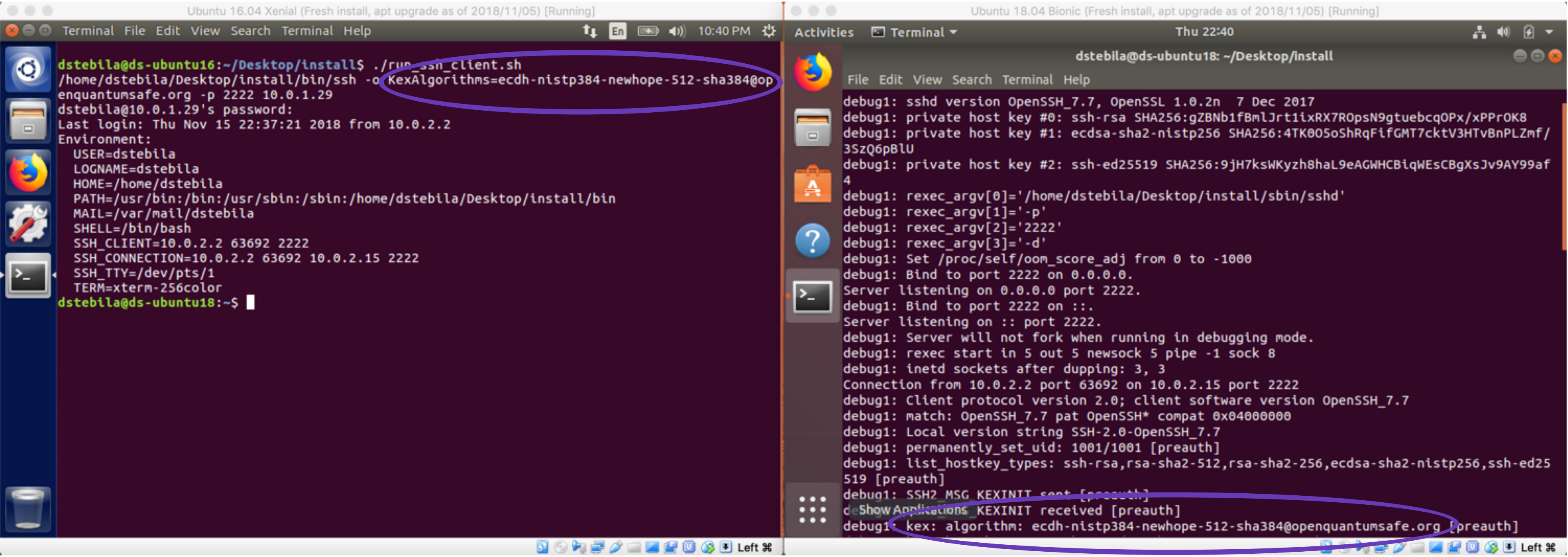
OQS demo: Chromium with BoringSSL talking to Apache



OpenSSH

- OQS fork of OpenSSH
  - PQ and hybrid key exchange
  - PQ and hybrid signature authentication

OQS demo: OpenSSH



Using OQS

- All open source software available on GitHub
- Instructions for building on Linux, macOS, and Windows
- Docker images available for building and running OQS-reliant applications
  - Apache httpd
  - curl
  - nginx
  - OpenSSH

Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH

Eric Crockett, Christian Paquin, Douglas Stebila. **Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH**. In *NIST 2nd Post-Quantum Cryptography Standardization Conference 2019*. August 2019. https://eprint.iacr.org/2019/858

Case study 1: TLS 1.2 in Amazon s2n

- Multi-level negotiation following TLS 1.2 design style:
  - Top-level ciphersuite with algorithm family: e.g. `TLS_ECDHE_SIKE_ECDSA_WITH_AES_256_GCM_SHA384`
  - Extensions used to negotiate parameterization within family:
    - 1 extension for which ECDH elliptic curve: nistp256, curve25519, …
    - 1 extension for which PQ parameterization: sikep403, sikep504, …
- Session key: concatenate session keys and apply KDF with public key/ciphertext as KDF label
- Experimental results: successfully implemented using nistp256+{bike1l1, sikep503}

Case studies 2, 3, 4:
TLS 1.2 in OpenSSL 1.0.2
TLS 1.3 in OpenSSL 1.1.1
SSH v2 in OpenSSH 7.9

- Negotiate pairs of algorithms in pre-defined combinations
- Session key: concatenate session keys and use directly in key schedule

- Easy implementation, no change to negotiation logic

- Based on implementations in liboqs
  - KEMs: 9 of 17 (BIKE round 1, FrodoKEM, Kyber, LEDAcrypt, NewHope, NTRU, NTS (1 variant), Saber, SIKE)
  - Signature schemes: 6 of 9 (Dilithium, MQDSS, Picnic, qTesla (round 1), Rainbow, SPHINCS+)

1st circle: PQ only
2nd circle: hybrid ECDH

● = success

◓ = fixable by changing implementation parameter

○ = would violate spec or otherwise unresolved error

† = algorithm on testing branch

	s2n (TLS 1.2)	OpenSSL 1.0.2 (TLS 1.2)	OpenSSL 1.1.1 (TLS 1.3)	OpenSSH
BIKE1-L1 (round 1)	– ●	●●	●●	●●
BIKE1-L3 (round 1)	– –	●●	●●	●●
BIKE1-L5 (round 1)	– –	●●	●●	●●
BIKE2-L1 (round 1)	– –	●●	●●	●●
BIKE2-L3 (round 1)	– –	●●	●●	●●
BIKE2-L5 (round 1)	– –	●●	●●	●●
BIKE3-L1 (round 1)	– –	●●	●●	●●
BIKE3-L3 (round 1)	– –	●●	●●	●●
BIKE3-L5 (round 1)	– –	●●	●●	●●
FrodoKEM-640-AES	– –	●●	●●	●●
FrodoKEM-640-SHAKE	– –	●●	●●	●●
FrodoKEM-976-AES	– –	●●	●●	●●
FrodoKEM-976-SHAKE	– –	●●	●●	●●
FrodoKEM-1344-AES	– –	◓◓	◓◓	●●
FrodoKEM-1344-SHAKE	– –	◓◓	◓◓	●●
Kyber512	– –	●●	●●	●●
Kyber768	– –	●●	●●	●●
Kyber1024	– –	●●	●●	●●
LEDAcrypt-KEM-LT-12†	– –	●●	●●	●●
LEDAcrypt-KEM-LT-32†	– –	●●	●●	●●
LEDAcrypt-KEM-LT-52†	– –	●●	●●	●●
NewHope-512-CCA	– –	●●	●●	●●
NewHope-1024-CCA	– –	●●	●●	●●
NTRU-HPS-2048-509	– –	●●	●●	●●
NTRU-HPS-2048-677	– –	●●	●●	●●
NTRU-HPS-4096-821	– –	●●	●●	●●
NTRU-HRSS-701	– –	●●	●●	●●
NTS-KEM(12,64)†	– –	○○	○○	○○
LightSaber-KEM	– –	●●	●●	●●
Saber-KEM	– –	●●	●●	●●
FireSaber-KEM	– –	●●	●●	●●
SIKEp503 (round 1)	– ●	– –	– –	– –
SIKEp434	– –	●●	●●	●●
SIKEp503	– –	●●	●●	●●
SIKEp610	– –	●●	●●	●●
SIKEp751	– –	●●	●●	●●

FrodoKEM 976, 1344

- OpenSSL 1.0.2 / TLS 1.2: too large for a pre-programmed buffer size, but easily fixed by increasing one buffer size
- OpenSSL 1.1.1 / TLS 1.3: same

NTS-KEM

- OpenSSL 1.0.2 / TLS 1.2: theoretically within spec's limitation of 2^{24} bytes, but buffer sizes that large caused failures we couldn't track down
- OpenSSL 1.1.1 / TLS 1.3: too large for spec (2^{16}-1 bytes)
- OpenSSH: theoretically within spec but not within RFC's "SHOULD", but couldn't resolve bugs

1st circle: PQ only
2nd circle: hybrid RSA

● = success

◓ = fixable by changing implementation parameter

○ = would violate spec or otherwise unresolved error

† = algorithm on testing branch

	OpenSSL 1.1.1 (TLS 1.3)
Dilithium-2	● ●
Dilithium-3	● ●
Dilithium-4	● ●
MQDSS-31-48	◓ ◓
MQDSS-31-64	◓ ◓
Picnic-L1-FS	◓ ◓
Picnic-L1-UR	◓ ◓
Picnic-L3-FS	○ ○
Picnic-L3-UR	○ ○
Picnic-L5-FS	○ ○
Picnic-L5-UR	○ ○
Picnic2-L1-FS	● ●
Picnic2-L3-FS	◓ ◓
Picnic2-L5-FS	◓ ◓
qTesla-I (round 1)	● ●
qTesla-III-size (round 1)	● ●
qTesla-III-speed (round 1)	● ●
Rainbow-Ia-Classic†	◓ ◓
Rainbow-Ia-Cyclic†	● ●
Rainbow-Ia-Cyclic-Compressed†	● ●
Rainbow-IIIc-Classic†	◓ ◓
Rainbow-IIIc-Cyclic†	◓ ◓
Rainbow-IIIc-Cyclic-Compressed†	◓ ◓
Rainbow-Vc-Classic†	◓ ◓
Rainbow-Vc-Cyclic†	◓ ◓
Rainbow-Vc-Cyclic-Compressed†	◓ ◓
SPHINCS+-{Haraka,SHA256,SHAKE256}-128f-{robust,simple}	◓ ◓
SPHINCS+-{Haraka,SHA256,SHAKE256}-128s-{robust,simple}	● ●
SPHINCS+-{Haraka,SHA256,SHAKE256}-192f-{robust,simple}	◓ ◓
SPHINCS+-{Haraka,SHA256,SHAKE256}-192s-{robust,simple}	◓ ◓
SPHINCS+-{Haraka,SHA256,SHAKE256}-256f-{robust,simple}	◓ ◓
SPHINCS+-{Haraka,SHA256,SHAKE256}-256s-{robust,simple}	◓ ◓

