Open Quantum Safe update and Post-quantum TLS without handshake signatures

Douglas Stebila

UNIVERSITY OF WATERLOO

NSERC CRSNG

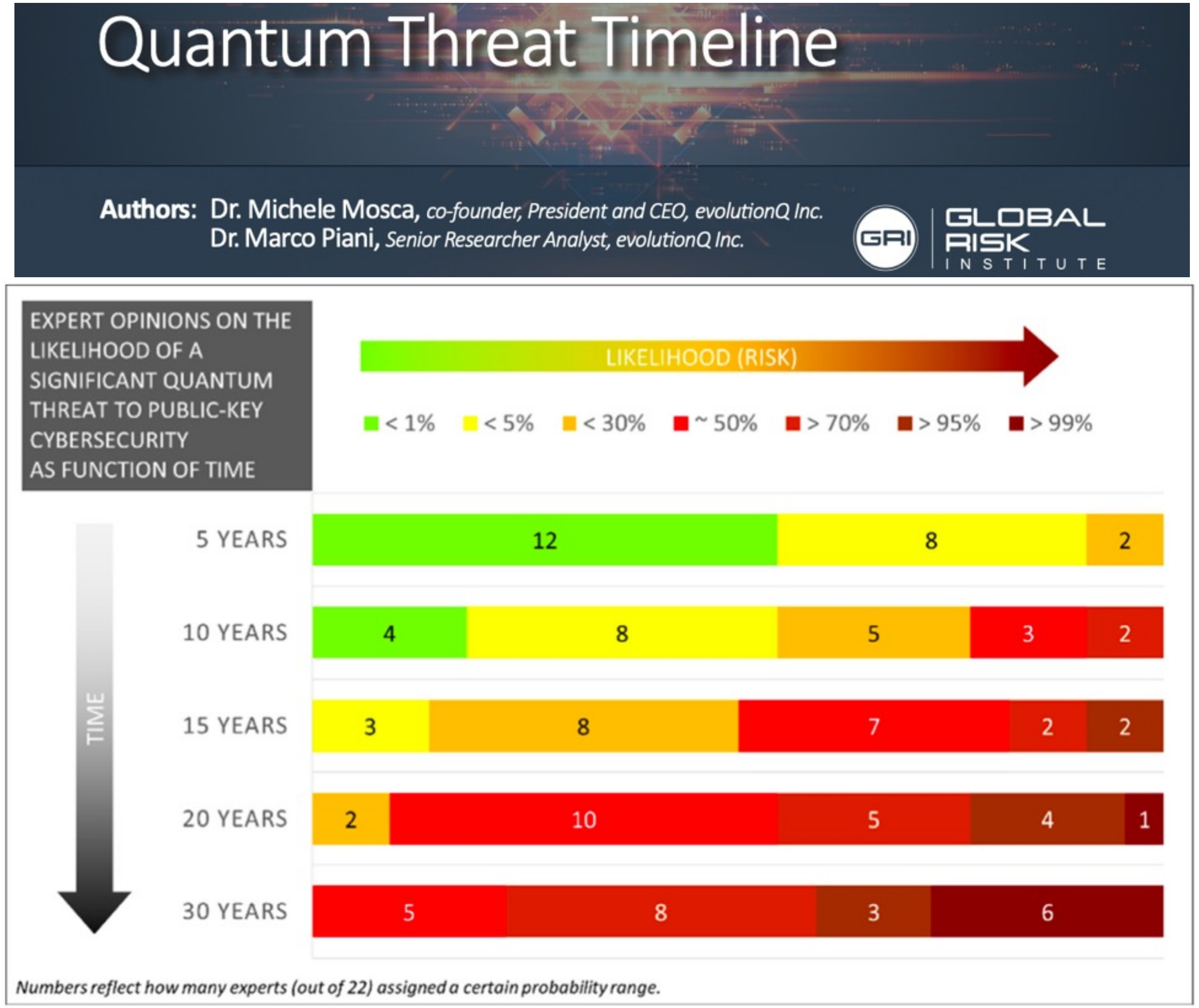
UNIVERSITY OF WATERLOO

IQC Institute for Quantum Computing

CYBER SECURITY AND PRIVACY INSTITUTE
UNIVERSITY OF WATERLOO

Cryptography @ University of Waterloo

- UW involved in 4 NIST PQC Round 3 submissions:
  - Finalists: CRYSTALS-Kyber, NTRU
  - Alternates: FrodoKEM, SIKE
- UW involved in 4 NIST Lightweight Crypto Round 2 submissions: ACE, SPIX, SpoC, WAGE
- Elliptic curves: David Jao, Alfred Menezes, (Scott Vanstone)
- Information theoretic cryptography: Doug Stinson
- Privacy-enhancing technologies: Ian Goldberg
- Quantum cryptanalysis: Michele Mosca
- Quantum cryptography: Norbert Lütkenhaus, Thomas Jennewein, Debbie Leung
- Gord Agnew, Vijay Ganesh, Guang Gong, Sergey Gorbunov, Anwar Hasan, Florian Kerschbaum



