Integrating post-quantum cryptography into protocols: the case of TLS

Douglas Stebila

UNIVERSITY OF WATERLOO

NSERC
CRSNG

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

Post-Quantum Networks Workshop (PQNet) • 2021-09-29

UNIVERSITY OF
WATERLOO

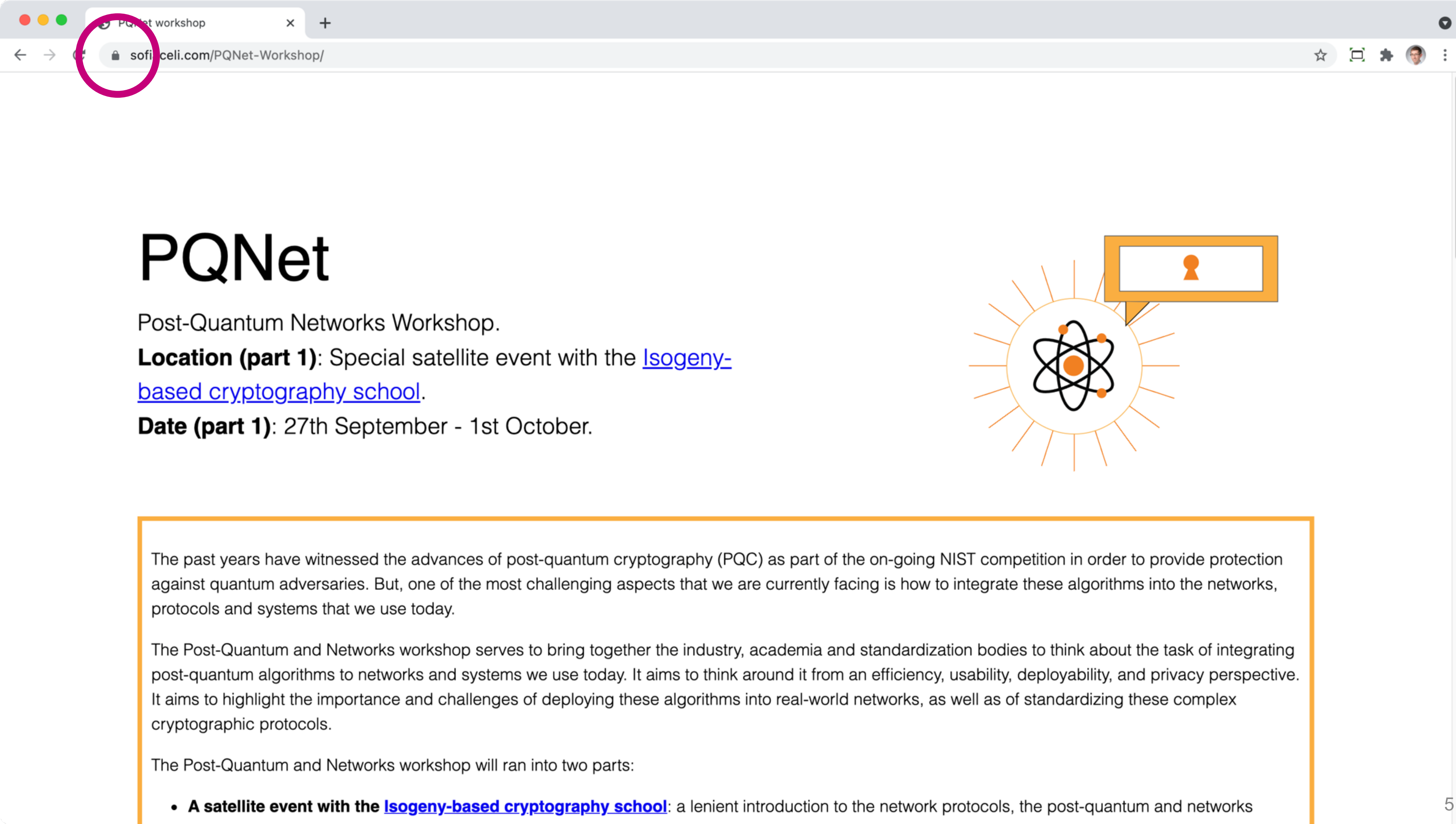
IQC Institute for Quantum Computing

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
- Elliptic curves: David Jao, Alfred Menezes, (Scott Vanstone)
- More cryptography: Sergey Gorbunov, Mohammad Hajiabadi, Doug Stinson
- Privacy-enhancing technologies: Ian Goldberg
- Quantum cryptanalysis: Michele Mosca
- Quantum cryptography: Norbert Lütkenhaus, Thomas Jennewein, Debbie Leung
- Even more cryptography and security: Gord Agnew, Vijay Ganesh, Guang Gong, Sergey Gorbunov, Anwar Hasan, Florian Kerschbaum

Background



PQNet

Post-Quantum Networks Workshop.

Location (part 1): Special satellite event with the Isogeny-based cryptography school.

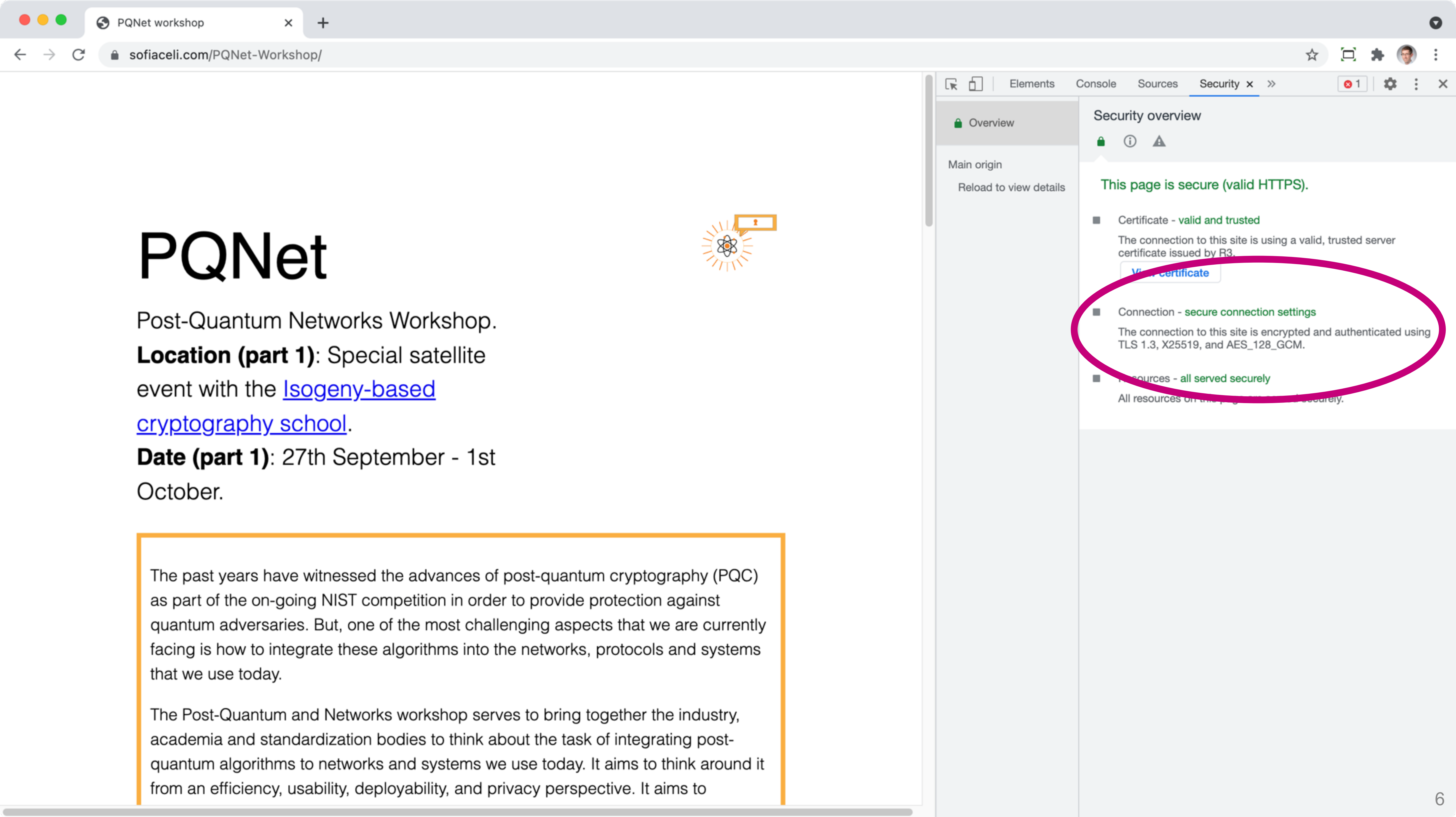
Date (part 1): 27th September - 1st October.

The past years have witnessed the advances of post-quantum cryptography (PQC) as part of the on-going NIST competition in order to provide protection against quantum adversaries. But, one of the most challenging aspects that we are currently facing is how to integrate these algorithms into the networks, protocols and systems that we use today.

The Post-Quantum and Networks workshop serves to bring together the industry, academia and standardization bodies to think about the task of integrating post-quantum algorithms to networks and systems we use today. It aims to think around it from an efficiency, usability, deployability, and privacy perspective. It aims to highlight the importance and challenges of deploying these algorithms into real-world networks, as well as of standardizing these complex cryptographic protocols.

The Post-Quantum and Networks workshop will ran into two parts:

- **A satellite event with the Isogeny-based cryptography school**: a lenient introduction to the network protocols, the post-quantum and networks



PQNet

Post-Quantum Networks Workshop.

Location (part 1): Special satellite event with the Isogeny-based cryptography school.

Date (part 1): 27th September - 1st October.

The past years have witnessed the advances of post-quantum cryptography (PQC) as part of the on-going NIST competition in order to provide protection against quantum adversaries. But, one of the most challenging aspects that we are currently facing is how to integrate these algorithms into the networks, protocols and systems that we use today.

The Post-Quantum and Networks workshop serves to bring together the industry, academia and standardization bodies to think about the task of integrating post-quantum algorithms to networks and systems we use today. It aims to think around it from an efficiency, usability, deployability, and privacy perspective. It aims to

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 R3.

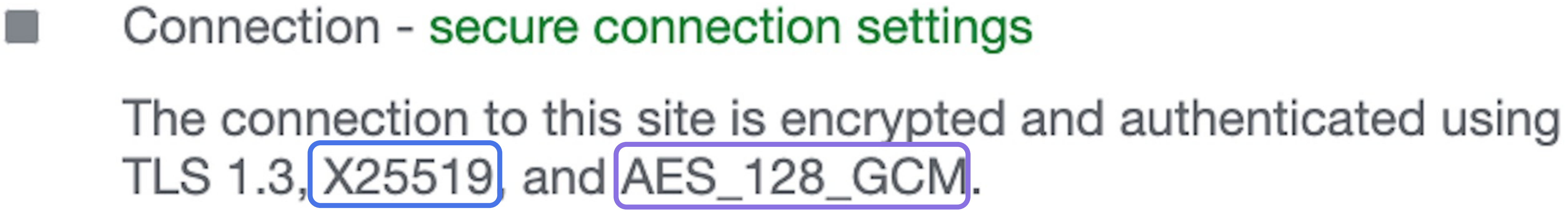
- Connection - secure connection settings

  The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM.

- sources - all served securely

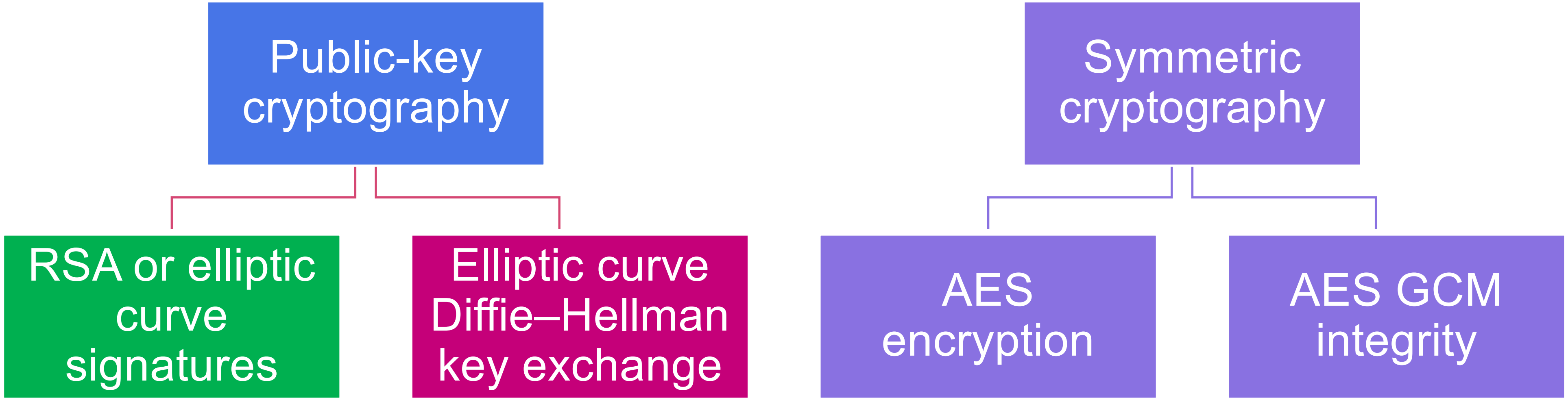
  All resources securely.

Cryptographic building blocks



Connection - secure connection settings

The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM.