TLS 1.3:

- Max certificate size: 2^{24}-1
- Max signature size: 2^{16}-1

OpenSSL 1.1.1:

- Max certificate size: 102,400 bytes, but runtime enlargeable
- Max signature size: 2^{14}

1st circle: PQ only
2nd circle: hybrid RSA

● = success

◓ = fixable by changing implementation parameter

○ = would violate spec or otherwise unresolved error

† = algorithm on testing branch

	OpenSSL 1.1.1 (TLS 1.3)	OpenSSH
Dilithium-2	● ●	● ●
Dilithium-3	● ●	● ●
Dilithium-4	● ●	● ●
MQDSS-31-48	◓ ◓	● ●
MQDSS-31-64	◓ ◓	● ●
Picnic-L1-FS	◓ ◓	● ●
Picnic-L1-UR	◓ ◓	● ●
Picnic-L3-FS	○ ○	● ●
Picnic-L3-UR	○ ○	● ●
Picnic-L5-FS	○ ○	● ●
Picnic-L5-UR	○ ○	● ●
Picnic2-L1-FS	● ●	● ●
Picnic2-L3-FS	◓ ◓	● ●
Picnic2-L5-FS	◓ ◓	● ●
qTesla-I (round 1)	● ●	● ●
qTesla-III-size (round 1)	● ●	● ●
qTesla-III-speed (round 1)	● ●	● ●
Rainbow-Ia-Classic†	◓ ◓	◓ ◓
Rainbow-Ia-Cyclic†	● ●	● ●
Rainbow-Ia-Cyclic-Compressed†	● ●	● ●
Rainbow-IIIc-Classic†	◓ ◓	○ ○
Rainbow-IIIc-Cyclic†	◓ ◓	○ ○
Rainbow-IIIc-Cyclic-Compressed†	◓ ◓	○ ○
Rainbow-Vc-Classic†	◓ ◓	○ ○
Rainbow-Vc-Cyclic†	◓ ◓	○ ○
Rainbow-Vc-Cyclic-Compressed†	◓ ◓	○ ○
SPHINCS+-{Haraka,SHA256,SHAKE256}-128f-{robust,simple}	◓ ◓	● ●
SPHINCS+-{Haraka,SHA256,SHAKE256}-128s-{robust,simple}	● ●	● ●
SPHINCS+-{Haraka,SHA256,SHAKE256}-192f-{robust,simple}	◓ ◓	● ●
SPHINCS+-{Haraka,SHA256,SHAKE256}-192s-{robust,simple}	◓ ◓	● ●
SPHINCS+-{Haraka,SHA256,SHAKE256}-256f-{robust,simple}	◓ ◓	● ●
SPHINCS+-{Haraka,SHA256,SHAKE256}-256s-{robust,simple}	◓ ◓	● ●

OpenSSH maximum packet size: 2^{18}

Summary

- Several design choices for hybrid key exchange in network protocols on negotiation and transmitting public keys, no consensus

- Protocols have size constraints which prevent some schemes from being used

- Implementations may have additional size constraints which affect some schemes, which can be bypassed with varying degrees of success

Extensions and open questions

Remaining Round 2 candidates

- Welcome help in getting code into our framework – either directly into liboqs or via PQClean

Constraints in other parts of the protocol ecosystem

- Other client/server implementations
- Middle boxes

Performance

- Latency and throughput in lab conditions
- Latency in realistic network conditions à la [Lan18]

Use in applications

- Tested our OpenSSL experiment with Apache, nginx, links, OpenVPN, with reasonable success
- More work to do:
  S/MIME, more TLS clients, …

Benchmarking PQ crypto in TLS

Christian Paquin, Douglas Stebila, Goutam Tamvada. **Benchmarking post-quantum cryptography in TLS**. November, 2019. https://eprint.iacr.org/2019/1447

Prior Work

2016

Google, with NewHope in TLS 1.2

2018

Google, with "dummy extensions"

2019

Google and Cloudflare, with SIKE and NTRU-HRSS in TLS 1.3

What if you don’t have billions of clients and millions of servers?