https://globalriskinstitute.org/publications/quantum-threat-timeline/

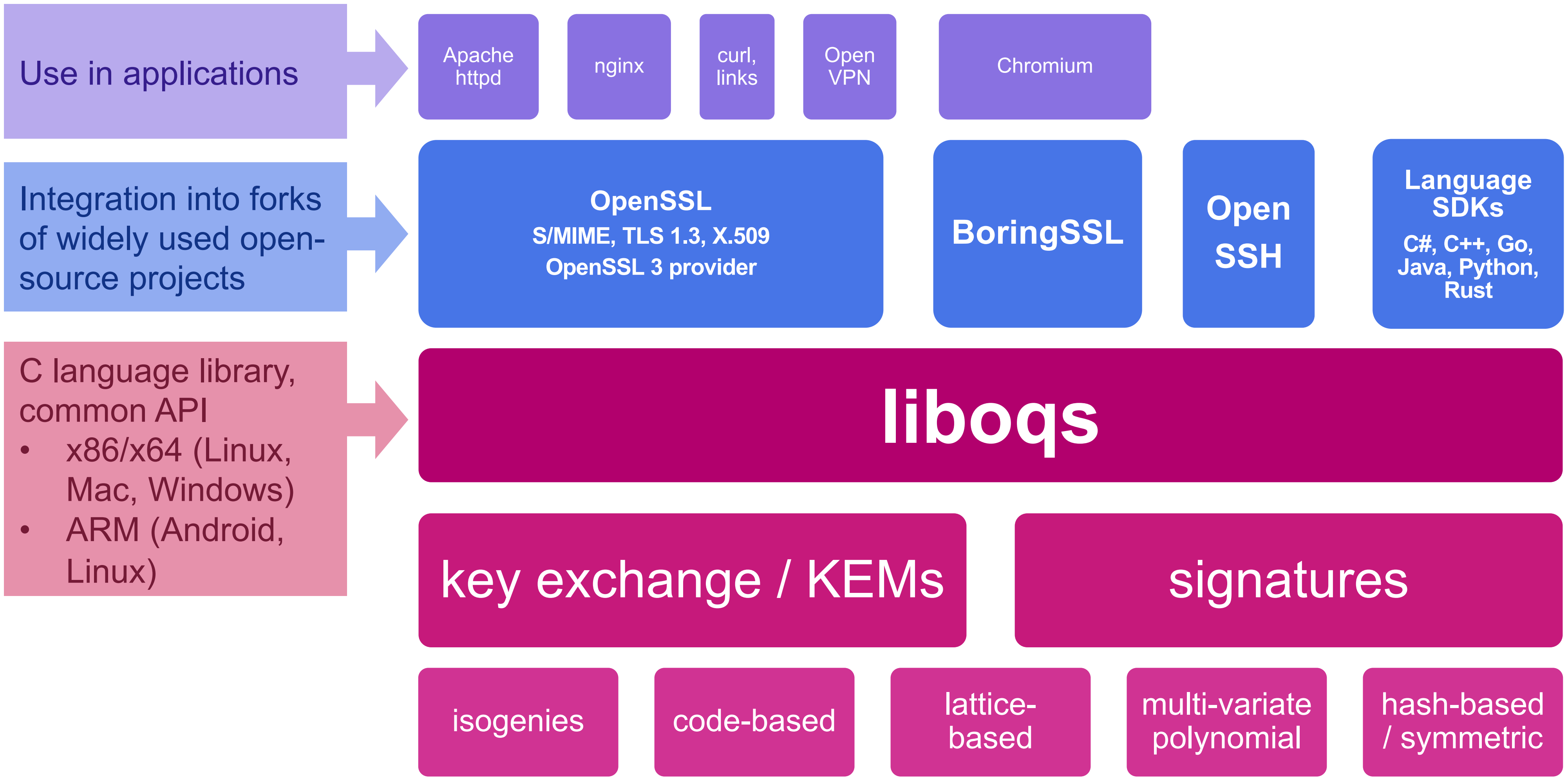
OPEN QUANTUM SAFE

software for prototyping quantum-resistant cryptography

https://openquantumsafe.org

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

Open Quantum Safe Project



Industry partners:
- Amazon Web Services
- evolutionQ
- IBM Research
- Microsoft Research

Additional contributors:
- Cisco
- Senetas
- PQClean project
- Individuals

Financial support:
- AWS
- Canadian Centre for Cyber Security
- NSERC
- Unitary Fund

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

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

- Version 0.5.0 released March 2021
- Includes all Round 3 finalists and alternate candidates
  - (except GeMSS)
  - Some implementations still Round 2 versions

https://openquantumsafe.org/liboqs/

TLS 1.3 implementations

	OQS-OpenSSL 1.1.1	OQS-OpenSSL 3 provider	OQS-BoringSSL
PQ key exchange in TLS 1.3	Yes	Yes	Yes
Hybrid key exchange in TLS 1.3	Yes	Coming soon	Yes
PQ certificates and signature authentication in TLS 1.3	Yes	No	Yes
Hybrid certificates and signature authentication in TLS 1.3	Yes	No	No