Public-key cryptography

- RSA or elliptic curve signatures
- Elliptic curve Diffie–Hellman key exchange

Symmetric cryptography

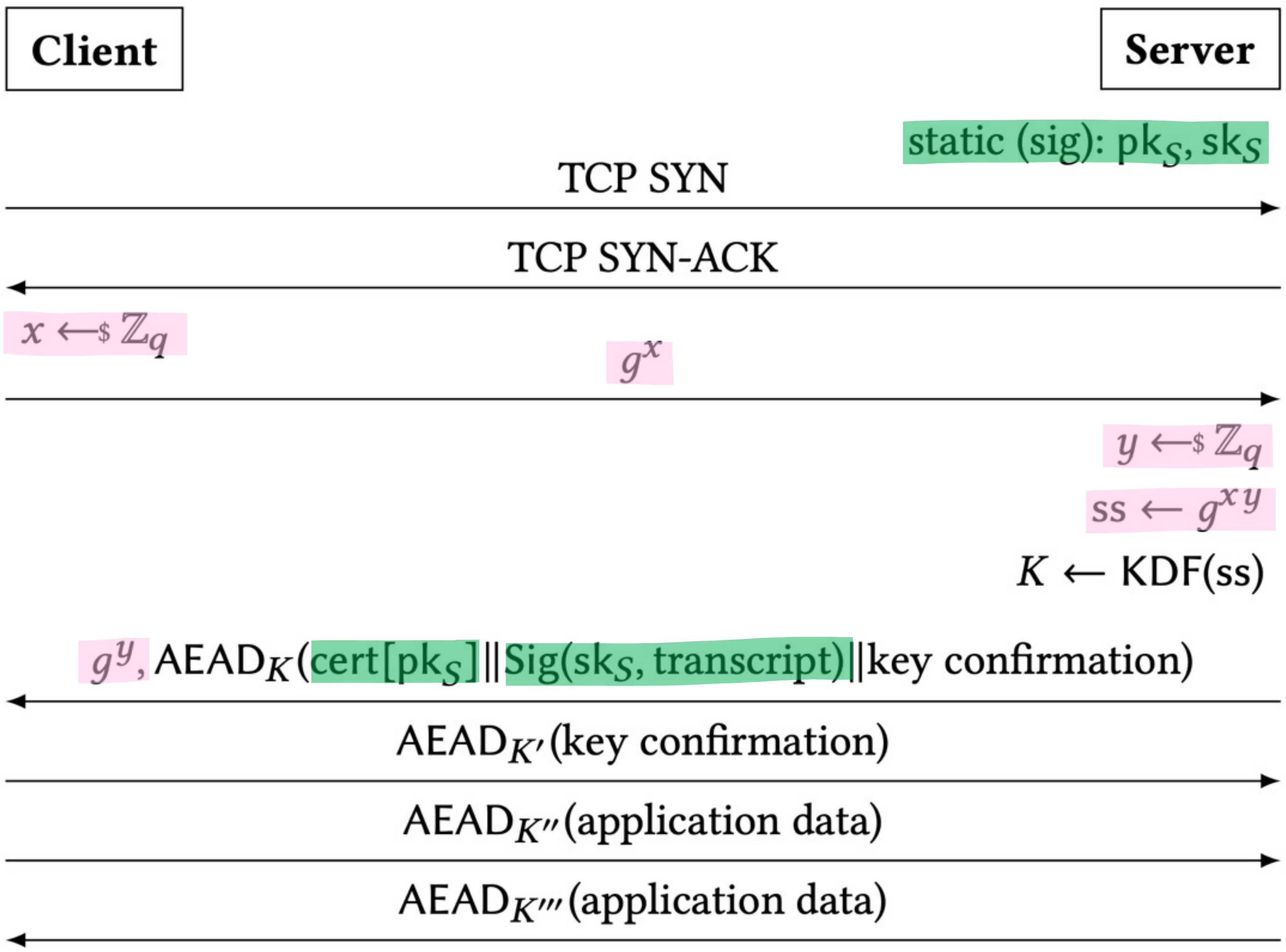
- AES encryption
- AES GCM integrity

TLS 1.3 handshake

Diffie-Hellman key exchange

Digital signature

Signed Diffie–Hellman



Client

Server

static (sig): $\mathrm{pk}_S, \mathrm{sk}_S$

TCP SYN →

← TCP SYN-ACK

$x \leftarrow_{\$} \mathbb{Z}_q$

g^x →

$y \leftarrow_{\$} \mathbb{Z}_q$

$\mathrm{ss} \leftarrow g^{xy}$

$K \leftarrow \mathrm{KDF}(\mathrm{ss})$

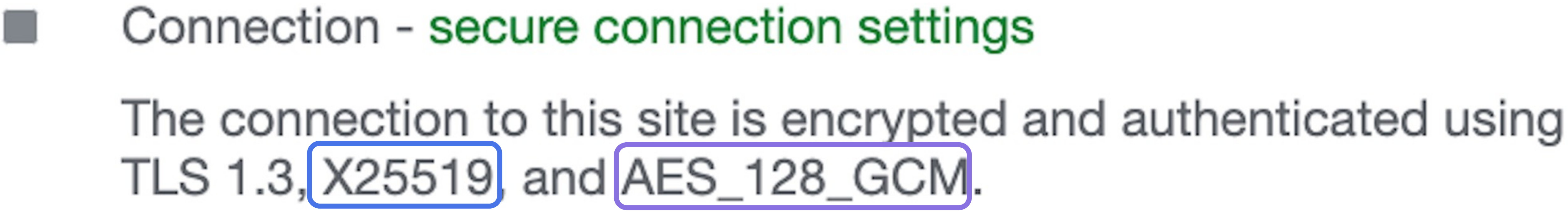
← $g^y, \mathrm{AEAD}_K(\mathrm{cert}[\mathrm{pk}_S] \| \mathrm{Sig}(\mathrm{sk}_S, \mathrm{transcript}) \| \text{key confirmation})$

$\mathrm{AEAD}_{K'}(\text{key confirmation})$ →

$\mathrm{AEAD}_{K''}(\text{application data})$ →

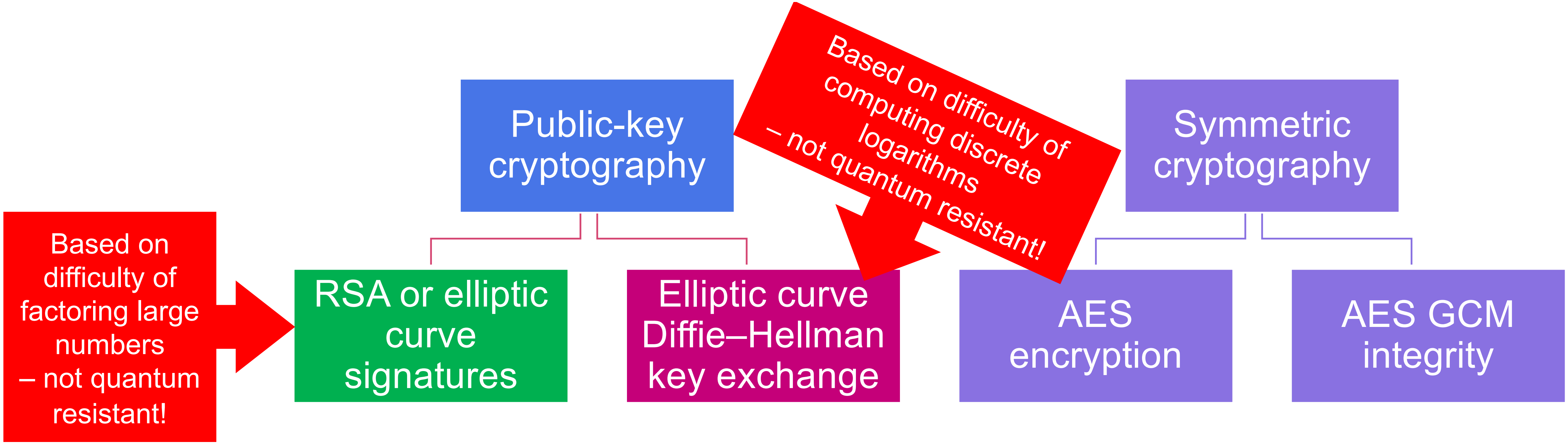
← $\mathrm{AEAD}_{K'''}(\text{application data})$

Cryptographic building blocks



■ Connection - secure connection settings

The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM.



Public-key cryptography

RSA or elliptic curve signatures

Elliptic curve Diffie–Hellman key exchange

Symmetric cryptography

AES encryption

AES GCM integrity

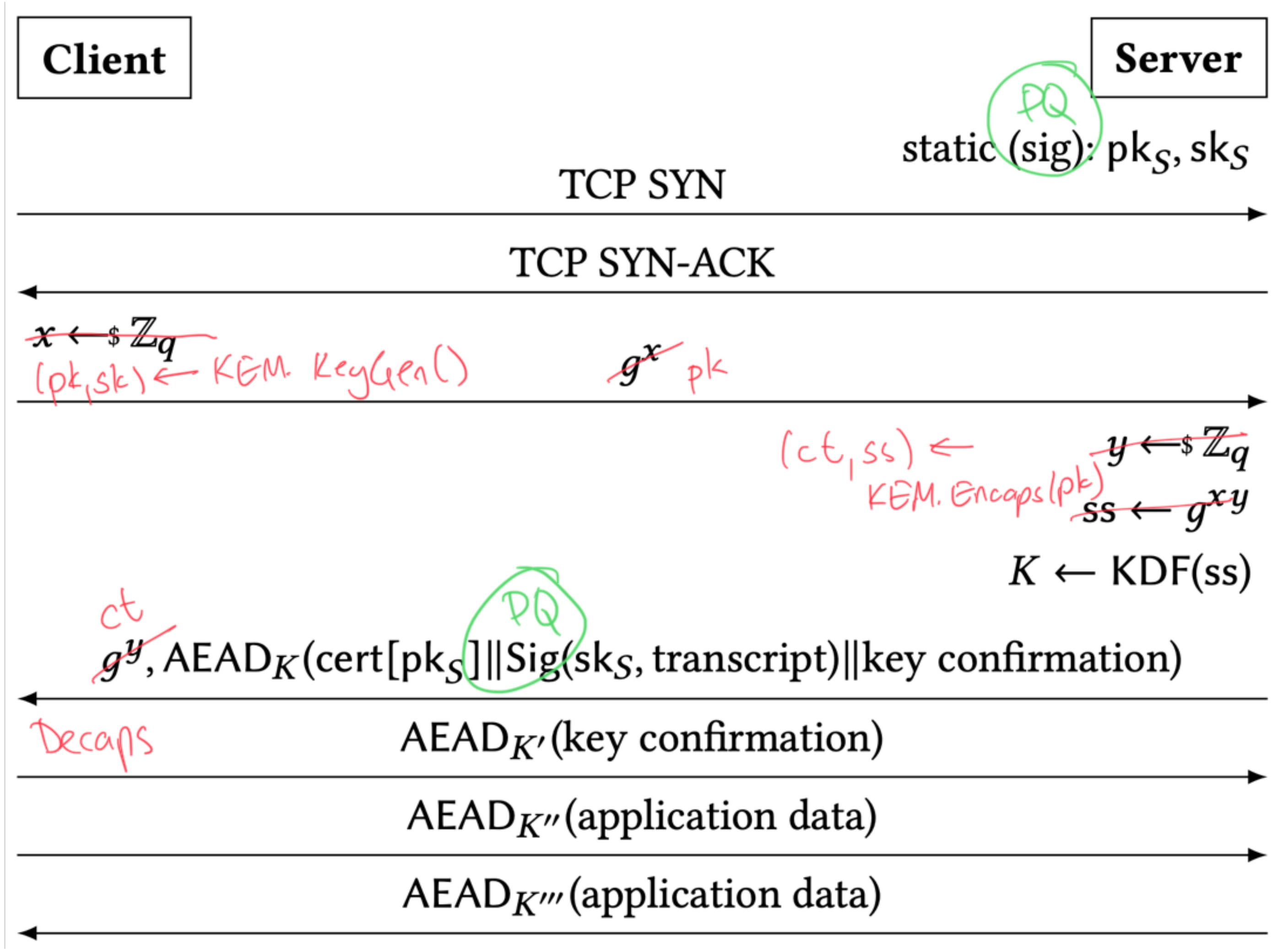
Based on difficulty of factoring large numbers – not quantum resistant!

Based on difficulty of computing discrete logarithms – not quantum resistant!

TLS 1.3 handshake

~~Signed Diffie–Hellman~~

Post-Quantum!!!



Client

Server

PQ

static (sig): pk_S, sk_S

TCP SYN

TCP SYN-ACK

~~$x \leftarrow_{\$} \mathbb{Z}_q$~~

$(pk, sk) \leftarrow$ KEM. KeyGen()

~~g^x~~ pk

$(ct, ss) \leftarrow$ KEM. Encaps(pk)

~~$y \leftarrow_{\$} \mathbb{Z}_q$~~

~~$ss \leftarrow g^{xy}$~~

$K \leftarrow \mathrm{KDF}(ss)$

ct

PQ

~~g^y~~, $\mathrm{AEAD}_K(\mathrm{cert}[pk_S] \| \mathrm{Sig}(sk_S, \mathrm{transcript}) \| \text{key confirmation})$

Decaps

$\mathrm{AEAD}_{K'}(\text{key confirmation})$

$\mathrm{AEAD}_{K''}(\text{application data})$

$\mathrm{AEAD}_{K'''}(\text{application data})$

Outline

Benchmarking

Hybrid standardization

New protocol designs (KEMTLS)

Benchmarking post-quantum crypto in TLS

Christian Paquin, Douglas Stebila, Goutam Tamvada.
PQCrypto 2020.
https://eprint.iacr.org/2019/1447

Goal

- Measure effect of **network latency** and **packet loss rate** on handshake completion time for post-quantum connections of various sizes

- Out of scope:
  - Effect of different CPU speeds from client or server
  - Effect of different post-quantum algorithms on server throughput

Related work

- [BCNS15] and [BCD+16] measured the impact of their post-quantum key-exchange schemes on the performance of an Apache server running TLS 1.2
- [KS19] and [SKD20] measured the impact of post-quantum signatures in TLS 1.3 on handshake time (with various server distances), and handshake failure rate and throughput for a heavily loaded server

[BCNS15] Bos, Costello, Naehrig, Stebila. IEEE S&P 2015. https://eprint.iacr.org/2014/599
[BCD+16] Bos, Costello, Ducas, Mironov, Naehrig, Nikolaenko, Raghunathan, Stebila. ACM CCS 2016. https://eprint.iacr.org/2016/659
[KS19] Kampanakis, Sikeriis. https://eprint.iacr.org/2019/1276
[SKD20] Sikeridis, Kampanaokis, Devetsikiotis. NDSS 2020. https://eprint.iacr.org/2020/071

Related work: Internet-wide experiments

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

Langley, 2016. https://www.imperialviolet.org/2016/11/28/cecpq1.html
Langley, 2018. https://www.imperialviolet.org/2018/12/12/cecpq2.html
Sullivan, Kwiatkowski, Langley, Levin, Mislove, Valenta. NIST 2nd PQC Standardization Conference 2019. https://csrc.nist.gov/Presentations/2019/measuring-tls-key-exchange-with-post-quantum-kem

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

(Inspired by NetMirage and Mininet)

Emulate the network!