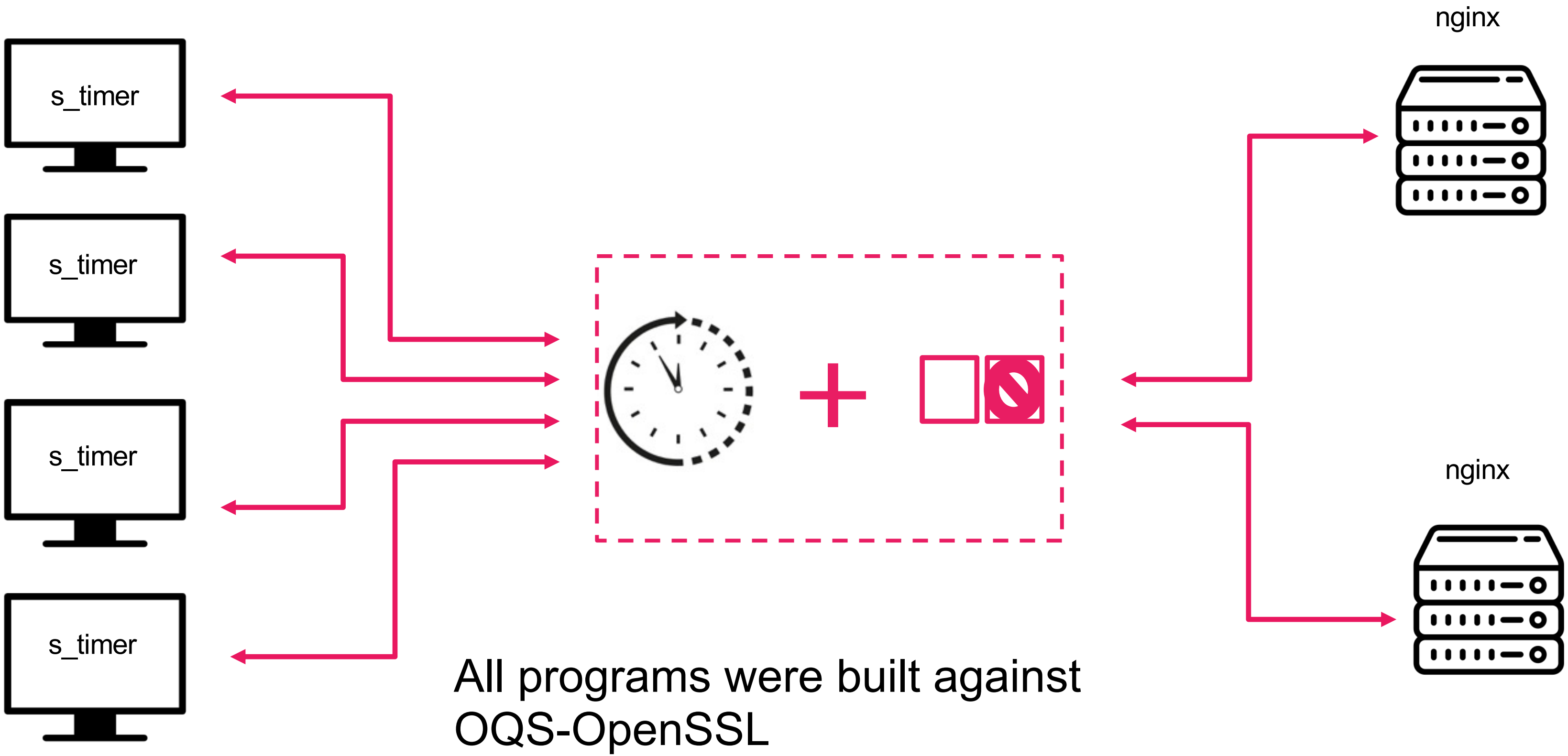
Emulate the network

+ more control over experiment parameters

+ easier to isolate effects of network characteristics

– loss in realism

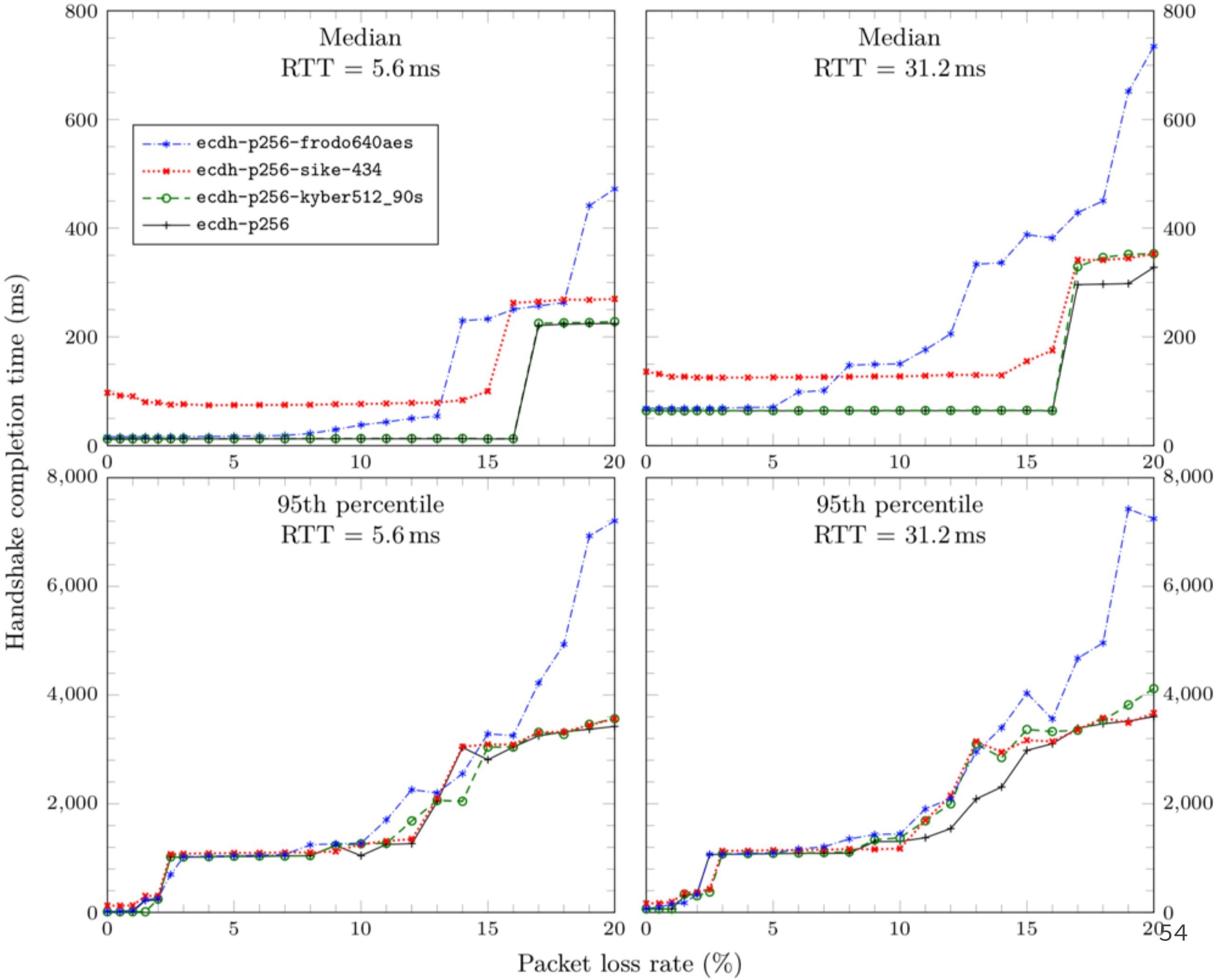
Experiment setup



All programs were built against OQS-OpenSSL

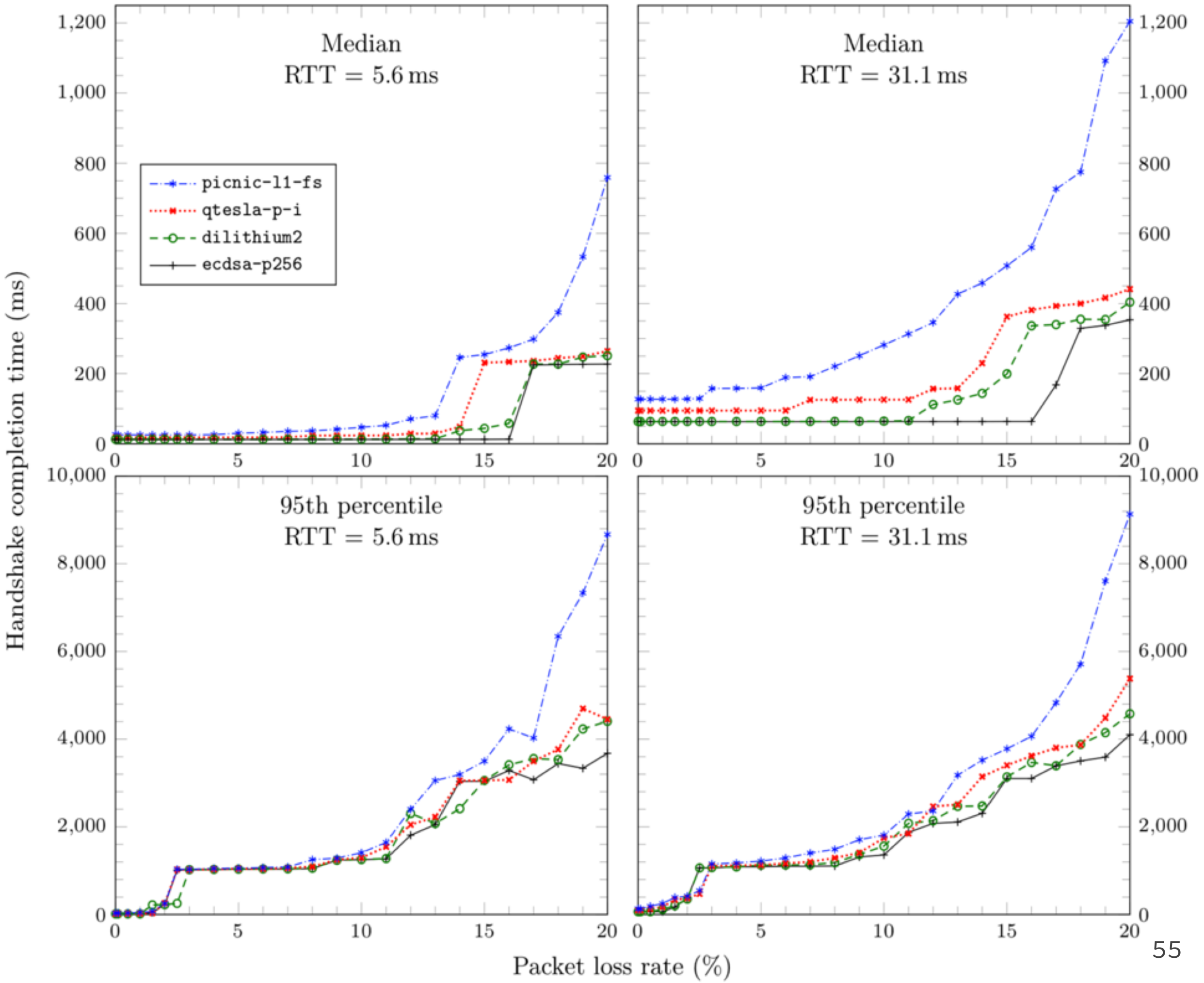
Key exchange

handshake latency as a function of packet loss rate



Authentication

handshake latency as a function of packet loss rate



Challenges in proving post-quantum key exchanges based on key encapsulation mechanisms

Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, Douglas Stebila. **Challenges in proving post-quantum key exchanges based on key encapsulation mechanisms**. Technical report. November 2019. https://eprint.iacr.org/2019/1356

Implicitly authenticated key exchange

Idea: Use **static DH + ephemeral DH** rather than signatures + ephemeral DH

Examples:

- TLS 1.2 static DH
- OPTLS (predecessor to TLS 1.3)
- Signal X3DH handshake
- QUIC original handshake
- Many protocols in the academic literature

PQ: Use **long-term KEM + ephemeral KEM** rather than signatures + ephemeral KEM

- Potentially save space since many PQ signatures are bigger than PQ KEMs

DH is too awesome

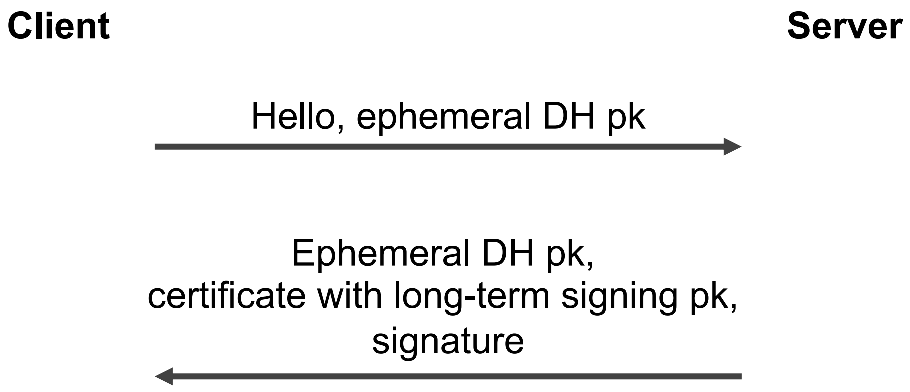
Diffie–Hellman is very flexible:

- Different message flows: serial versus parallel
- Key reuse
- Same cryptographic object for different purposes
- Range of cryptographic assumptions: from plain CDH and DDH up to interactive PRF-ODH

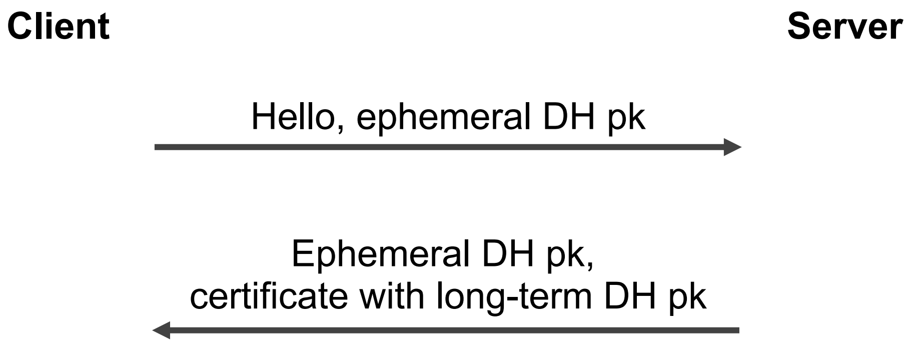
KEMs are not flexible:

- Encapsulator needs to know the public key against which they're encapsulating
- Most PQ KEMs not secure against key reuse without protection (Fujisaki–Okamoto transform)
- No known efficient methods for static–static KEM agreement (FO transform gets in the way)

Case study: TLS 1.3



Case study: TLS 1.3 implicitly authenticated DH



Session key = H(ephemeral-ephemeral, ephemeral-static)

Case study: TLS 1.3 implicitly authenticated KEMs

Client → **Server**

Hello, ephemeral KEM pk →

Would like to use this with the server's long-term KEM pk but don't know it yet

← Ephemeral KEM ciphertext, certificate with long-term KEM pk

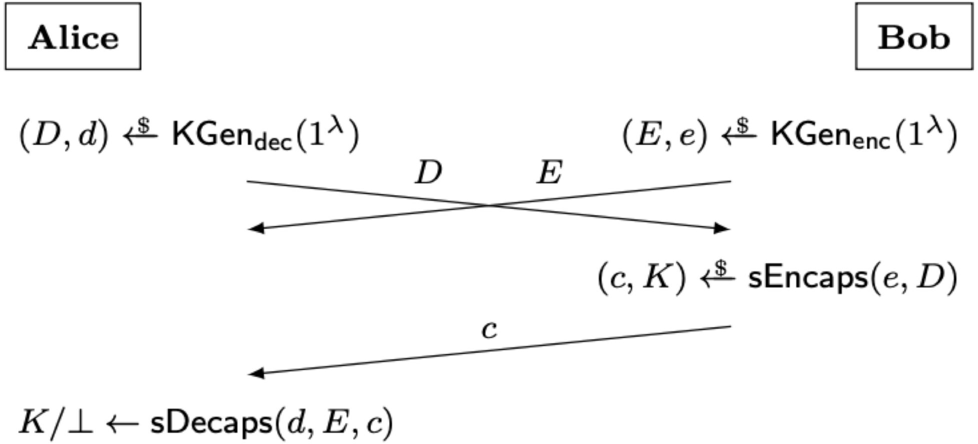
Ciphertext for long-term KEM →

So we need an extra round trip

Session key = H(ephemeral-ephemeral, ephemeral-static)

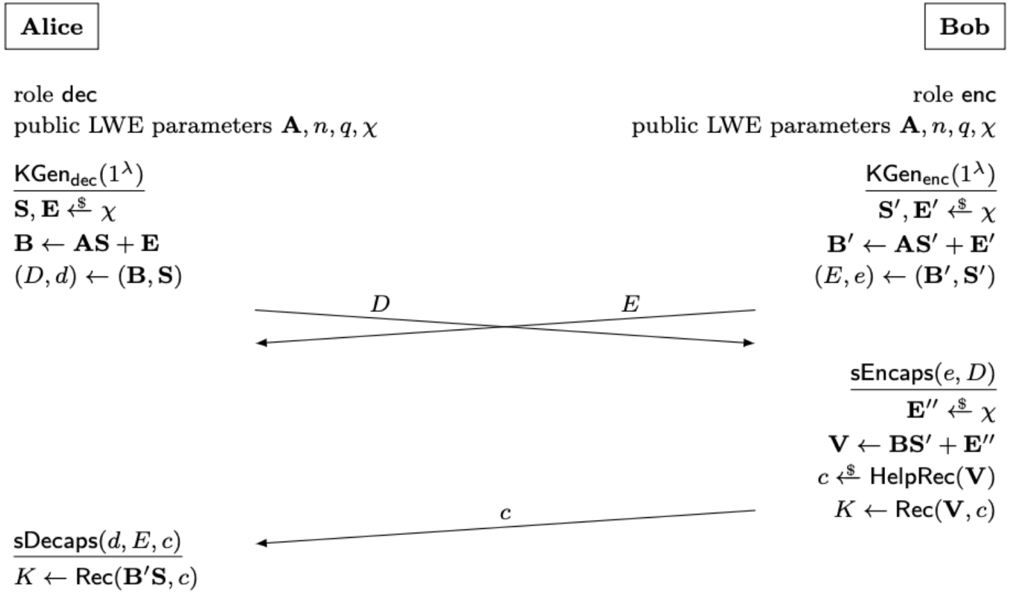
Idea: “split KEMs”

- Some LWE-based KEMs (Lindner–Peikert/Ding style) have ciphertexts part of which could be treated as a public key
- So order of public key and encapsulation could be partially swapped or separated



LWE as a split KEM

- Some LWE-based KEMs (Lindner–Peikert/Ding style) have ciphertexts part of which could be treated as a public key
- So order of public key and encapsulation could be partially swapped or separated
- **Not a full solution: couldn't figure out how to achieve active (CCA) security without FO transform**



Wrapping up

Some questions for adoption

- Hybrid key exchange: 2 or ≥ 2 algorithms?
- What level of network performance is acceptable?

Some questions for academia

- Is it safe to use an IND-CPA KEM for ephemeral key exchange in TLS 1.3?
- Can CCA-secure split KEMs be instantiated?