Using draft-ietf-tls-hybrid-design for hybrid key exchange

Interoperability test server running at https://test.openquantumsafe.org

https://openquantumsafe.org/applications/tls/

Applications

- Demonstrator application integrations into:
  - Apache
  - nginx
  - haproxy
  - curl
  - Chromium
- In most cases required few/no modifications to work with updated OpenSSL
- Runnable Docker images available for download

https://openquantumsafe.org/applications/tls/#demo-integrations

Benchmarking

- New benchmarking portal at https://openquantumsafe.org/benchmarking/

- Core algorithm speed and memory usage
- TLS performance in ideal network conditions
- Intel AVX2 and ARM 64

Part 2:
Post-quantum TLS without handshake signatures

Peter Schwabe, Douglas Stebila, Thom Wiggers. In Proc. 27th ACM Conference on Computer and Communications Security (CCS) 2020. ACM, November 2020.
https://eprint.iacr.org/2020/534

Authenticated key exchange

- Two parties establish a shared secret over a public communication channel

Vast literature on AKE protocols

- Many **security definitions** capturing various adversarial powers: BR, CK, eCK, …
- Different types of **authentication credentials**: public key, shared secret key, password, identity-based, …
- **Additional security goals**: weak/strong forward secrecy, key compromise impersonation resistance, post-compromise security, …
- Additional **protocol functionality**: multi-stage, ratcheting, …
- **Group** key exchange
- **Real-world protocols**: TLS, SSH, Signal, IKE, ISO, EMV, …
- …

Explicit authentication

Alice receives assurance that she really is talking to Bob

Implicit authentication

Alice is assured that only Bob would be able to compute the shared secret

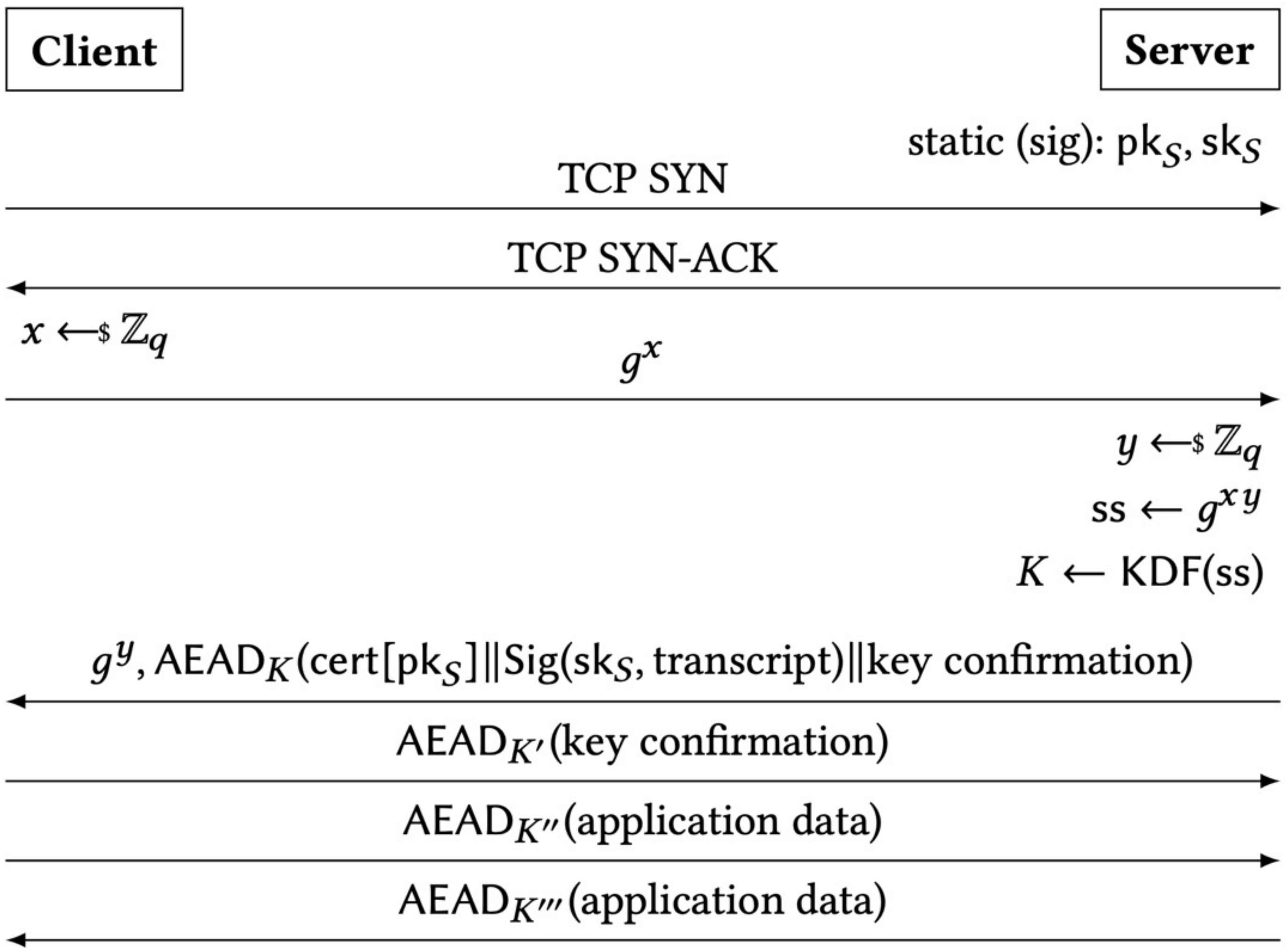
Explicitly authenticated key exchange: Signed Diffie–Hellman