+ more control over experiment parameters

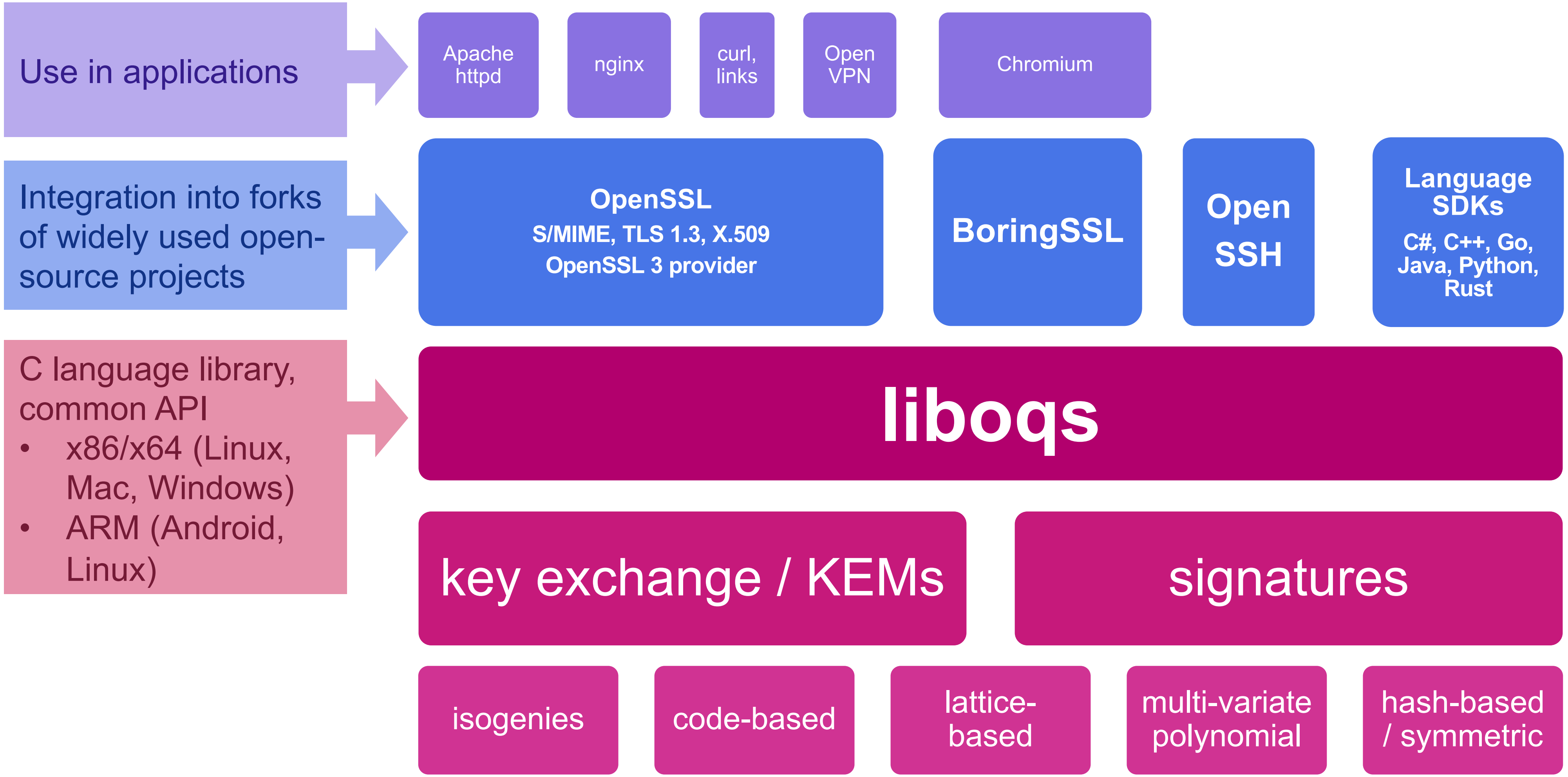
+ easier to isolate effects of network characteristics

– loss in realism

Network emulation in Linux

- Kernel can create **network namespaces**: Independent copies of the kernel’s network stack
- **Virtual ethernet devices** can be created to connect the two namespaces
- **netem (network emulation)** kernel module
  - Can instruct kernel to apply a specified delay to packets
  - Can instruct kernel to drop packets with a specified probability

Open Quantum Safe Project



Use in applications

- Apache httpd
- nginx
- curl, links
- Open VPN
- Chromium

Integration into forks of widely used open-source projects

- **OpenSSL**
  - **S/MIME, TLS 1.3, X.509**
  - **OpenSSL 3 provider**
- **BoringSSL**
- **Open SSH**
- **Language SDKs**
  - **C#, C++, Go, Java, Python, Rust**

C language library, common API

- x86/x64 (Linux, Mac, Windows)
- ARM (Android, Linux)

liboqs

- key exchange / KEMs
- signatures

- isogenies
- code-based
- lattice-based
- multi-variate polynomial
- hash-based / symmetric

Industry partners:

- Amazon Web Services
- evolutionQ
- IBM Research
- Microsoft Research

Additional contributors:

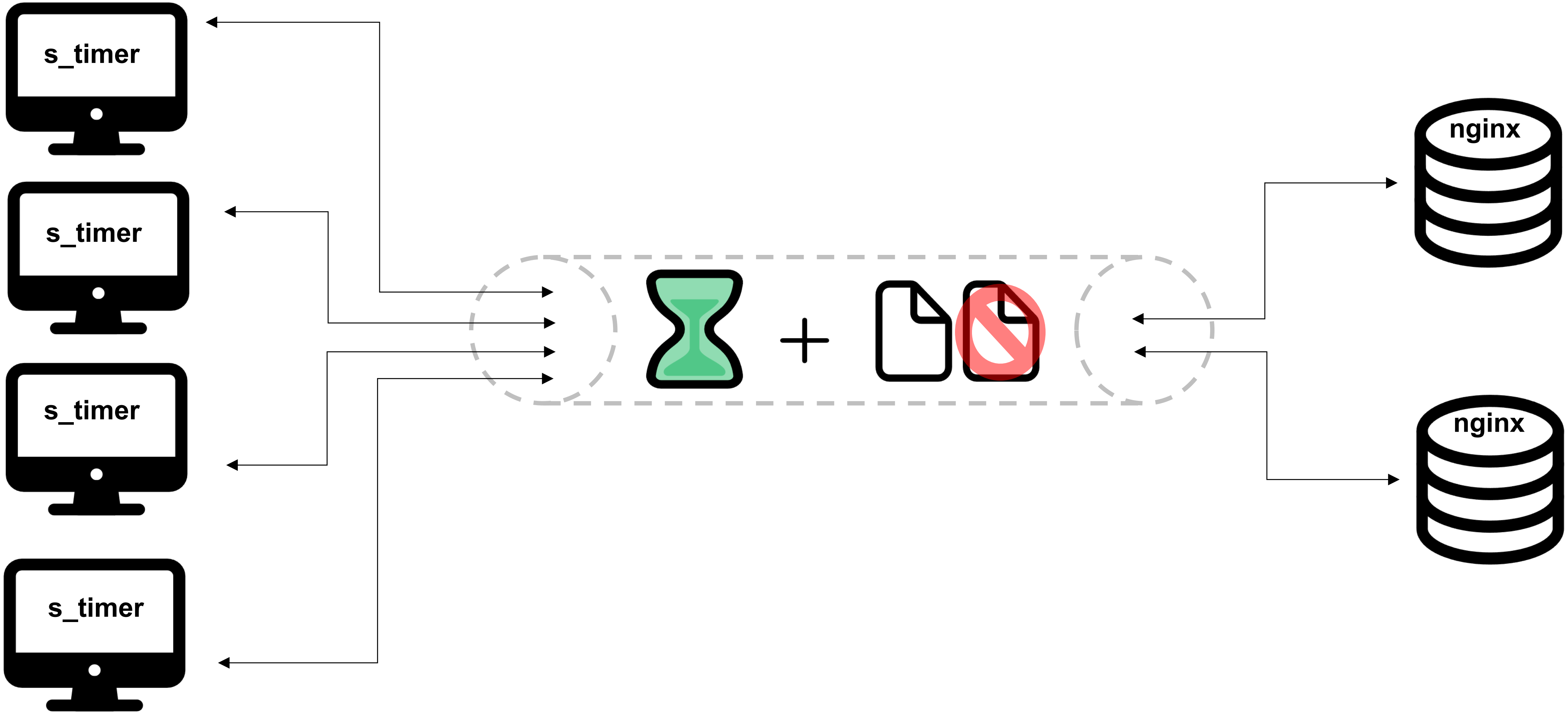
- Cisco
- Senetas
- PQClean project
- Individuals

Financial support:

- AWS
- Canadian Centre for Cyber Security
- NSERC
- Unitary Fund

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

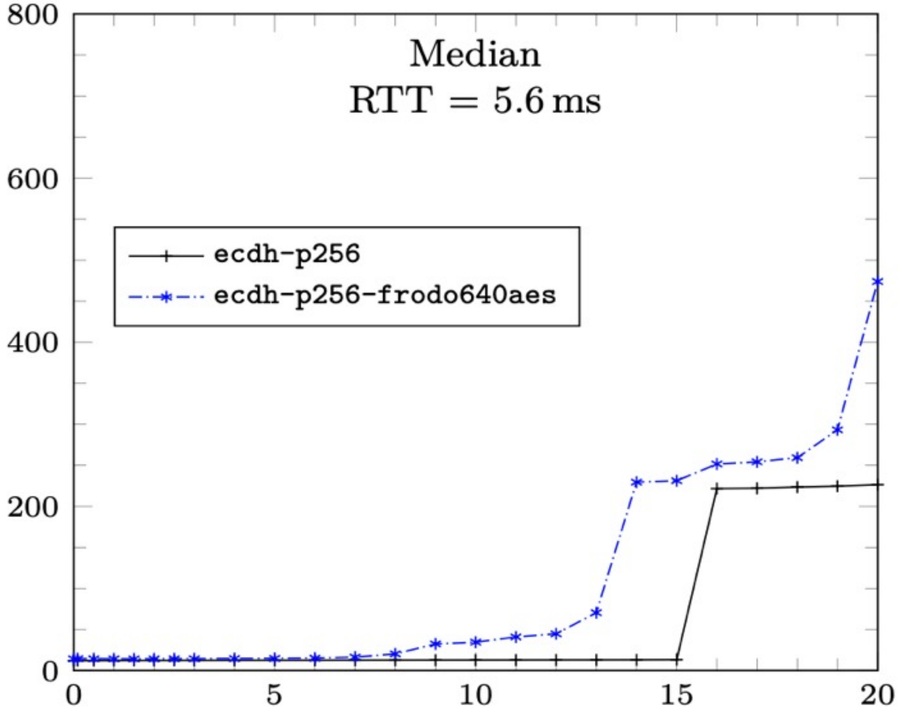
Network emulation experiment (contd.)