Exploring post-quantum cryptography in Internet protocols

Douglas Stebila

UNIVERSITY OF WATERLOO

NSERC CRSNG

https://eprint.iacr.org/2019/858

https://eprint.iacr.org/2019/1356

https://eprint.iacr.org/2019/1447

https://tools.ietf.org/html/draft-stebila-tls-hybrid-design-01

https://openquantumsafe.org/

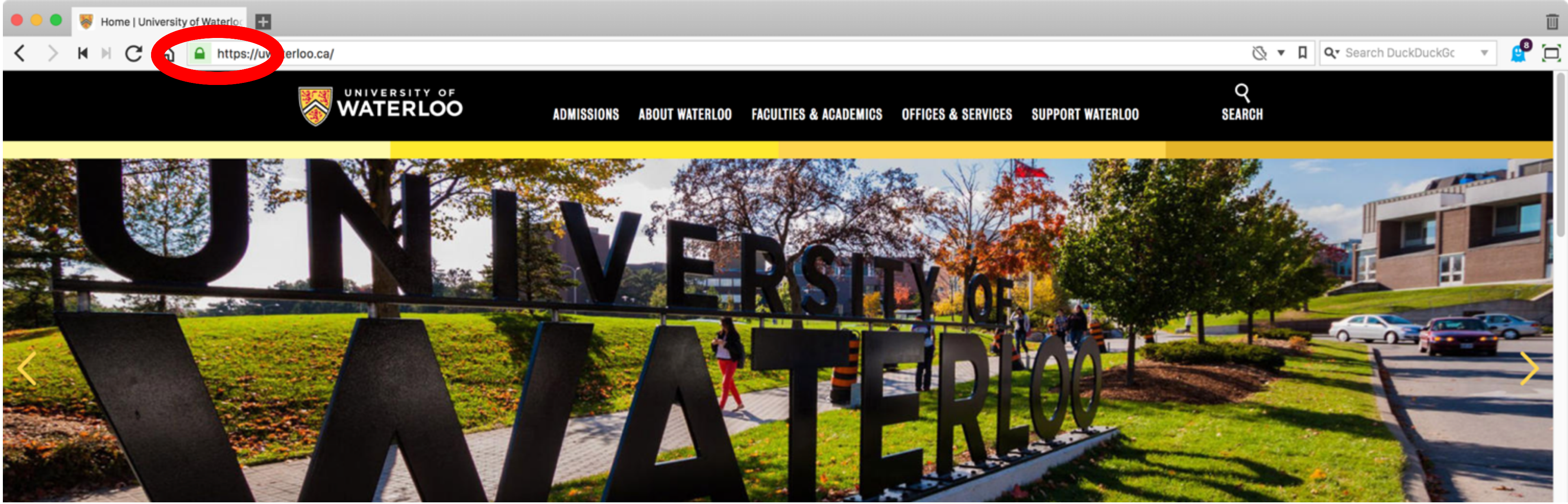
https://github.com/open-quantum-safe/

https://www.douglas.stebila.ca/

Univ. Grenoble Alpes • 2019-12-17

Appendix

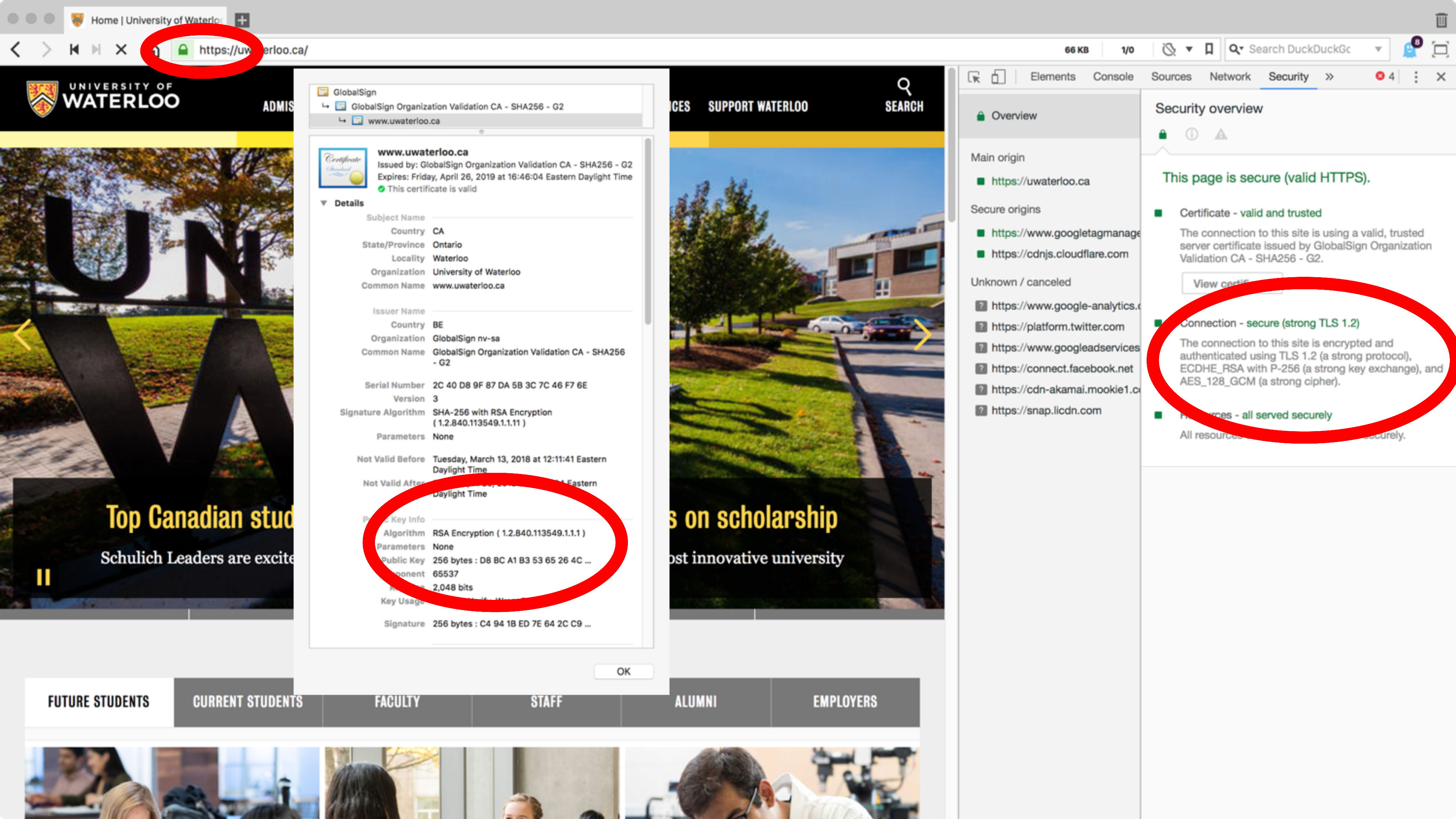
Motivating post-quantum cryptography



TLS (Transport Layer Security) protocol

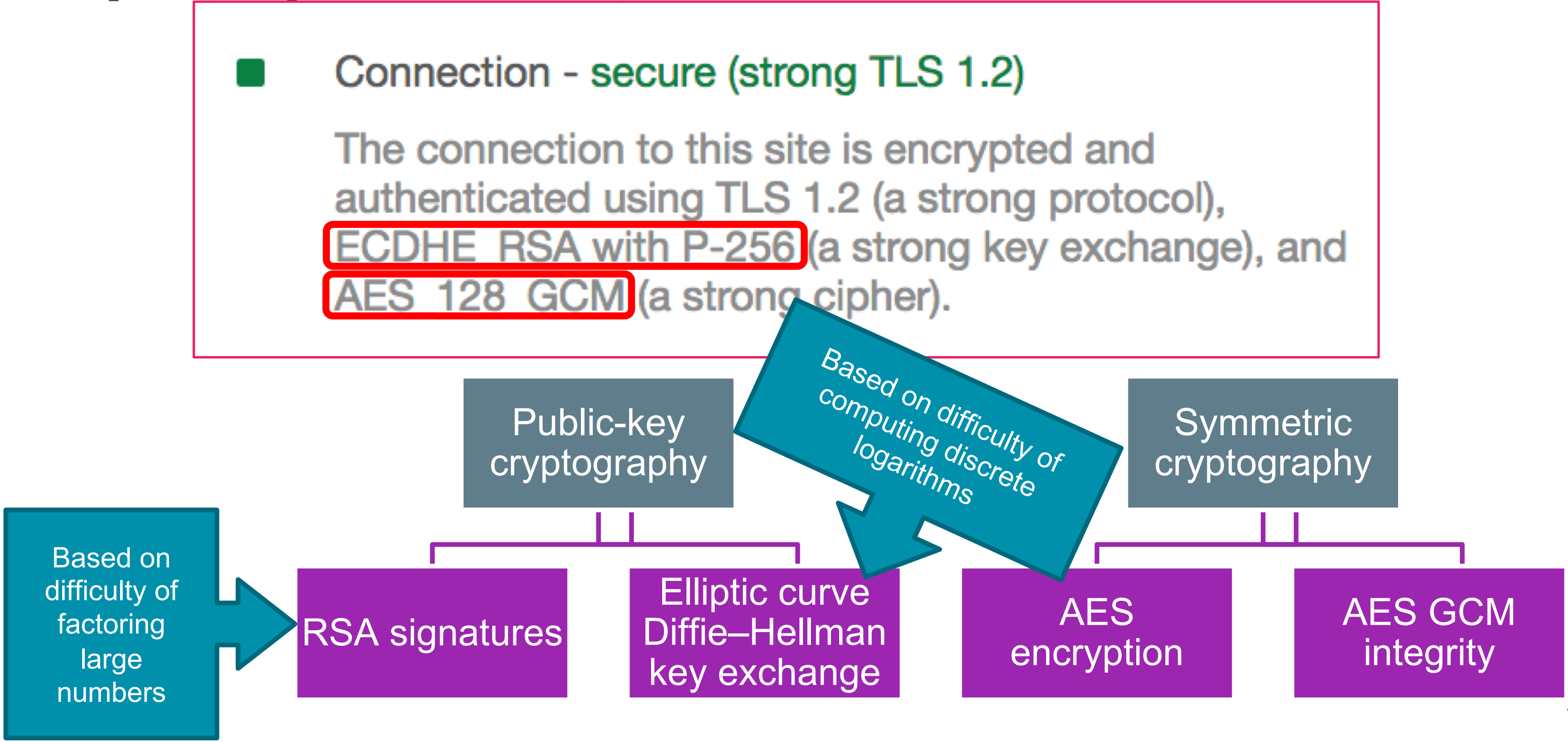
a.k.a. SSL (Secure Sockets Layer)

- The “s” in “https”
- The **most important cryptographic protocol on the Internet** — used to secure billions of connections every day.



Home | University of Waterloo
https://uw erloo.ca/
66 KB
1/0
Search DuckDuckGo
UNIVERSITY OF WATERLOO
ADMIS
CES
SUPPORT WATERLOO
SEARCH
GlobalSign
GlobalSign Organization Validation CA - SHA256 - G2
www.uwaterloo.ca
www.uwaterloo.ca
Issued by: GlobalSign Organization Validation CA - SHA256 - G2
Expires: Friday, April 26, 2019 at 16:46:04 Eastern Daylight Time
This certificate is valid
Details
Subject Name
Country CA
State/Province Ontario
Locality Waterloo
Organization University of Waterloo
Common Name www.uwaterloo.ca
Issuer Name
Country BE
Organization GlobalSign nv-sa
Common Name GlobalSign Organization Validation CA - SHA256 - G2
Serial Number 2C 40 D8 9F 87 DA 5B 3C 7C 46 F7 6E
Version 3
Signature Algorithm SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters None
Not Valid Before Tuesday, March 13, 2018 at 12:11:41 Eastern Daylight Time
Not Valid After Eastern Daylight Time
Key Info
Algorithm RSA Encryption (1.2.840.113549.1.1.1)
Parameters None
Public Key 256 bytes : D8 BC A1 B3 53 65 26 4C ...
65537
2,048 bits
Key Usage
Signature 256 bytes : C4 94 1B ED 7E 64 2C C9 ...
OK
Top Canadian stud
s on scholarship
Schulich Leaders are excite
ost innovative university
FUTURE STUDENTS
CURRENT STUDENTS
FACULTY
STAFF
ALUMNI
EMPLOYERS
Elements
Console
Sources
Network
Security
Overview
Main origin
https://uwaterloo.ca
Secure origins
https://www.googletagmanage
https://cdnjs.cloudflare.com
Unknown / canceled
https://www.google-analytics.c
https://platform.twitter.com
https://www.googleadservices
https://connect.facebook.net
https://cdn-akamai.mookie1.c
https://snap.licdn.com
Security overview
This page is secure (valid HTTPS).
Certificate - valid and trusted
The connection to this site is using a valid, trusted server certificate issued by GlobalSign Organization Validation CA - SHA256 - G2.
View certi
Connection - secure (strong TLS 1.2)
The connection to this site is encrypted and authenticated using TLS 1.2 (a strong protocol), ECDHE_RSA with P-256 (a strong key exchange), and AES_128_GCM (a strong cipher).
all served securely
All resources
ecurely.

Cryptographic building blocks



When will a large-scale quantum computer be built?

“I estimate a 1/7 chance of breaking RSA-2048 by 2026 and a 1/2 chance by 2031.”