Alice		**Bob**
$(pk_A, sk_A) \leftarrow \text{SIG.KeyGen}()$		$(pk_B, sk_B) \leftarrow \text{SIG.KeyGen}()$
obtain pk_B		obtain pk_A
$x \leftarrow_\$ \{0, \ldots, q-1\}$		
$X \leftarrow g^x$	X $\longrightarrow$	
		$y \leftarrow_\$ \{0, \ldots, q-1\}$
		$Y \leftarrow g^y$
	$\longleftarrow$ Y, σ_B	$\sigma_B \leftarrow \text{SIG.Sign}(sk_B, A \| B \| X \| Y)$
$\sigma_A \leftarrow \text{SIG.Sign}(sk_A, A \| B \| X \| Y)$	σ_A $\longrightarrow$	
$k \leftarrow H(sid, Y^x)$		$k \leftarrow H(sid, X^y)$
	application data $\Longleftrightarrow$ using authenticated encryption	

Implicitly authenticated key exchange: Double-DH

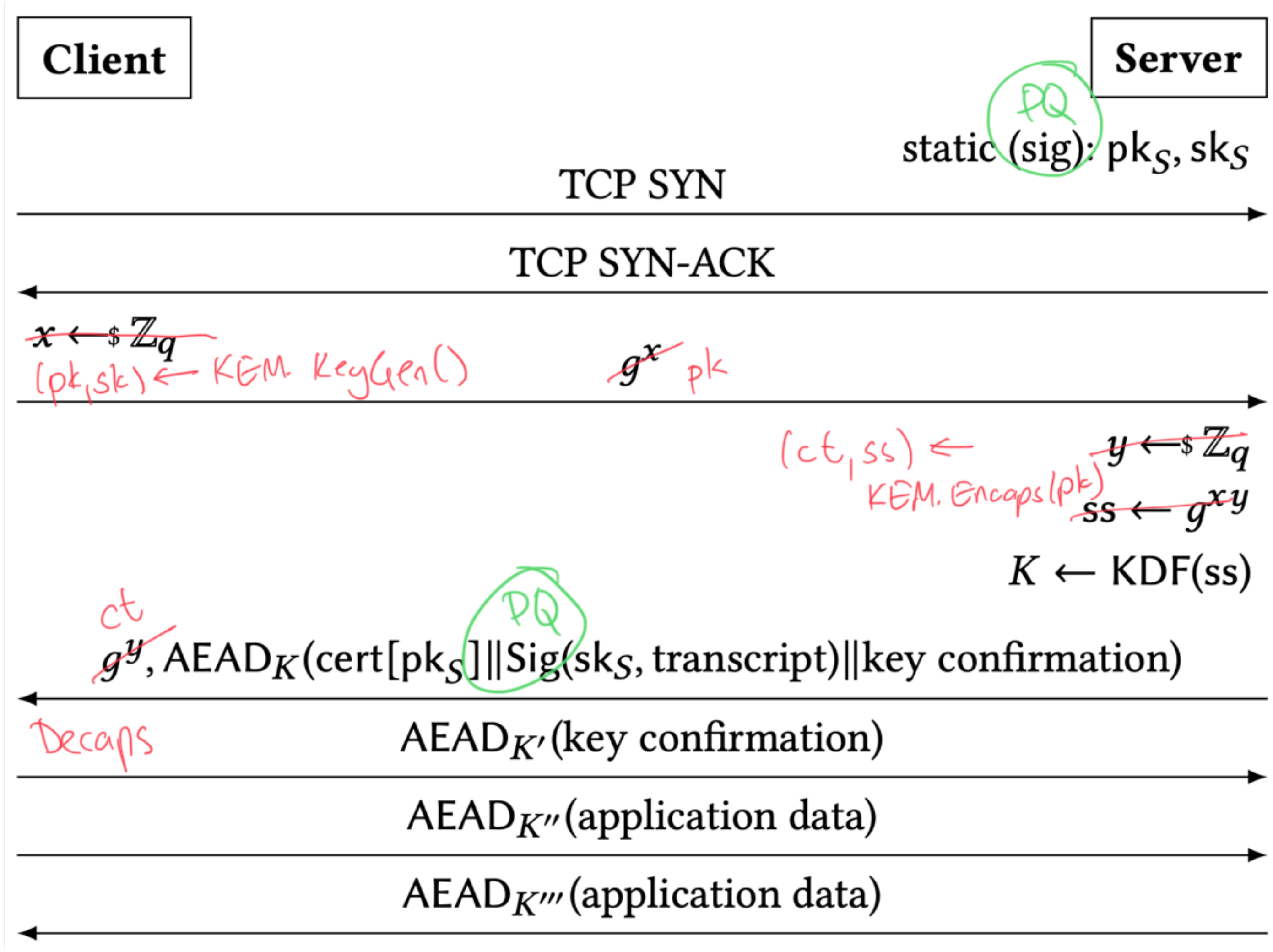
Alice		**Bob**
$sk_A \leftarrow_\$ \{0, \ldots, q-1\}$		$sk_B \leftarrow_\$ \{0, \ldots, q-1\}$
$pk_A \leftarrow g^{sk_A}$		$pk_B \leftarrow g^{sk_B}$
obtain pk_B		obtain pk_A
$x \leftarrow_\$ \{0, \ldots, q-1\}$		$y \leftarrow_\$ \{0, \ldots, q-1\}$
$X \leftarrow g^x$	$\xrightarrow{X}$	$Y \leftarrow g^y$
	$\xleftarrow{Y}$	
$k \leftarrow H(sid,\ pk_B^{sk_A} \Vert Y^x)$		$k \leftarrow H(sid,\ \boxed{pk_A^{sk_B}} \Vert X^y)$
	application data $\Longleftrightarrow$ using authenticated encryption	