Icons from https://ionicons.com/

Key exchange in TLS 1.3

median

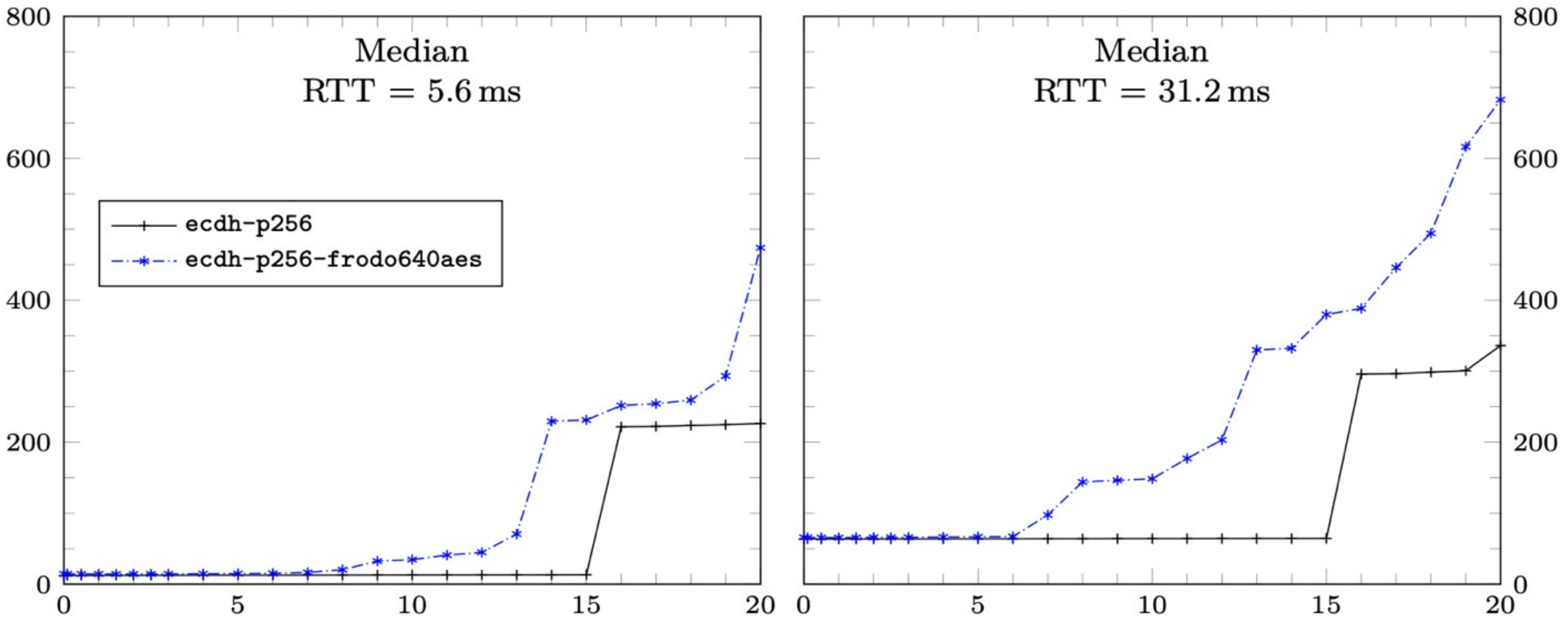


handshake completion time (ms)

packet loss rate %

Key exchange in TLS 1.3

median

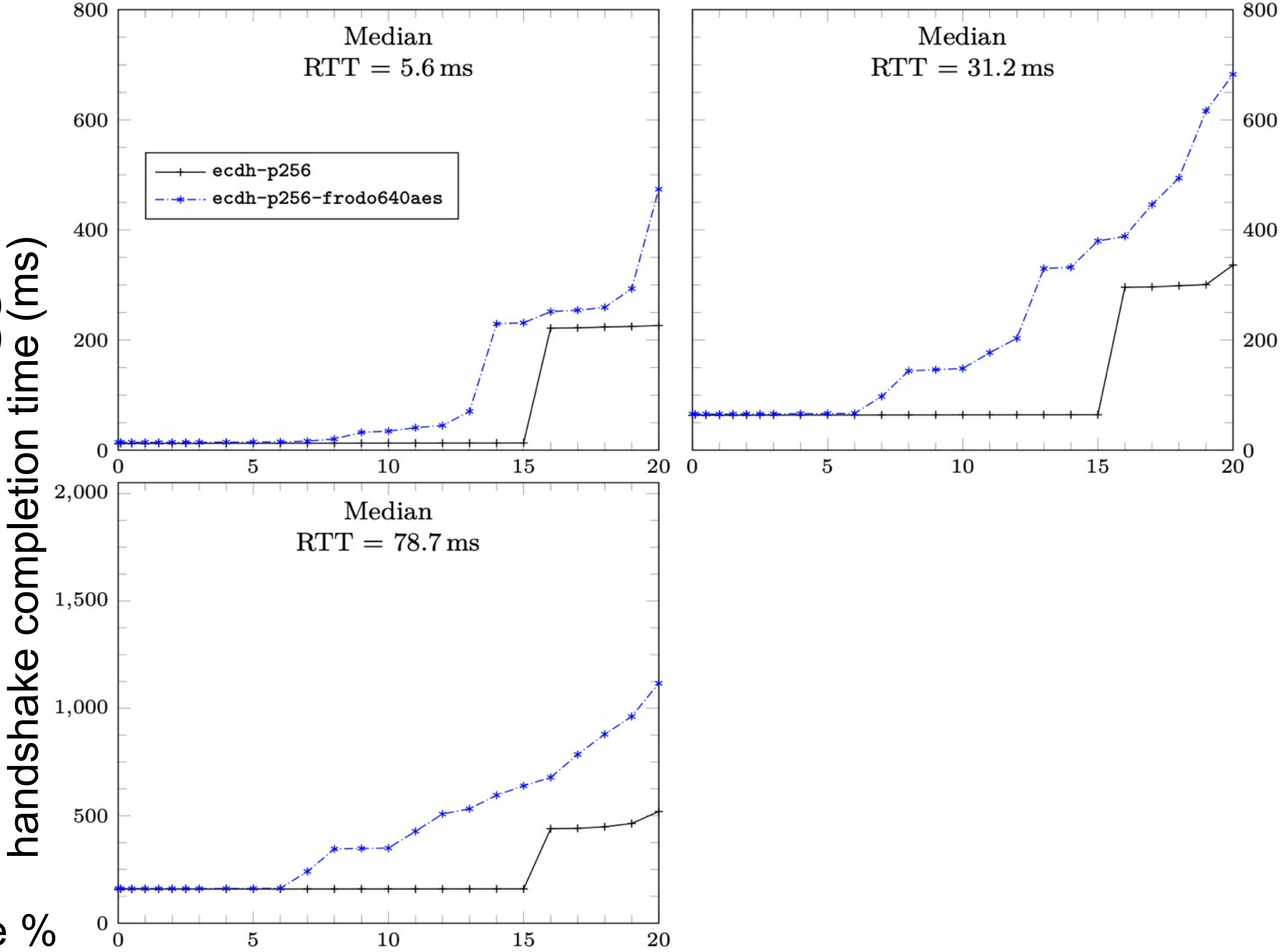


handshake completion time (ms)

packet loss rate %

Key exchange in TLS 1.3

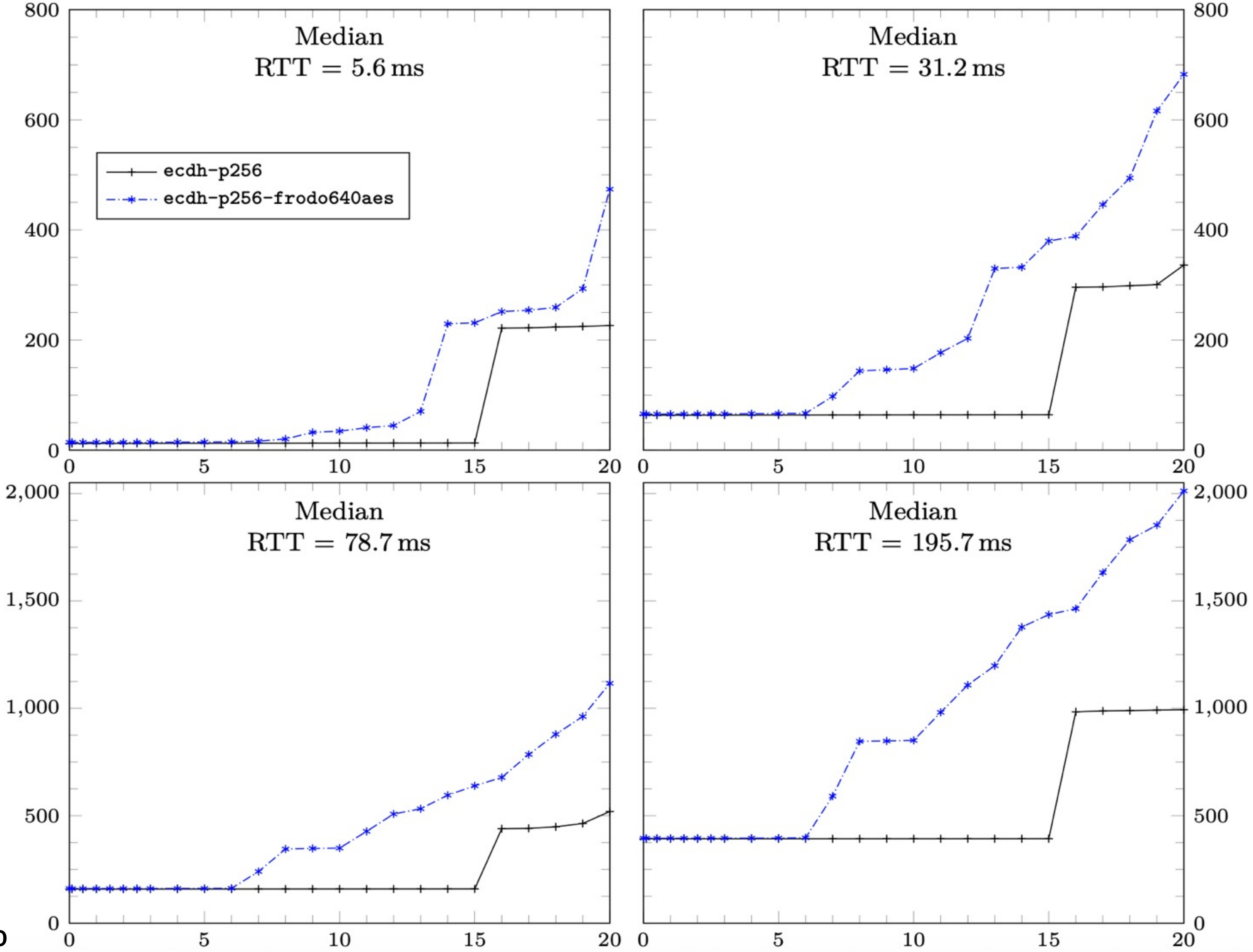
median



Key exchange in TLS 1.3

median

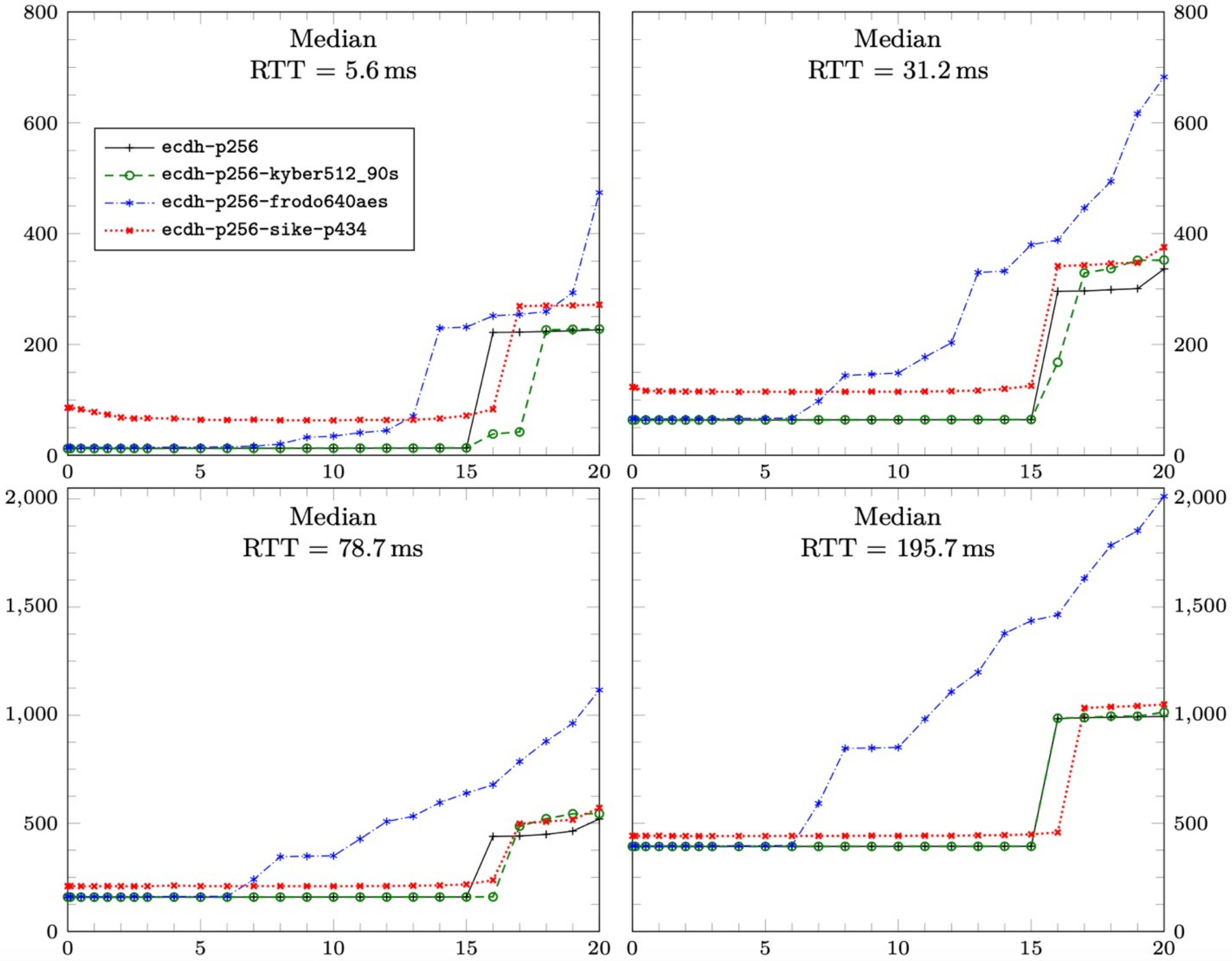
handshake completion time (ms)