— Michele Mosca, University of Waterloo
https://eprint.iacr.org/2015/1075

http://qurope.eu/system/files/u7/93056_Quantum%20Manifesto_WEB.pdf

Post-quantum cryptography

a.k.a. quantum-resistant algorithms

Cryptography believed to be resistant to attacks by quantum computers

Uses only classical (non-quantum) operations to implement

Not as well-studied as current encryption

- Less confident in its security
- More implementation tradeoffs

Hash-based & symmetric

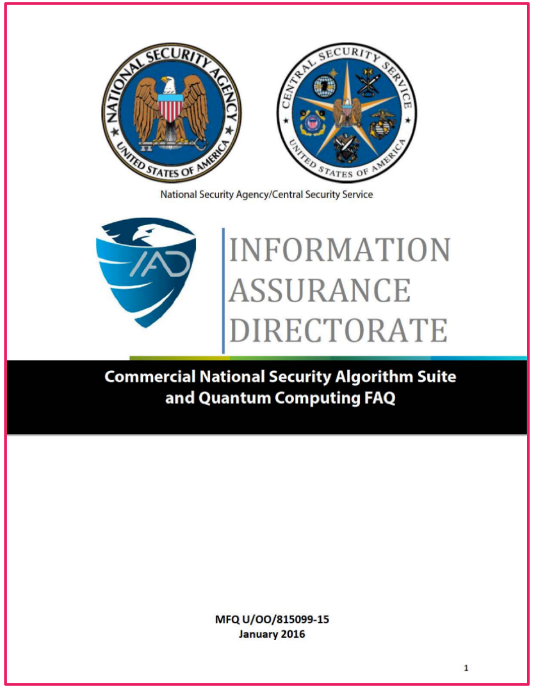
Multivariate quadratic

Code-based

Lattice-based

Elliptic curve isogenies

Standardizing post-quantum cryptography


National Security Agency/Central Security Service

INFORMATION ASSURANCE DIRECTORATE

Commercial National Security Algorithm Suite and Quantum Computing FAQ

MFQ U/OO/815099-15
January 2016

1

Aug. 2015 (Jan. 2016)

“IAD will initiate a transition to quantum resistant algorithms in the not too distant future.”

– NSA Information Assurance Directorate, Aug. 2015

Design issues for hybrid key exchange in TLS 1.3

Douglas Stebila, Scott Fluhrer, Shay Gueron. **Design issues for hybrid key exchange in TLS 1.3**. **Internet-Draft**. Internet Engineering Task Force, July 2019. https://tools.ietf.org/html/draft-stebila-tls-hybrid-design-01

Candidate Instantiation 1 – Negotiation

Follows draft-whyte-qsh-tls13-06

NamedGroup enum for supported_groups extension contains “hybrid markers” with no pre-defined meaning

Each hybrid marker points to a mapping in an extension, which lists which combinations the client proposes; between 2 and 10 algorithms permitted

supported_groups:
hybrid_marker00, hybrid_marker01, hybrid_marker02, secp256r1

HybridExtension:

- hybrid_marker00 → secp256r1+sike123+ntru456
- hybrid_marker01 → secp256r1+sike123
- hybrid_marker02 → secp256r1+ntru456

Candidate Instantiation 1 – Conveying keyshares

Client's key shares:

- Existing KeyShareClientHello allows multiple key shares
- => Send 1 key share per algorithm
  - secp256r1, sike123, ntru456
- No changes required to data structures or logic

Server's key shares:

- Respond with NamedGroup = hybrid_markerXX
- Existing KeyShareServerHello only permits one key share
- => Squeeze 2+ key shares into single key share field by concatenation

```
struct {
    KeyShareEntry key_share<2..10>;
} HybridKeyShare;
```

Instantiation 1 – Combining keys

```
                                   0
                                   |
                                   v
                    PSK ->  HKDF-Extract = Early Secret
                                   |
                                   +-----> Derive-Secret(...)
                                   +-----> Derive-Secret(...)
                                   +-----> Derive-Secret(...)
                                   |
                                  v

concatenated      0
shared            |
secret   -> HKDF-Extract        Derive-Secret(., "derived", "")
^^^^^^            |                |
                  v                v
              output ------> HKDF-Extract = Handshake Secret
              ^^^^^^               |
                                   +-----> Derive-Secret(...)
                                   +-----> Derive-Secret(...)
                                   |
                                   v
                            Derive-Secret(., "derived", "")
                                   |
                                   v
                       0 -> HKDF-Extract = Master Secret
                                   |
                                   +-----> Derive-Secret(...)
                                   +-----> Derive-Secret(...)
                                   +-----> Derive-Secret(...)
                                   +-----> Derive-Secret(...)
```

Candidate Instantiation 2 – Negotiation

Follows draft-kiefer-tls-ecdhe-sidh-00, Open Quantum Safe implementation, ...

New NamedGroup element standardized for each desired combination

No internal structure to new code points

```
enum {
    /* existing named groups */
    secp256r1 (23),
    x25519 (0x001D),
    ...,

    /* new code points eventually defined for post-quantum algorithms */
    PQ1 (0x????),
    PQ2 (0x????),
    ...,

    /* new code points defined for hybrid combinations */
    secp256r1_PQ1 (0x????),
    secp256r1_PQ2 (0x????),
    x25519_PQ1 (0x????),
    x25519_PQ2 (0x????),

    /* existing reserved code points */
    ffdhe_private_use (0x01FC..0x01FF),
    ecdhe_private_use (0xFE00..0xFEFF),
    (0xFFFF)
} NamedGroup;
```

Candidate Instantiation 2 – Conveying keyshares

KeyShareClientHello contains an entry for each code point listed in supported_groups

KeyShareServerHello contains a single entry for the chosen code point

KeyShareEntry for hybrid code points is an opaque string parsed with the following internal structure:

```
struct {
    KeyShareEntry key_share<2..10>;
} HybridKeyShare;
```

Candidate Instantiation 1

Adds new negotiation logic and ClientHello extensions

Does not result in duplicate key shares or combinatorial explosion of NamedGroups

Candidate Instantiation 2

No change in negotiation logic or data structures

No change to protocol logic: concatenation of key shares and KDFing shared secrets can be handled “internally” to a method

Results in combinatorial explosion of NamedGroups

Duplicate key shares will be sent

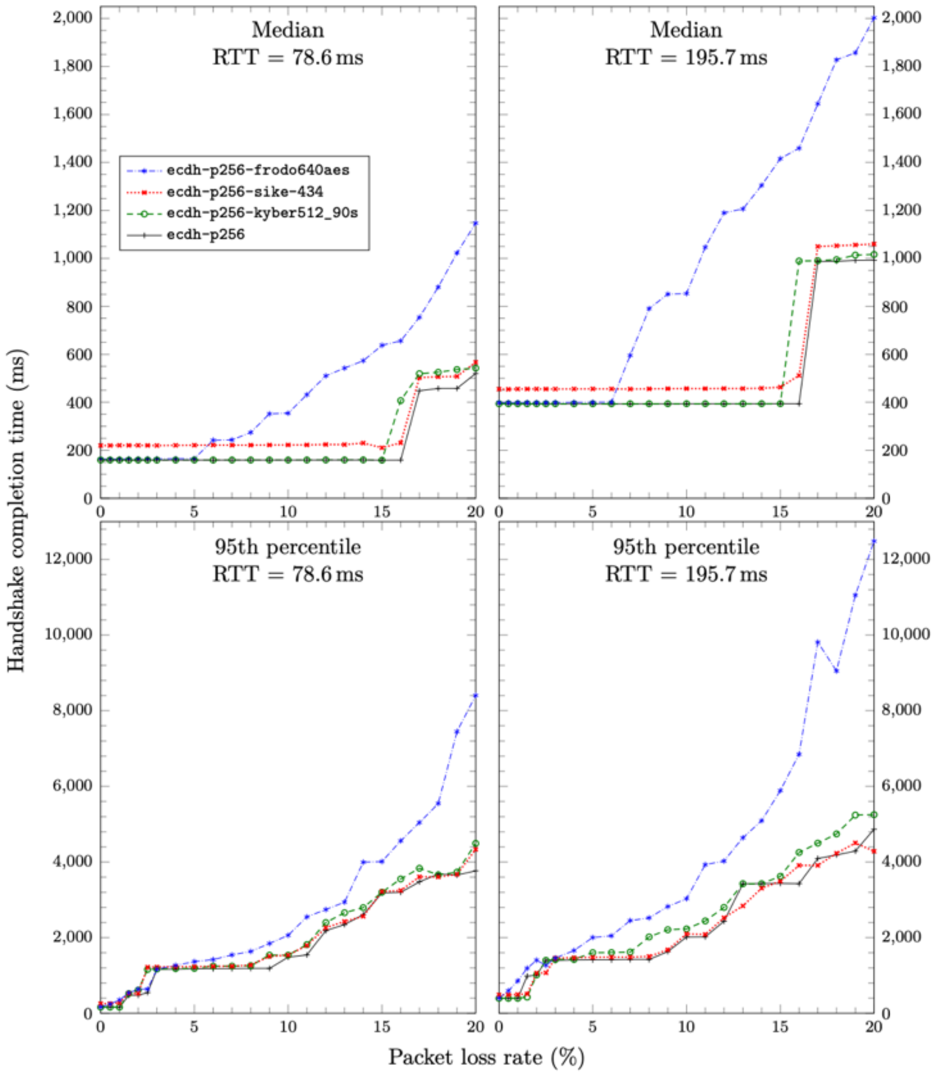
Benchmarking PQ crypto in TLS

Christian Paquin, Douglas Stebila, Goutam Tamvada. **Benchmarking post-quantum cryptography in TLS**. November, 2019. https://eprint.iacr.org/2019/1447