TLS 1.3 handshake



Signed Diffie–Hellman

TLS 1.3 handshake



~~Signed Diffie–Hellman~~

Post-Quantum!!!

Problem

post-quantum
signatures
are big

Signature scheme		Public key (bytes)	Signature (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
Dilithium	**Lattice-based (MLWE/MSIS)**	**1,184**	**2,044**
Falcon	**Lattice-based (NTRU)**	**897**	**690**
XMSS	**Hash-based**	**32**	**979**
GeMSS	**Multi-variate**	**352,180**	**32**

Solution

use
post-quantum KEMs
for authentication

Key encapsulation mechanisms (KEMs)

An abstraction of Diffie–Hellman key exchange

$(pk, sk) \leftarrow \mathsf{KEM.KeyGen}()$

$$\xrightarrow{\quad pk \quad}$$

$(ct, k) \leftarrow \mathsf{KEM.Encaps}(pk)$

$$\xleftarrow{\quad ct \quad}$$

$k \leftarrow \mathsf{KEM.Decaps}(sk, ct)$

Signature scheme		Public key (bytes)	Signature (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
Dilithium	**Lattice-based (MLWE/MSIS)**	**1,184**	**2,044**
Falcon	**Lattice-based (NTRU)**	**897**	**690**
XMSS	**Hash-based**	**32**	**979**
GeMSS	**Multi-variate**	**352,180**	**32**

KEM		Public key (bytes)	Ciphertext (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
Kyber	**Lattice-based (MLWE)**	**800**	**768**
NTRU	**Lattice-based (NTRU)**	**699**	**699**
Saber	**Lattice-based (MLWR)**	**672**	**736**
SIKE	**Isogeny-based**	**330**	**330**
SIKE compressed	**Isogeny-based**	**197**	**197**

Implicitly authenticated KEX is not new

In theory

- DH-based: SKEME, MQV, HMQV, …
- KEM-based: BCGP09, FSXY12, …

In practice

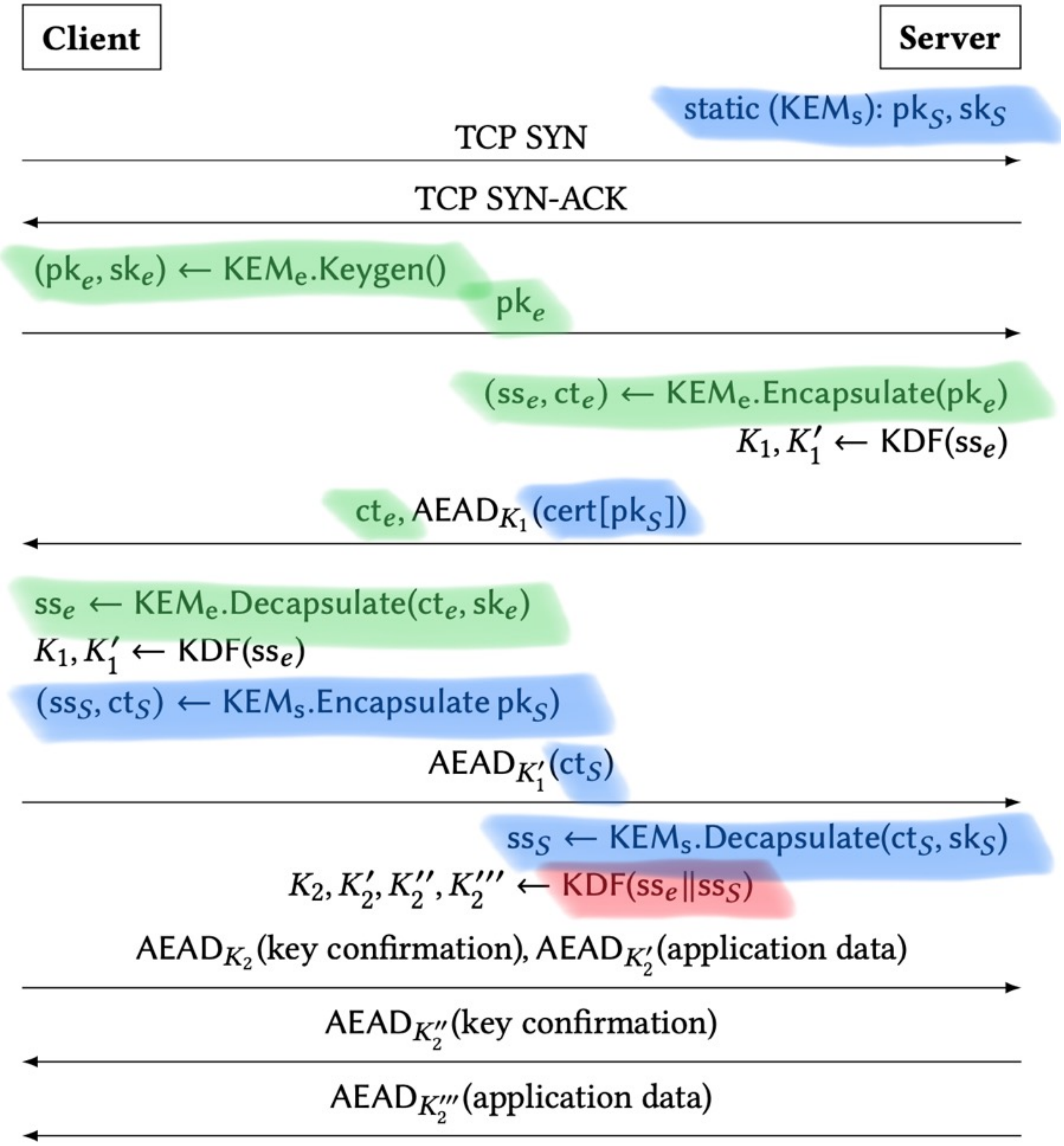
- RSA key transport in TLS ≤ 1.2
  - Lacks forward secrecy
- Signal, Noise, Wireguard
  - DH-based
  - Different protocol flows
- OPTLS
  - DH-based
  - Requires a non-interactive key exchange (NIKE)