packet loss rate %

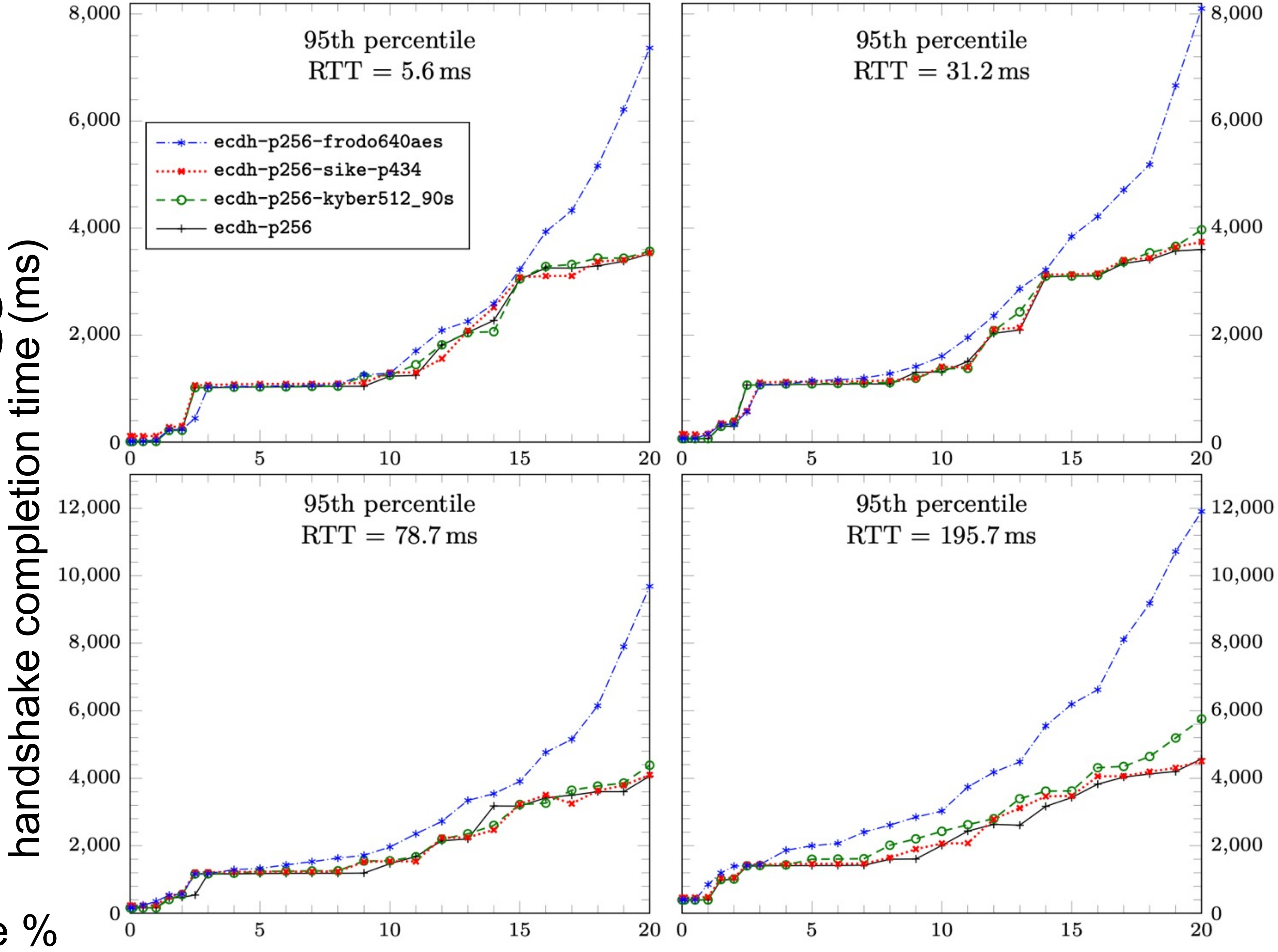
Key exchange in TLS 1.3

median



Key exchange in TLS 1.3

95th percentile



Conclusions

- On **fast, reliable network links**, the cost of public key cryptography dominates the median TLS establishment time, but does not substantially affect the 95th percentile establishment time

- On **unreliable network links** (packet loss rates >= 3%), communication sizes come to govern handshake completion time

- As application data sizes grow, the relative cost of TLS handshake establishment diminishes compared to application data transmission

Hybrid key exchange in TLS 1.3

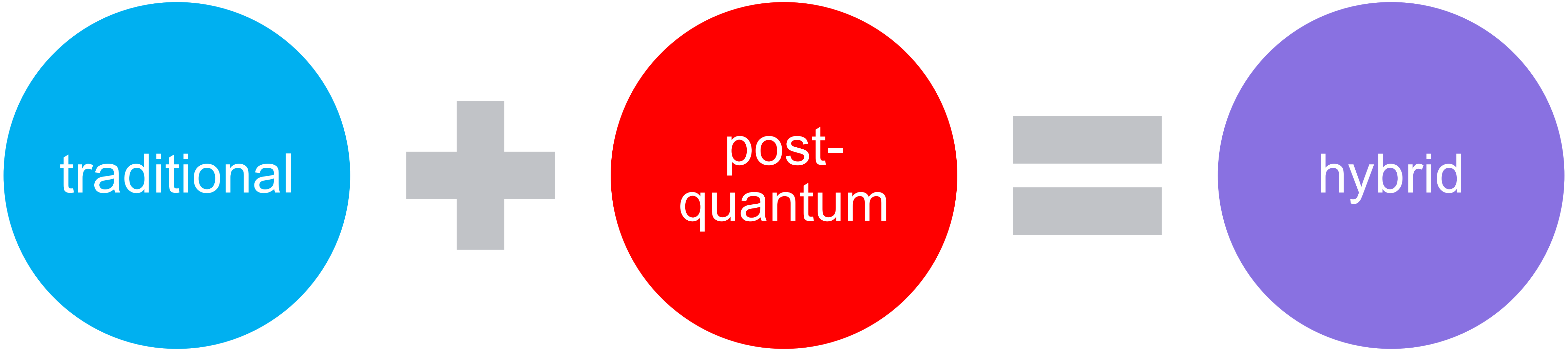
draft-ietf-tls-hybrid-design-03

Douglas Stebila, Scott Fluhrer, Shay Gueron

https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-03

Cautious "hybrid" approach

- Some proposed post-quantum solutions could be broken
- **Hybrid approach**: use traditional and post-quantum simultaneously to reduce risk during transition



traditional

post-quantum

hybrid

Hybrid approach

- **Permit simultaneous use of traditional and post-quantum key exchange**

- Enable early adopters to get post-quantum security without discarding security of existing algorithms
- Why do this?
  - Uncertainty re: newer cryptographic assumptions
  - Temporary need to keep traditional algorithms for e.g. FIPS certification

Goals

Define data structures for negotiation, communication, and shared secret calculation for hybrid* key exchange

Non-goals

- Hybrid/composite certificates or digital signatures
- Selecting which post-quantum algorithms to use in TLS

* Some people use the word “composite” instead of “hybrid”.

Mechanism

Idea: Each desired combination of traditional + post-quantum algorithm will be a new (opaque) key exchange “group”

- **Negotiation**: new named groups for each desired combination will need to be standardized
- **Key shares**: concatenate key shares for each constituent algorithm
- **Shared secret calculation**: concatenate shared secrets for each constituent algorithm and use as input to key schedule

Other design options

Negotiation

- 2 vs ≥2 algorithms
- Extension for representing algorithm options and constraints

Key shares

- Separately list key shares for each algorithm
- Use extensions for extra key shares

Shared secret

- Apply KDF before inserting into key schedule
- XOR shares
- Insert into different parts of TLS key schedule

See Appendix A of draft for related work and Appendix B for detailed discussion of other design options.

Securely combining keying material

Is it okay to use concatenation?

$ss = k_1 || k_2$

$ss = H(k_1 || k_2)$

Note concatenation is the primary hybrid method approved by NIST.

- Assume at least one of k_1 or k_2 is indistinguishable from random.

- If H is a random oracle, then ss is indistinguishable from random.
- If k_1 and k_2 are fixed length and H is a dual PRF in either half of its input, then ss is indistinguishable from random.

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf#page=10

Securely combining keying material

Is it okay to use concatenation?

$ss = k_1 \| k_2$

$ss = H(k_1 \| k_2)$

- Aviram et al: If H is not collision resistant, then concatenating secrets may be dangerous.
  - Attack if k_1 is adversary-controlled and variable length, like APOP or CRIME attacks.
  - Applies to other parts of the TLS 1.3 key schedule.
  - Currently discussing impact and mitigation.

Aviram, Dowling, Komargodski, Paterson, Ronen, Yogev. Concatenating secrets may be dangerous, August 2021.
https://github.com/nimia/kdf_public

Composite certificates at the LAMPS working group

Led by Mike Ounsworth from Entrust Datacard
and Massimiliano Pala from CableLabs

(I’m not involved – just including here FYI)

LAMPS working group

- “Limited Additional Mechanisms for PKIX and SMIME”
  - PKIX: Public key infrastructure a.k.a. X.509 certificates
  - SMIME: Secure email (encrypted/signed)

Composite drafts at LAMPS

- LAMPS charter now includes milestones related to PQ
- Four drafts currently available:
  - draft-ounsworth-pq-composite-keys-00
  - draft-ounsworth-pq-explicit-composite-keys-00
  - draft-ounsworth-pq-composite-sigs-05
  - draft-ounsworth-pq-composite-encryption-00

https://datatracker.ietf.org/wg/lamps/about/

Composite OR versus Composite AND

- How is a credential with two public keys meant to be used?
  - Must both algorithms be used? (Composite AND)
  - Is either algorithm okay? (Composite OR)

New protocol designs: KEMTLS

Peter Schwabe, Douglas Stebila, Thom Wiggers
ACM CCS 2020. https://eprint.iacr.org/2020/534
ESORICS 2021. https://eprint.iacr.org/2021/779

Sofía Celi, Peter Schwabe, Douglas Stebila, Nick Sullivan, Thom Wiggers.
https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-00