Key exchange

handshake latency as a function of packet loss rate

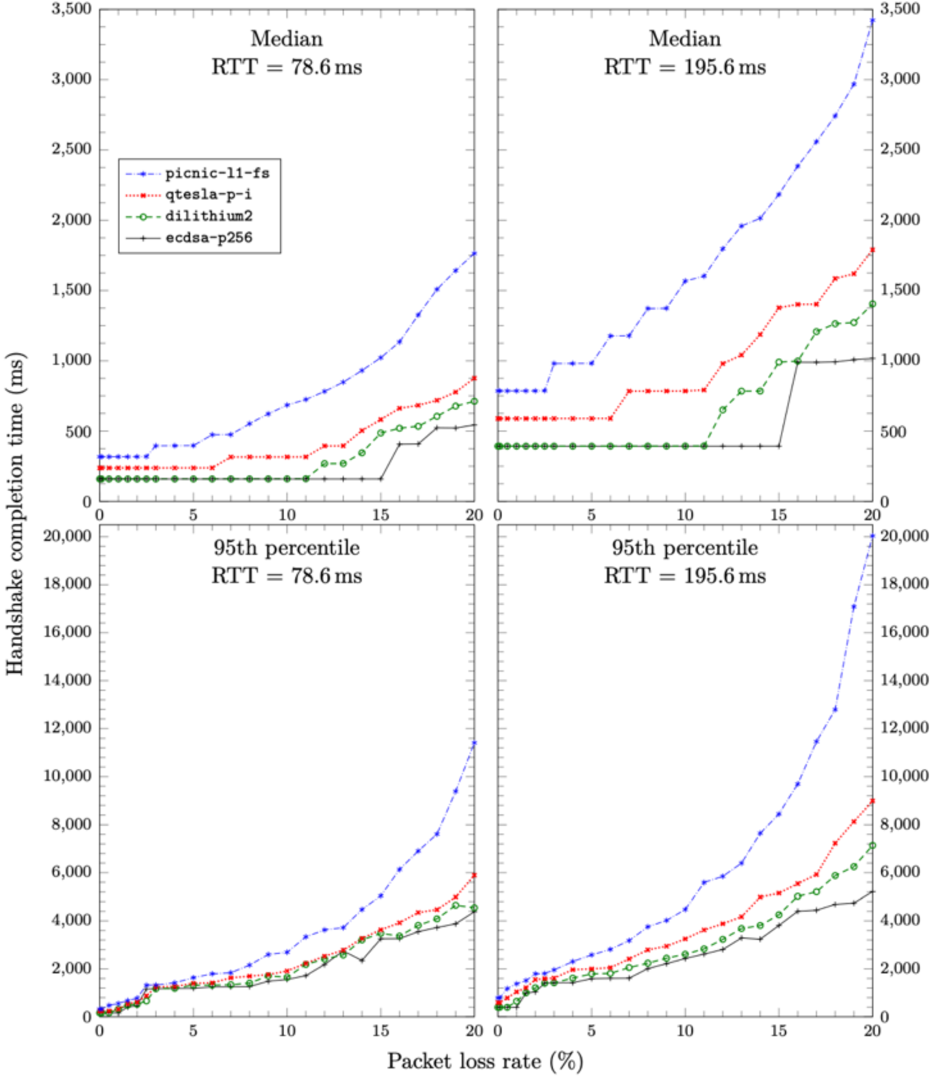
higher network latency



Authentication

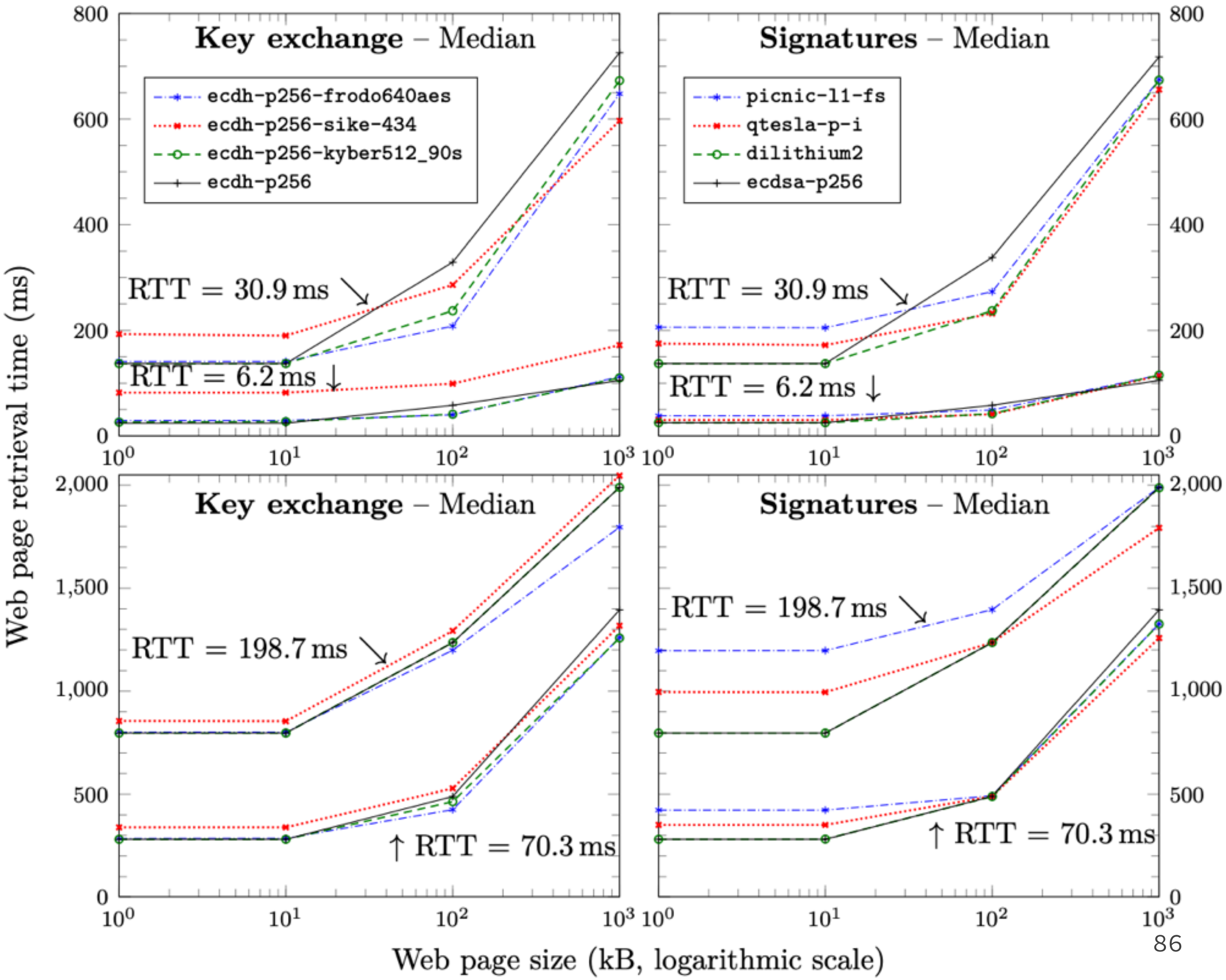
handshake latency
as a function of packet
loss rate

higher network latency



Data-centre-to-data-centre

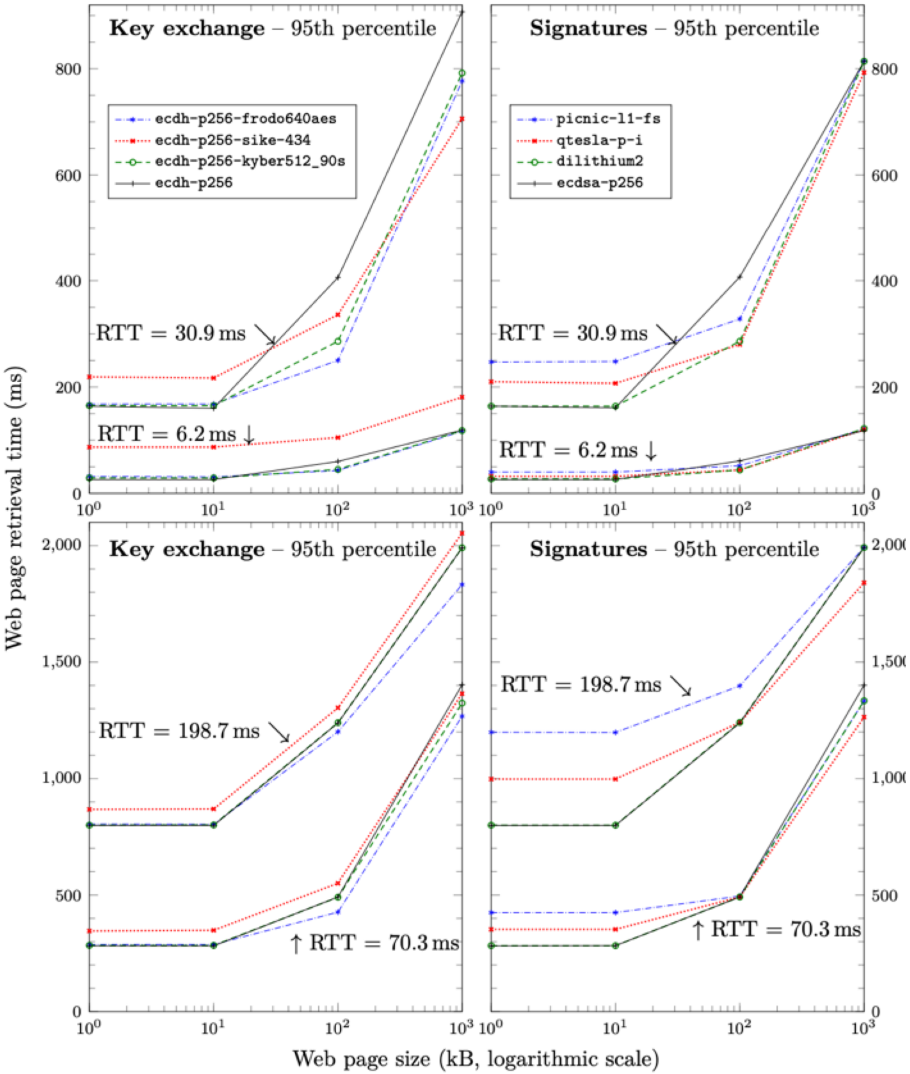
web page latency as a function of page size