“KEMTLS” handshake

KEM for
ephemeral key exchange

KEM for
server-to-client
authenticated key exchange

Combine shared secrets



Algorithm choices

KEM for ephemeral key exchange

- IND-CCA (or IND-1CCA)
- Want small public key + small ciphertext

KEM for authenticated key exchange

- IND-CCA
- Want small public key + small ciphertext

Signature scheme for intermediate CA

- Want small public key + small signature

Signature scheme for root CA

- Want small signature

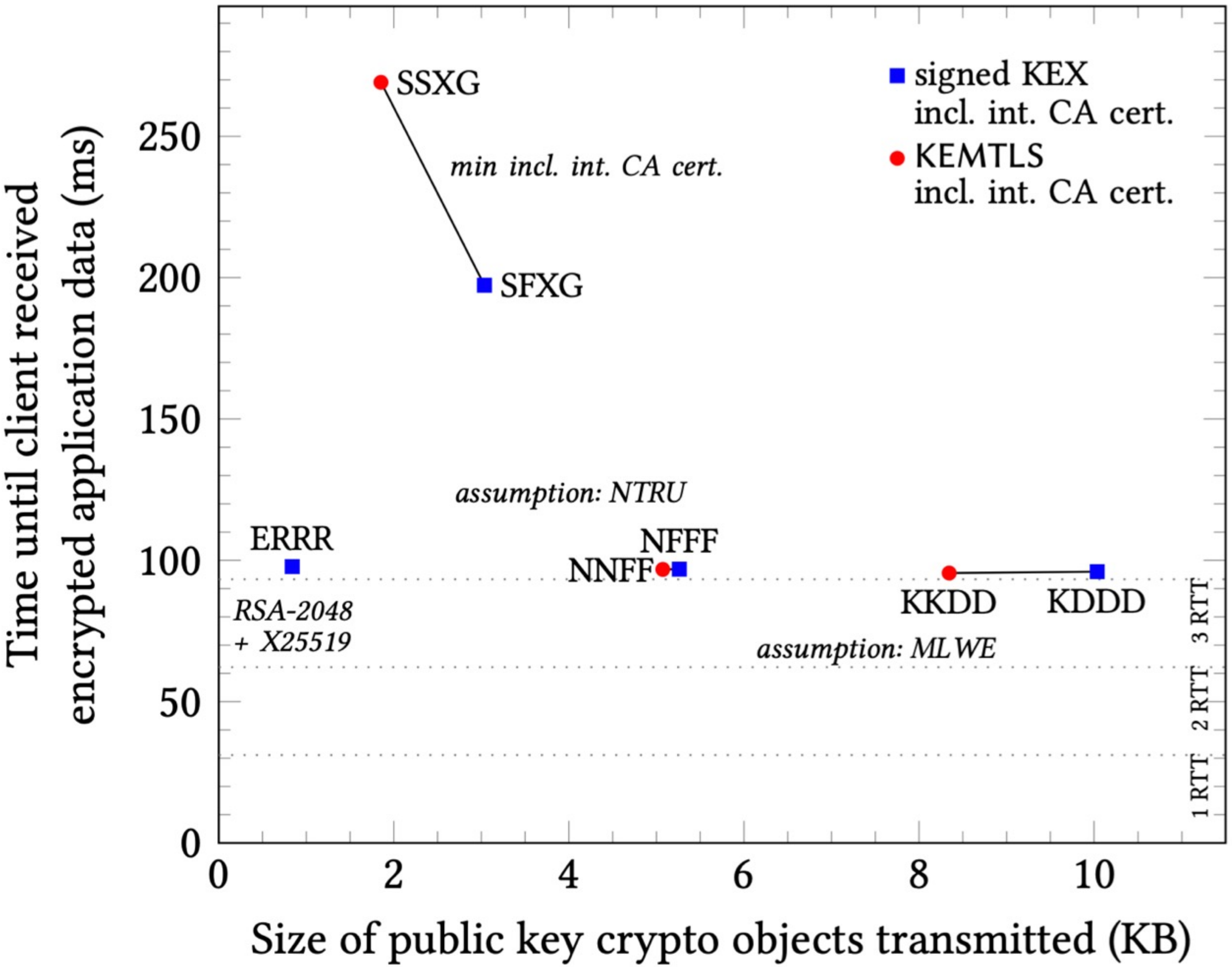
4 scenarios

1. Minimize size when intermediate certificate transmitted