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, …
- …

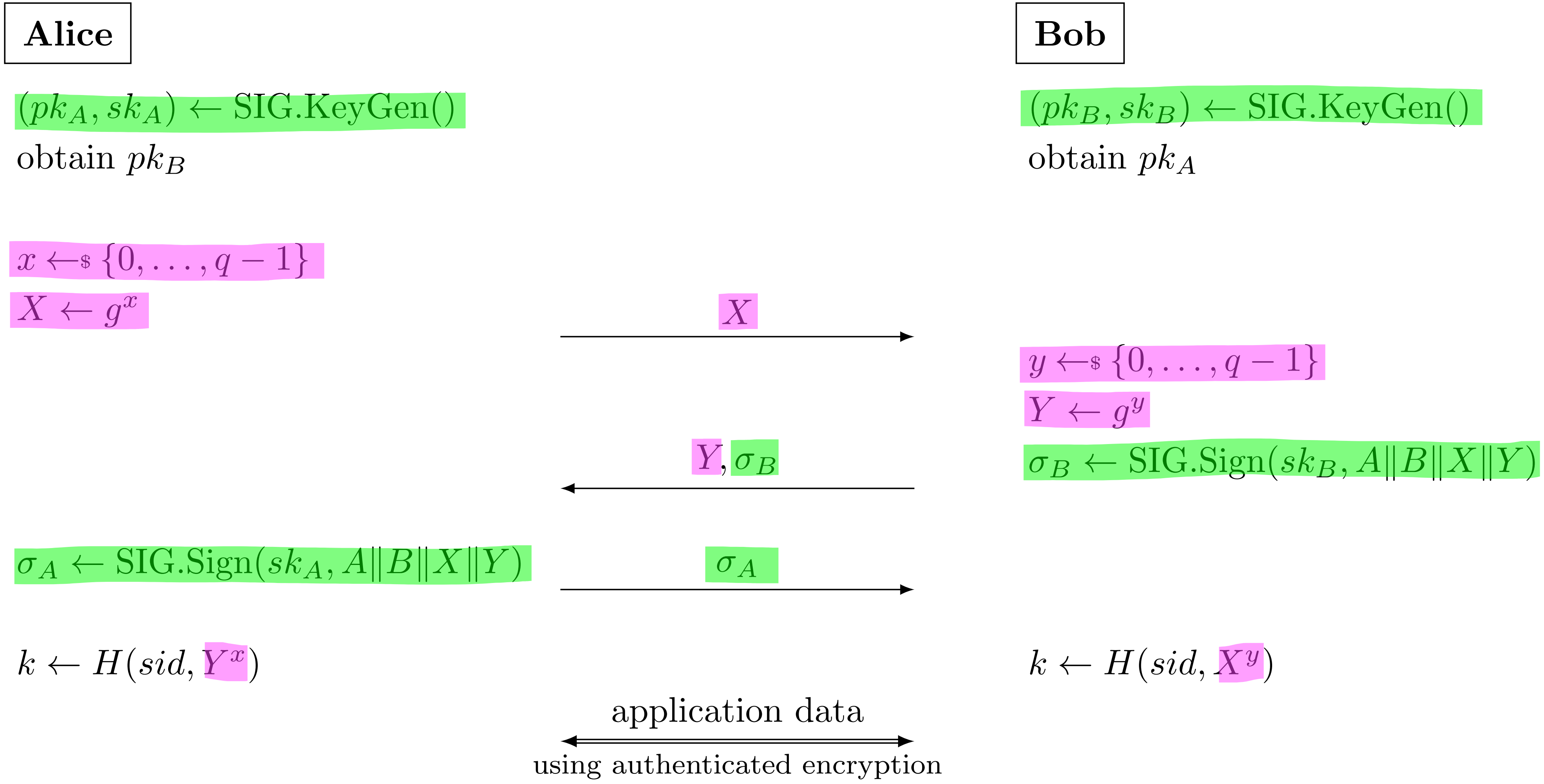
Explicit authentication

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

$x \leftarrow_{\$} \{0, \ldots, q-1\}$

$X \leftarrow g^x$

$\xrightarrow{X}$

$y \leftarrow_{\$} \{0, \ldots, q-1\}$

$Y \leftarrow g^y$

$\sigma_B \leftarrow \text{SIG.Sign}(sk_B, A \| B \| X \| Y)$

$\xleftarrow{Y, \sigma_B}$

$\sigma_A \leftarrow \text{SIG.Sign}(sk_A, A \| B \| X \| Y)$

$\xrightarrow{\sigma_A}$

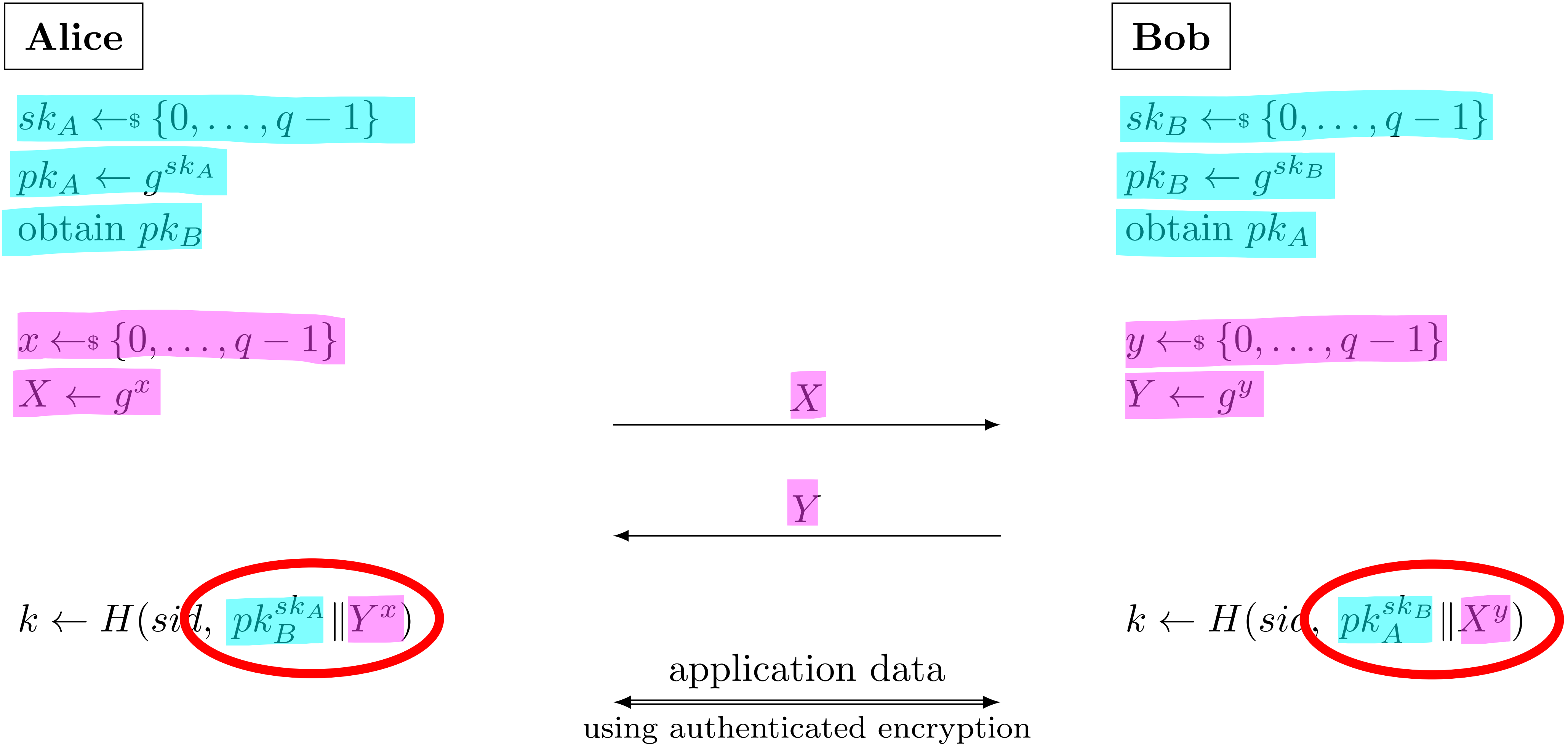
$k \leftarrow H(sid, Y^x)$ | $k \leftarrow H(sid, X^y)$

application data

$\Longleftrightarrow$

using authenticated encryption

Implicitly authenticated key exchange: Double-DH



Alice

$sk_A \leftarrow_\$ \{0, \ldots, q-1\}$

$pk_A \leftarrow g^{sk_A}$

obtain pk_B

$x \leftarrow_\$ \{0, \ldots, q-1\}$

$X \leftarrow g^x$

Bob

$sk_B \leftarrow_\$ \{0, \ldots, q-1\}$

$pk_B \leftarrow g^{sk_B}$

obtain pk_A

$y \leftarrow_\$ \{0, \ldots, q-1\}$

$Y \leftarrow g^y$

$\xrightarrow{X}$

$\xleftarrow{Y}$

$k \leftarrow H(sid, pk_B^{sk_A} \| Y^x)$

$k \leftarrow H(sid, pk_A^{sk_B} \| X^y)$

application data

$\Longleftrightarrow$

using authenticated encryption

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**
Rainbow	**Multi-variate**	**60,192**	**66**

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**
Rainbow	**Multi-variate**	**60,192**	**66**

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**
Classic McEliece	**Code-based**	**261,120**	**128**

Implicitly authenticated KEX is not new

In theory

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

In practice

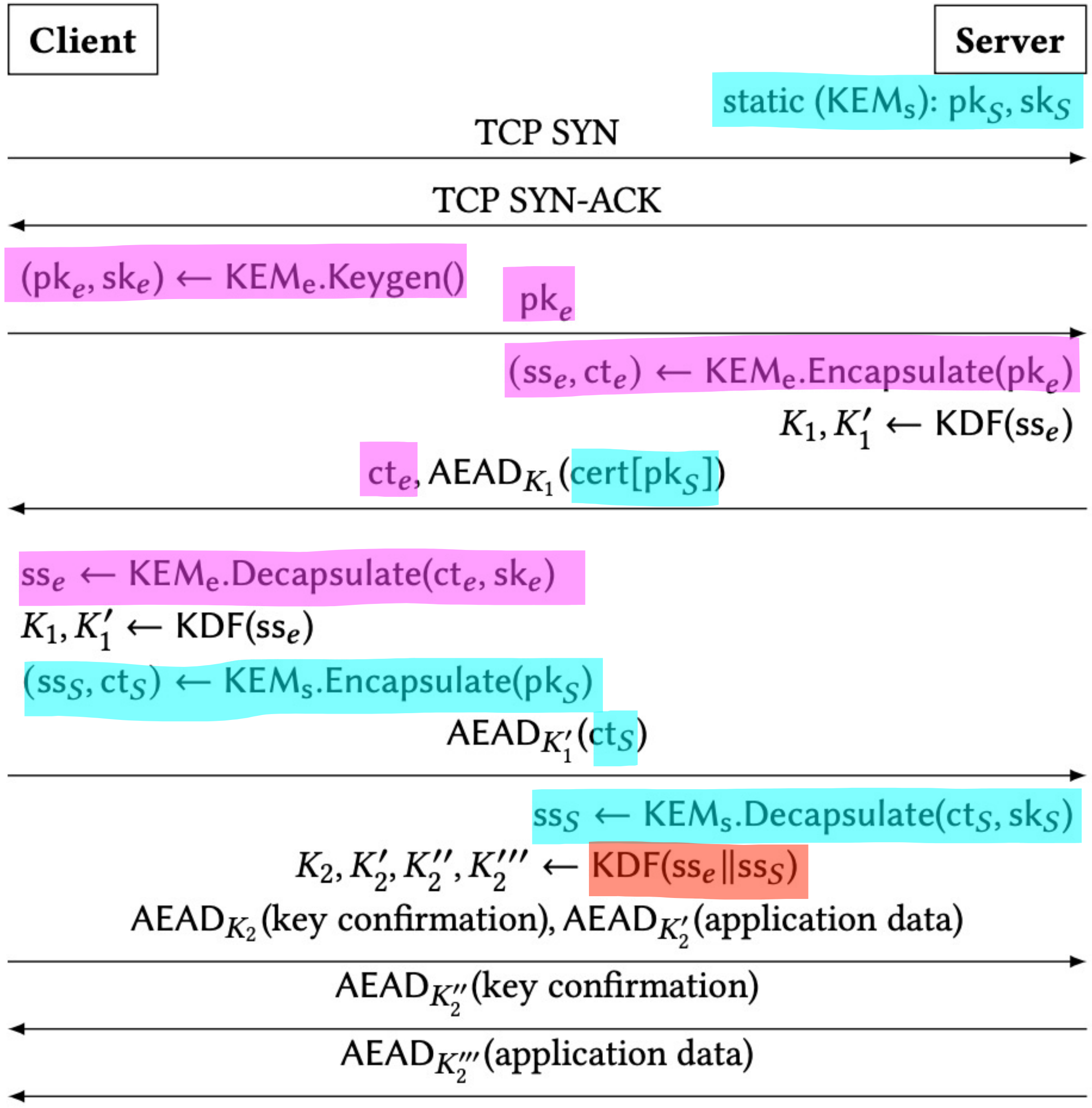
- RSA key transport in TLS ≤ 1.2
  - Lacks forward secrecy
- Signal, Noise, Wireguard
  - DH-based
  - Different protocol flows
- OPTLS
  - DH-based
  - Requires a non-interactive key exchange (NIKE)

KEMTLS handshake

KEM for ephemeral key exchange

KEM for server-to-client authenticated key exchange

Combine shared secrets



Client

Server

static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$

TCP SYN

TCP SYN-ACK

$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$

pk_e

$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$

$K_1, K_1' \leftarrow \mathsf{KDF}(\mathsf{ss}_e)$

$\mathsf{ct}_e, \mathsf{AEAD}_{K_1}(\mathsf{cert}[\mathsf{pk}_S])$

$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$

$K_1, K_1' \leftarrow \mathsf{KDF}(\mathsf{ss}_e)$

$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_s.Encapsulate}(\mathsf{pk}_S)$

$\mathsf{AEAD}_{K_1'}(\mathsf{ct}_S)$

$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$

$K_2, K_2', K_2'', K_2''' \leftarrow \mathsf{KDF}(\mathsf{ss}_e \| \mathsf{ss}_S)$

AEAD_{K_2}(key confirmation), $\mathsf{AEAD}_{K_2'}$(application data)

$\mathsf{AEAD}_{K_2''}$(key confirmation)

$\mathsf{AEAD}_{K_2'''}$(application data)

Algorithm choices

KEM for ephemeral key exchange

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

Signature scheme for intermediate CA

- Want small public key
  + small signature

KEM for authenticated key exchange

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

Signature scheme for root CA

- Want small signature

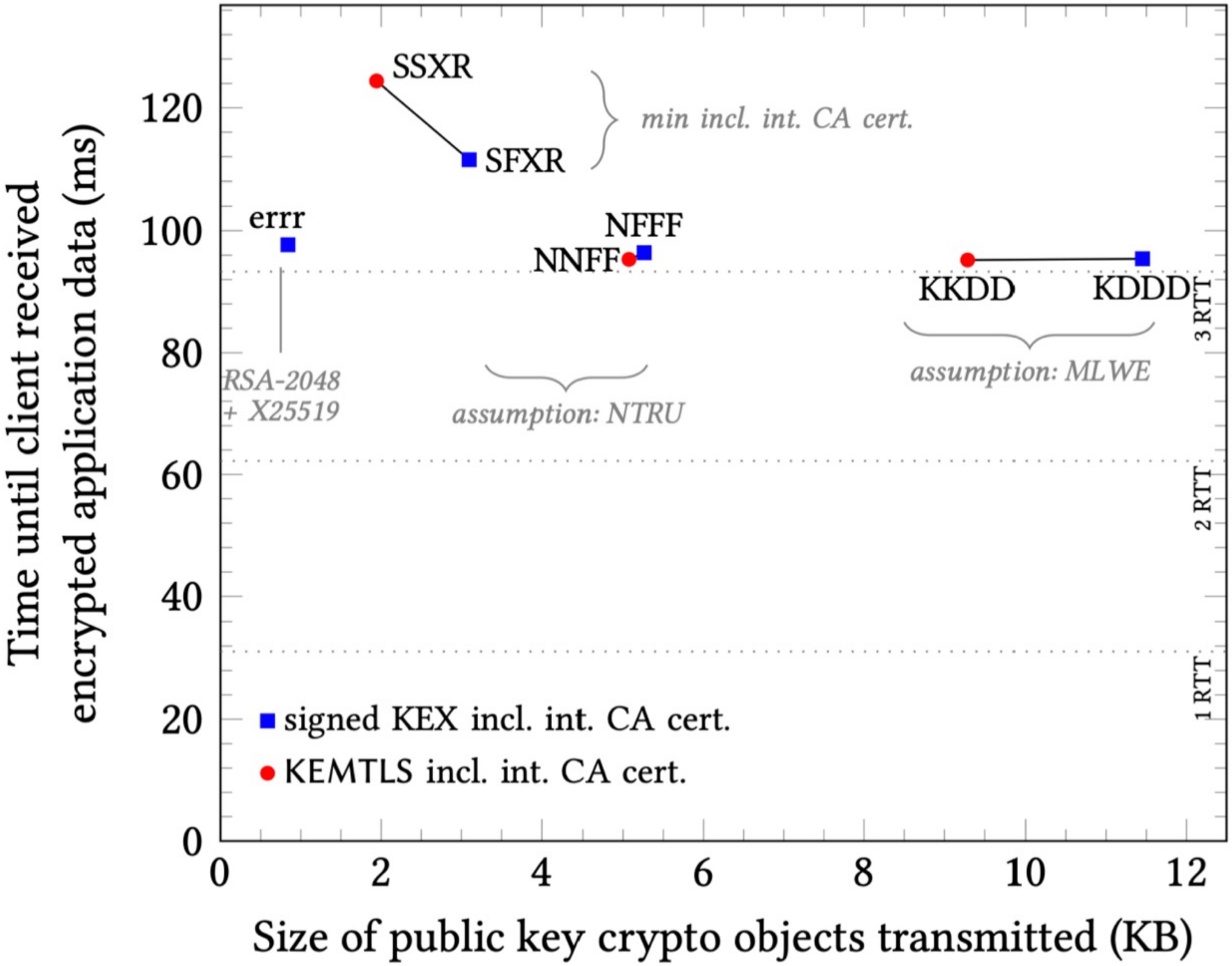
4 scenarios

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,
Kyber,
NTRU,
Rainbow,
rSA-2048,
SIKE,
XMSS’