Data-centre-to-data-centre

web page latency as a function of page size

higher network latency



Challenges in proving post-quantum key exchanges based on key encapsulation mechanisms

Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, Douglas Stebila. **Challenges in proving post-quantum key exchanges based on key encapsulation mechanisms**. Technical report. November 2019. https://eprint.iacr.org/2019/1356

Protocol	Core message flow	Session key	Security
SSHv2 signed ephemeral DH	$\xrightarrow{\texttt{hello}}$ $\xleftarrow{\texttt{hello}}$ $\xrightarrow{epk_A}$ $\xleftarrow{epk_B, lpk_B, \texttt{sig}}$	$\mathsf{DH}(epk_A, epk_B)$	DDH [4]
TLS 1.2 signed ephemeral DH	$\xrightarrow{\texttt{hello}}$ $\xleftarrow{epk_B, \texttt{cert}(lpk_B), \texttt{sig}}$ $\xrightarrow{epk_A}$	$\mathsf{DH}(epk_A, epk_B)$	snPRF-ODH [32]
TLS 1.3 signed ephemeral DH	$\xrightarrow{\texttt{hello}, epk_A}$ $\xleftarrow{epk_B, \texttt{cert}(lpk_B), \texttt{sig}}$	$\mathsf{DH}(epk_A, epk_B)$	snPRF-ODH [22]

Protocol	Core message flow	Session key	Security
TLS 1.2 [12] (implicitly-auth static Diffie–Hellman + explicit-auth MAC)	$\xrightarrow{\texttt{hello}}$ $\xleftarrow{\texttt{cert}[lpk_B],\texttt{mac}}$ $\xrightarrow{epk_A,\texttt{mac}}$	$\mathsf{DH}(epk_A, lpk_B)$	mnPRF-ODH [36]
OPTLS [37] (TLS 1.3–style, implicitly-auth Diffie–Hellman + explicit-auth MAC)	$\xrightarrow{\texttt{hello}, epk_A}$ $\xleftarrow{epk_B, \texttt{cert}[lpk_B], \texttt{mac}}$	$\mathsf{DH}(epk_A, epk_B)$ $\Vert\ \mathsf{DH}(epk_A, lpk_B)$	GapDH, DDH [37] (random oracle model)
Signal [54] X3DH triple handshake [+ optional ephemeral-ephemeral]	$\xrightarrow{\texttt{hello}}$ $\xleftarrow{lpk_B, sspk_B, [epk_B]}$ $\xrightarrow{lpk_A, epk_A}$	$\mathsf{DH}(lpk_A, sspk_B)$ $\Vert\ \mathsf{DH}(epk_A, lpk_B)$ $\Vert\ \mathsf{DH}(epk_A, sspk_B)$ $\Vert\ [\mathsf{DH}(epk_A, epk_B)]$	mmPRF-ODH, smPRF-ODH, smPRF-ODH, [snPRF-ODH] [7]
QUIC original handshake [41]	$\xrightarrow{\texttt{hello}, epk_A}$ $\xleftarrow{sspk_B}$	$\mathsf{DH}(epk_A, lpk_B)$ $\Vert\ \mathsf{DH}(epk_A, sspk_B)$	GapDH [25] (random oracle model)

Signal X3DH handshake

Alice | **Signal Server** | **Bob**

Alice:

identity A

static identity key (lpk_A, lsk_A)

semi-static prekey $(sspk_A, sssk_A)$

(opt.) eph. prekeys $\{(eppk_A^i, epsk_A^i)\}_i$

Bob:

identity B

static identity key (lpk_B, lsk_B)

semi-static prekey $(sspk_B, sssk_B)$

(opt.) eph. prekeys $\{(eppk_B^i, epsk_B^i)\}_i$

Signal Server → Alice: $lpk_B, sspk_B, eppk_B$

Signal Server → Bob: lpk_A

Alice:

$(epk_A, esk_A) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$

$\mathsf{ms} \leftarrow sspk_B^{lsk_A} \| lpk_B^{esk_A} \| sspk_B^{esk_A} \| eppk_B^{esk_A}$

$K \leftarrow \mathsf{F}(\mathsf{ms}, \cdot)$

Alice → Bob: epk_A

Bob:

$\mathsf{ms} \leftarrow lpk_A^{sssk_B} \| epk_A^{lsk_B} \| epk_A^{sssk_B} \| epk_A^{epsk_B}$

$K \leftarrow \mathsf{F}(\mathsf{ms}, \cdot)$

Signal handshake with KEMs

Alice | **Signal Server** | **Bob**

Alice:

identity A

static identity key (lpk_A, lsk_A)

semi-static prekey $(sspk_A, sssk_A)$

(opt.) eph. prekeys $\{(eppk^i_A, epsk^i_A)\}_i$

Bob:

identity B

static identity key (lpk_B, lsk_B)

semi-static prekey $(sspk_B, sssk_B)$

(opt.) eph. prekeys $\{(eppk^i_B, epsk^i_B)\}_i$

Server → Alice: $lpk_B, sspk_B, eppk_B$

Server → Bob: lpk_A

Alice:

$(c_1, K_1) \xleftarrow{\$} \mathsf{Encaps}(lpk_B)$

$(c_2, K_2) \xleftarrow{\$} \mathsf{Encaps}(sspk_B)$

$(c_3, K_3) \xleftarrow{\$} \mathsf{Encaps}(eppk_B)$

Alice → Bob: c_1, c_2, c_3

Bob:

$K_1 \leftarrow \mathsf{Decaps}(sssk_B, c_1)$

$K_2 \leftarrow \mathsf{Decaps}(lsk_B, c_2)$

$K_3 \leftarrow \mathsf{Decaps}(epsk_B, c_3)$

$(c_4, K_4) \xleftarrow{\$} \mathsf{Encaps}(lpk_A)$

$\mathsf{ms} \leftarrow K_4 || K_1 || K_2 || K_3$

$K \leftarrow \mathsf{F}(\mathsf{ms}, \cdot)$

Bob → Alice: c_4

Alice:

$K_4 \leftarrow \mathsf{Decaps}(lsk_A, c_4)$

$\mathsf{ms} \leftarrow K_4 || K_1 || K_2 || K_3$

$K \leftarrow \mathsf{F}(\mathsf{ms}, \cdot)$

Signal handshake with split KEMs

Alice | **Signal Server** | **Bob**

Alice:

identity A

static identity key (lpk_A, lsk_A)

semi-static prekey $(sspk_A, sssk_A)$

(opt.) eph. prekeys $\{(eppk^i_A, epsk^i_A)\}_i$

Bob:

identity B

static identity key (lpk_B, lsk_B)

semi-static prekey $(sspk_B, sssk_B)$

(opt.) eph. prekeys $\{(eppk^i_B, epsk^i_B)\}_i$

Server → Alice: $lpk_B, sspk_B, eppk_B$

Server → Bob: lpk_A

Alice:

$(epk_A, esk_A) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$

$(c_1, K_1) \xleftarrow{\$} \mathsf{sEncaps}(lsk_A, sspk_B)$

$(c_2, K_2) \xleftarrow{\$} \mathsf{sEncaps}(esk_A, lpk_B)$

$(c_3, K_3) \xleftarrow{\$} \mathsf{sEncaps}(esk_A, sspk_B)$

$(c_4, K_4) \xleftarrow{\$} \mathsf{sEncaps}(esk_A, eppk_B)$

$\mathsf{ms} \leftarrow K_1 || K_2 || K_3 || K_4$

$K \leftarrow \mathsf{F}(\mathsf{ms}, \cdot)$

Alice → Bob: $epk_A, c_1, c_2, c_3, c_4$

Bob:

$K_1 \leftarrow \mathsf{sDecaps}(sssk_B, lpk_A, c_1)$

$K_2 \leftarrow \mathsf{sDecaps}(lsk_B, epk_A, c_2)$

$K_3 \leftarrow \mathsf{sDecaps}(sssk_B, epk_A, c_3)$

$K_4 \leftarrow \mathsf{sDecaps}(epsk_B, epk_A, c_4)$

$\mathsf{ms} \leftarrow K_1 || K_2 || K_3 || K_4$

$K \leftarrow \mathsf{F}(\mathsf{ms}, \cdot)$