2. Minimize size when intermediate certificate not transmitted (cached)
3. Use solely NTRU assumptions
4. Use solely module LWE/SIS assumptions

Signed KEX versus KEMTLS

Labels ABCD:
A = ephemeral KEM
B = leaf certificate
C = intermediate CA
D = root CA

Algorithms: (all level 1)
Dilithium,
ECDH X25519,
Falcon,
GeMSS,
Kyber,
NTRU,
RSA-2048,
SIKE,
XMSS'

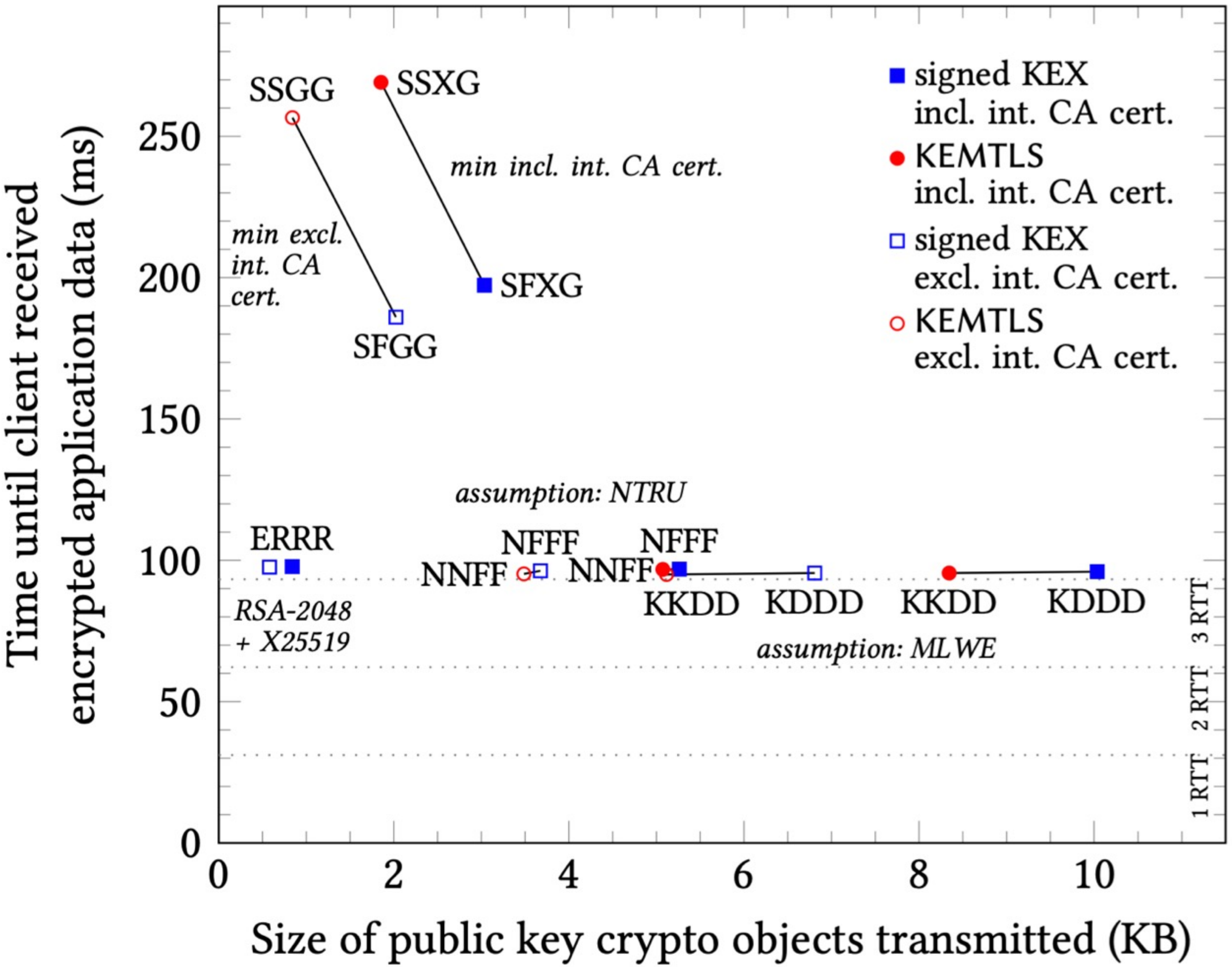


Rustls client/server with AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

Signed KEX versus KEMTLS

Labels ABCD:
A = ephemeral KEM
B = leaf certificate
C = intermediate CA
D = root CA