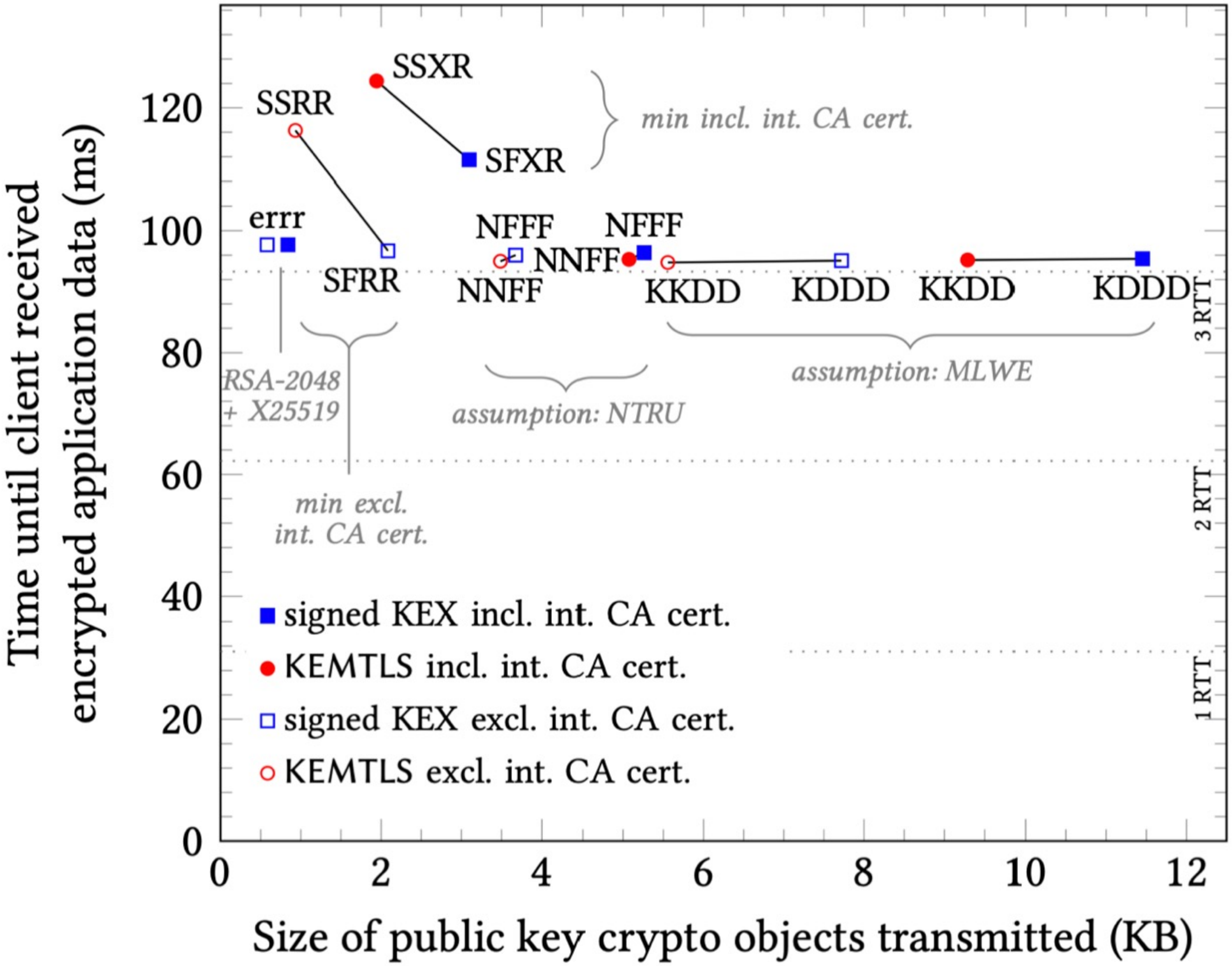


Rustls client/server with some AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

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,
Kyber,
NTRU,
Rainbow,
rSA-2048,
SIKE,
XMSS'



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

KEMTLS benefits

- 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

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

Does compromise of a party’s long-term key allow decryption of past sessions?

- **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

Variant: KEMTLS with client authentication

1. Client has a long-term KEM public key
2. Client transmits it encrypted under key derived from
    a) server long-term KEM key exchange
    b) ephemeral KEM key exchange

- Adds extra round trip

Variant: Pre-distributed public keys

What if server public keys are pre-distributed?

- Cached in a browser
- Pinned in mobile apps
- Embedded in IoT devices
- Out-of-band (e.g., DNS)
- TLS 1.3: RFC 7924

TLS 1.3 already supports pre-shared symmetric keys

- Harder(?) key management problem
- Different compromise model

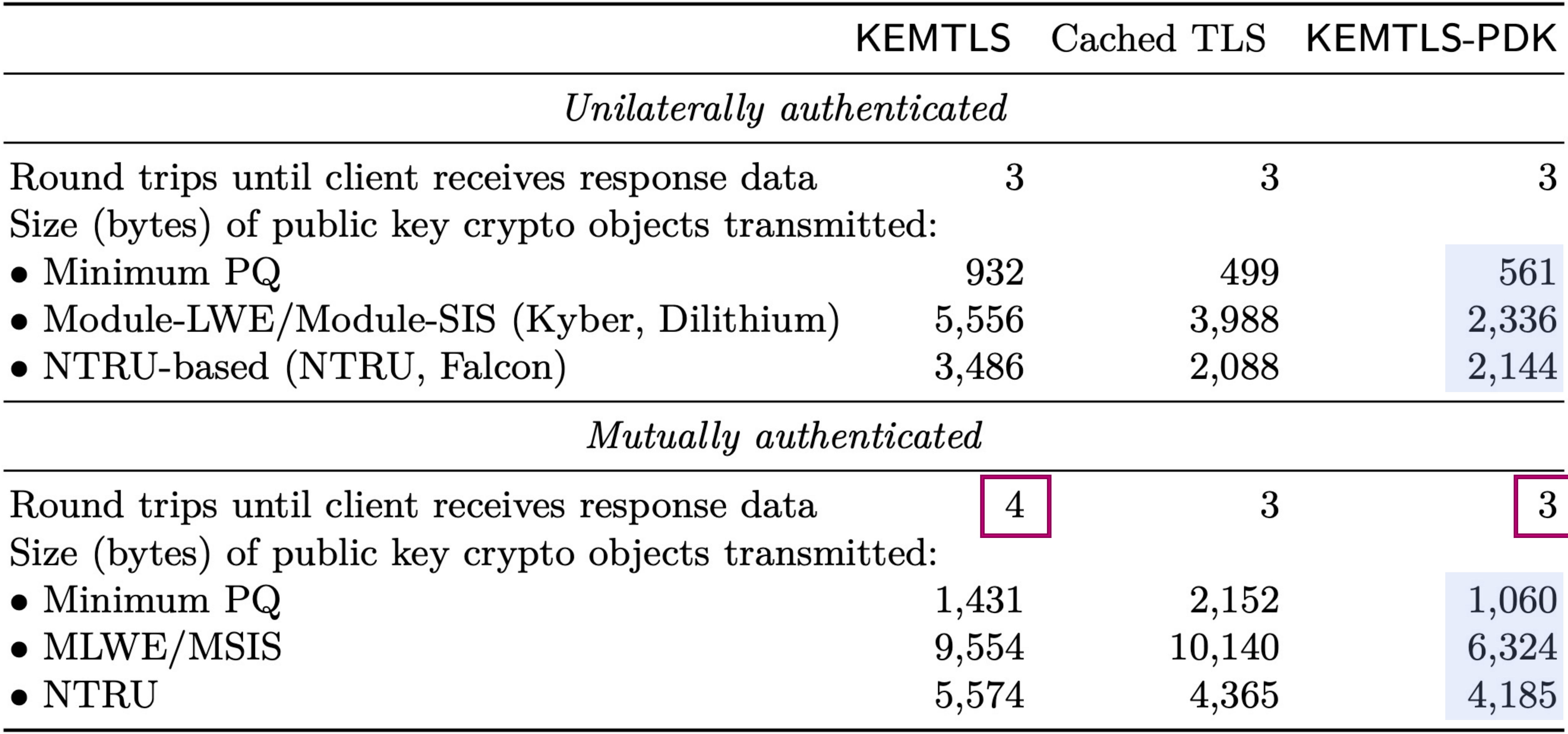
KEMTLS-PDK

- Alternate KEMTLS protocol flow when server certificates are known in advance

https://thomwiggers.nl/publication/kemtlspdk/

KEMTLS-PDK benefits

- Additional bandwidth savings
- Makes some PQ algorithms viable
  - Large public keys, small ciphertexts/signatures: Classic McEliece and Rainbow
- Client authentication 1 round-trip earlier if proactive
- Explicit server authentication 1 round-trip earlier
  - => better downgrade resilience



	KEMTLS	Cached TLS	KEMTLS-PDK
Unilaterally authenticated			
Round trips until client receives response data	3	3	3
Size (bytes) of public key crypto objects transmitted:			
• Minimum PQ	932	499	561
• Module-LWE/Module-SIS (Kyber, Dilithium)	5,556	3,988	2,336
• NTRU-based (NTRU, Falcon)	3,486	2,088	2,144
Mutually authenticated			
Round trips until client receives response data	4	3	3
Size (bytes) of public key crypto objects transmitted:			
• Minimum PQ	1,431	2,152	1,060
• MLWE/MSIS	9,554	10,140	6,324
• NTRU	5,574	4,365	4,185

Other security properties

Anonymity

- Client certificate encrypted
- Server certificate encrypted
- Server identity not protected
  - Due to Server Name Indication extension
  - May be able to combine KEMTLS-PDK with Encrypted ClientHello?

Deniability

- KEMTLS and KEMTLS-PDK don't use signatures for authentication
- Yields offline deniability
  - Judge cannot distinguish honest transcript from forgery
- Does not yield online deniability
  - When one party doesn't follow protocol or colludes with jduge

TLS ecosystem is complex – lots to consider!

- Datagram TLS
- Use of TLS handshake in other protocols
    - e.g. QUIC
- Application-specific behaviour
    - e.g. HTTP3 SETTINGS frame not server authenticated
- PKI involving KEM public keys
- Long tail of implementations
- …

X.509 certificates for KEM public keys: Proof of possession

How does requester prove possession of corresponding secret keys?

- Interactive challenge-response protocol: RFC 4210 Sect. 5.2.8.3
- Send certificate back encrypted under subject public key RFC 4210 Sect. 5.2.8.2
  - Weird confidentiality requirement on certificate. Maybe broken by Certificate Transparency?
- Non-interactive certificate signing requests: Not a signature scheme!
  - Research in progress: Can build a not-too-inefficient Picnic-like signature scheme from the KEM operation
    - Kyber proof of possession: 227 KB, < 1 sec proof generation and verifcation

Integrating post-quantum cryptography into protocols: the case of TLS

Douglas Stebila

UNIVERSITY OF WATERLOO

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

Benchmarking and prototypes

Open Quantum Safe project

https://eprint.iacr.org/2019/1447 • https://openquantumsafe.org • https://github.com/open-quantum-safe/