Algorithms: (all level 1)
Dilithium,
ECDH X25519,
Falcon,
GeMSS,
Kyber,
NTRU,
RSA-2048,
SIKE,
XMSS'



Rustls client/server with AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

Observations

- Size-optimized KEMTLS requires < ½ communication of size-optimized PQ signed-KEM
- Speed-optimized KEMTLS uses 90% fewer server CPU cycles and still reduces communication
  - NTRU KEX (27 μs) 10x faster than Falcon signing (254 μs)
- No extra round trips required until client starts sending application data
- Smaller trusted code base (no signature generation on client/server)

Security

Security model: multi-stage key exchange, extending [DFGS21]

- Key indistinguishability
- Forward secrecy
- Implicit and explicit authentication

Ingredients in security proof:

- **IND-CCA for long-term KEM**
- **IND-1CCA for ephemeral KEM**
- Collision-resistant hash function
- Dual-PRF security of HKDF
- EUF-CMA of HMAC

[DFGS21] Dowling, Fischlin, Günther, Stebila. Journal of Cryptology, 2021. https://eprint.iacr.org/2020/1044

Security subtleties: authentication

Implicit authentication

- Client's first application flow can't be read by anyone other than intended server, but client doesn't know server is live at the time of sending
- Also provides a form of deniable authentication since no signatures are used
  - Formally: offline deniability [DGK06]

Explicit authentication

- Explicit authentication once key confirmation message transmitted
- *Retroactive* explicit authentication of earlier keys

[DGK06] Di Raimondo, Gennaro, Krawczyk. ACM CCS 2006. https://eprint.iacr.org/2006/280

Security subtleties: downgrade resilience

- Choice of cryptographic algorithms not authenticated at the time the client sends its first application flow
  - MITM can't trick client into using undesirable algorithm
  - But MITM can trick them into temporarily using suboptimal algorithm

- Formally model 3 levels of downgrade-resilience:
  1. Full downgrade resilience
  2. No downgrade resilience to unsupported algorithms
  3. No downgrade resilience

Security subtleties: forward secrecy

- **Weak forward secrecy 1:** adversary passive in the test stage
- **Weak forward secrecy 2:** adversary passive in the test stage or never corrupted peer's long-term key
- **Forward secrecy:** adversary passive in the test stage or didn't corrupt peer's long-term key before acceptance

- Can make detailed forward secrecy statements, such as:
  - Stage 1 and 2 keys are wfs1 when accepted, retroactive fs once stage 6 accepts

Certificate lifecycle management for KEM public keys

Starting to be discussed on IETF LAMPS mailing list and part of re-charter [1,2]

Proof of possession: How does requester prove possession of corresponding secret keys?

- Not really addressed in practice, since RSA and DL/ECDL keys can be used for both signing and encryption/KEX
- Can't sign like in a Certificate Signing Request (CSR)
- Could do interactive challenge-response protocol (or just run KEMTLS), but need online verification (RFC 4210 Sect. 5.2.8.3)
- Send cert to requestor encrypted under key in the certificate (RFC 4210 Sect. 5.2.8.2) – but maybe broken by Certificate Transparency?
- Zero-knowledge proof of knowledge?

Thanks to Mike Ounsworth (Entrust Datacard) for raising some of these issues.

[1] https://mailarchive.ietf.org/arch/msg/spasm/FCCZv3Xi3rkbZyZWQnnMQM0EFYY/
[2] https://mailarchive.ietf.org/arch/msg/spasm/9tukY1yTOzuNE8yHhuBLxzQWAko/

Certificate lifecycle management for KEM public keys

Revocation: How can certificate owner authorize a revocation request?

- Put a (hash of a) signature public key in the cert which can be used to revoke the cert?
  - Possibly could simplify to just revealing a hash preimage

Conclusions on KEMTLS

- Summary of protocol design: implicit authentication via KEMs

- Saves bytes on the wire and server CPU cycles
- Preserves client request after 1-RTT
- Caching intermediate CA certs brings even greater benefits

- Protocol design is simple to implement, provably secure
- Also have a variant supporting client authentication
- Working with Cloudflare to test within their infrastructure

Open Quantum Safe update
and
Post-quantum TLS without handshake signatures

Douglas Stebila UNIVERSITY OF WATERLOO

KEMTLS

Implicitly authenticated TLS without handshake signatures using KEMs

https://eprint.iacr.org/2020/534
https://github.com/thomwiggers/kemtls-experiment/

Open Quantum Safe project

Open-source software for prototyping and experimenting with PQ crypto, including in TLS

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

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