Hybrid key exchange in TLS 1.3

Working towards standardization

https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-03

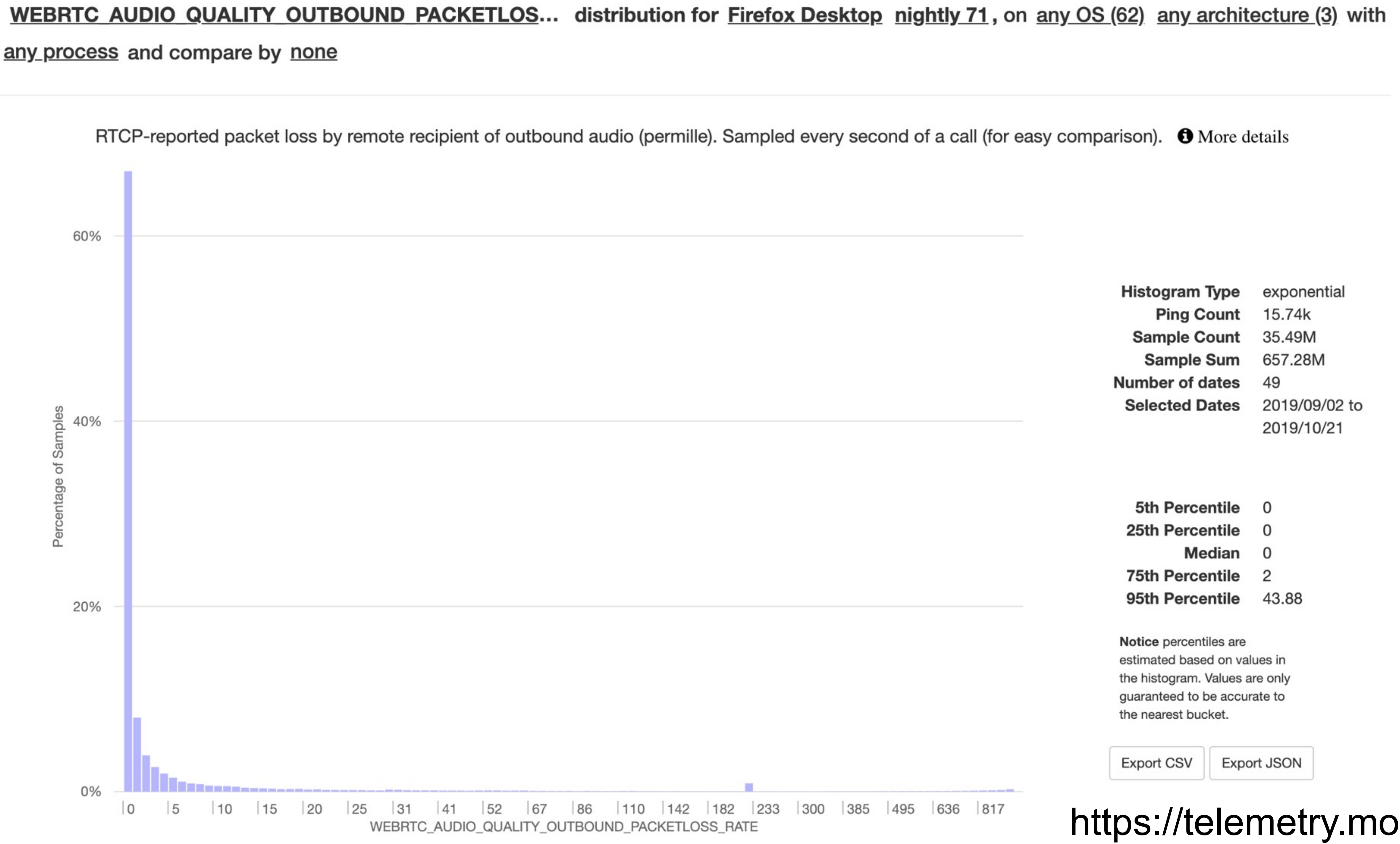
KEMTLS

Implicitly authenticated TLS without handshake signatures using KEMs

- Saves bytes on the wire and server CPU cycles
- Variants for client authentication and pre-distributed public keys
- Lots of work to make viable in TLS ecosystem, including certificates

https://eprint.iacr.org/2020/534 • https://eprint.iacr.org/2021/779
https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-00

Appendix: Benchmarking



WEBRTC_AUDIO_QUALITY_OUTBOUND_PACKETLOS... distribution for Firefox Desktop nightly 71, on any OS (62) any architecture (3) with any process and compare by none

RTCP-reported packet loss by remote recipient of outbound audio (permille). Sampled every second of a call (for easy comparison). More details

Histogram Type	exponential
Ping Count	15.74k
Sample Count	35.49M
Sample Sum	657.28M
Number of dates	49
Selected Dates	2019/09/02 to 2019/10/21

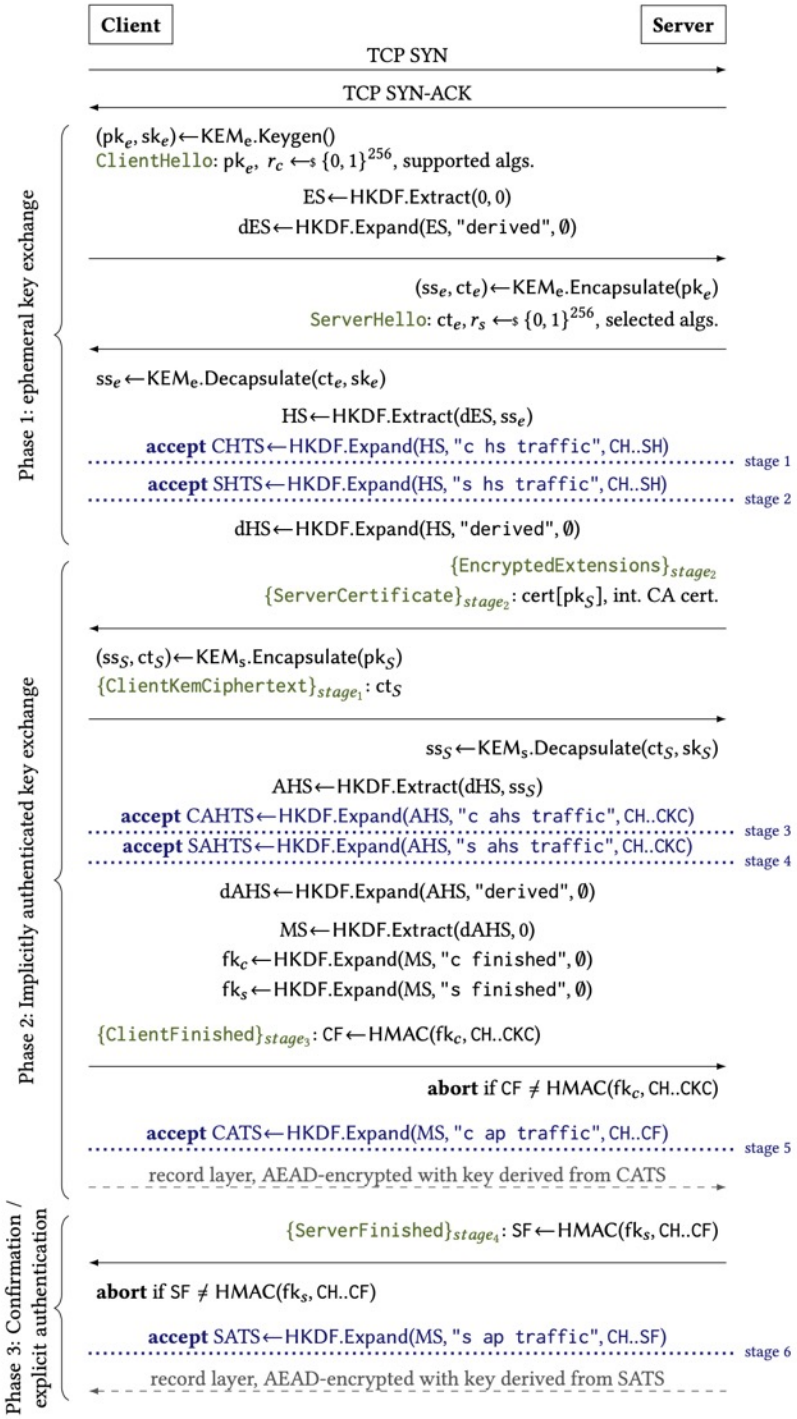
5th Percentile	0
25th Percentile	0
Median	0
75th Percentile	2
95th Percentile	43.88

Notice percentiles are estimated based on values in the histogram. Values are only guaranteed to be accurate to the nearest bucket.

https://telemetry.mozilla

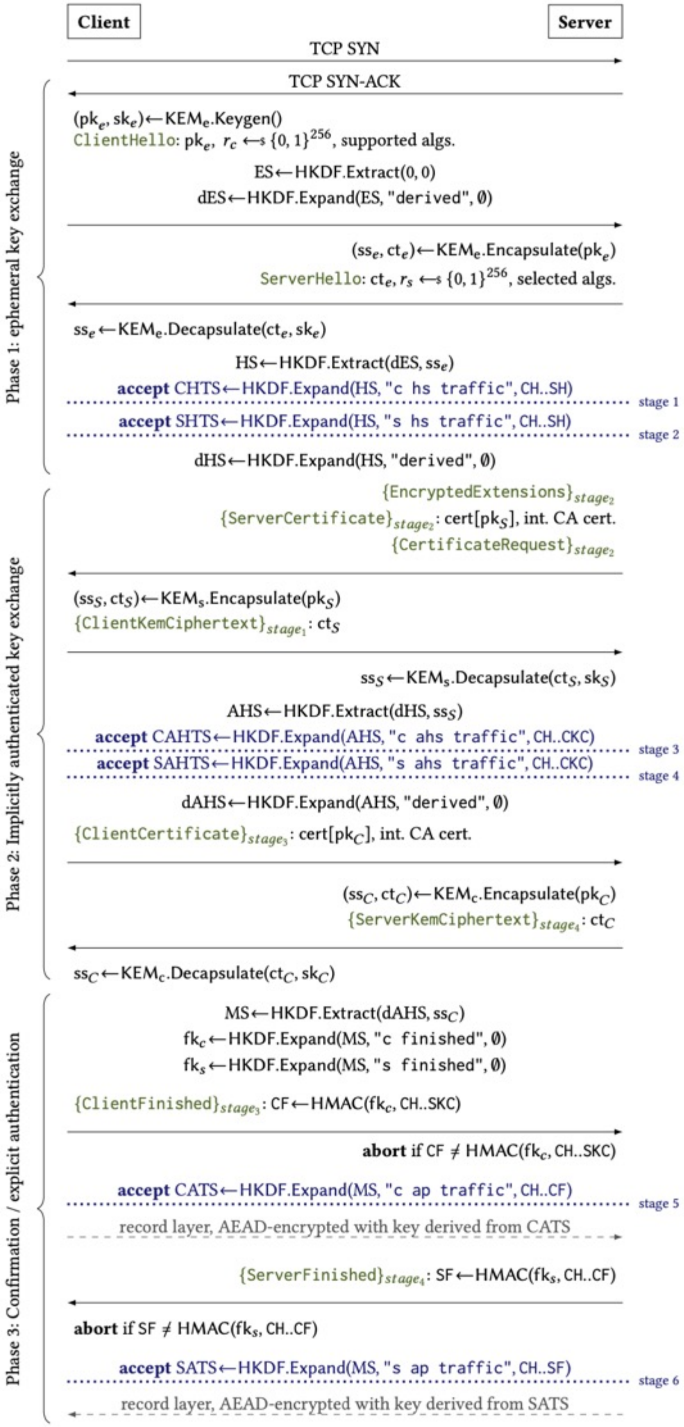
Appendix: KEMTLS

KEMTLS

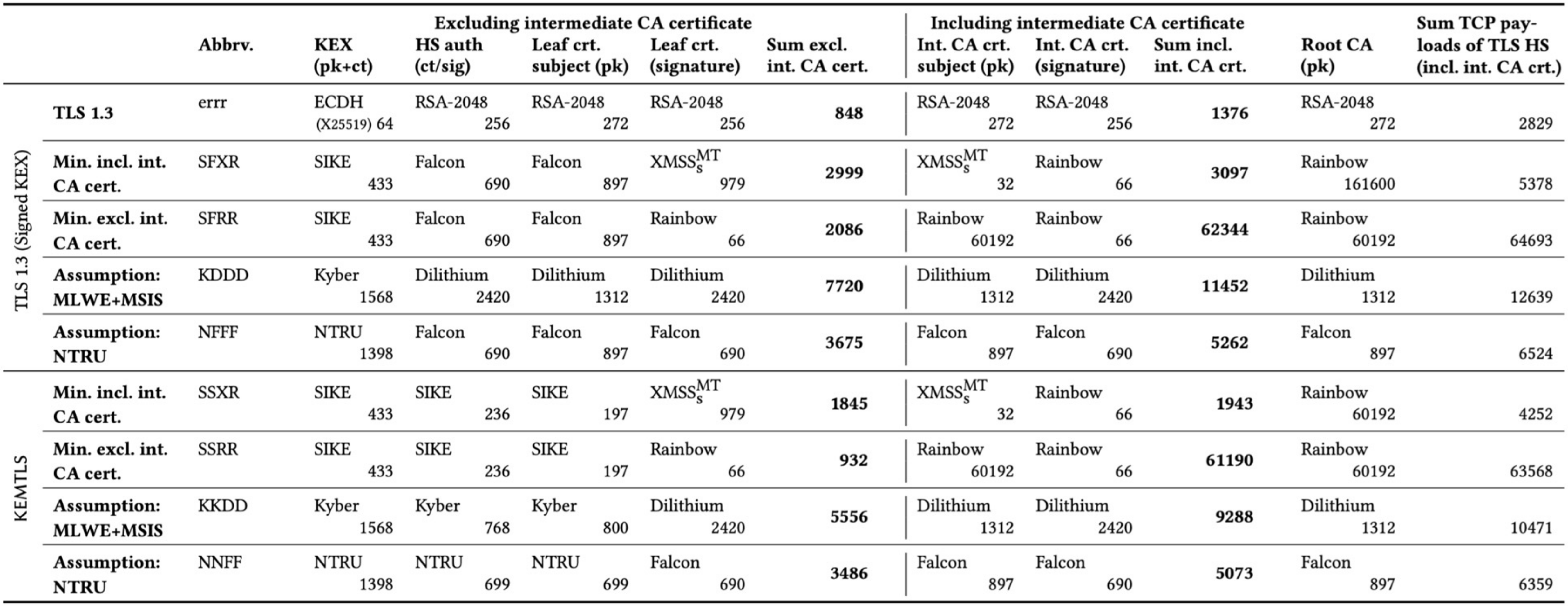


Client
Server
TCP SYN
TCP SYN-ACK
Phase 1: ephemeral key exchange
(pk_e, sk_e) ← KEM_e.Keygen()
ClientHello: pk_e, r_c ←$ {0,1}^256, supported algs.
ES ← HKDF.Extract(0, 0)
dES ← HKDF.Expand(ES, "derived", ∅)
(ss_e, ct_e) ← KEM_e.Encapsulate(pk_e)
ServerHello: ct_e, r_s ←$ {0,1}^256, selected algs.
ss_e ← KEM_e.Decapsulate(ct_e, sk_e)
HS ← HKDF.Extract(dES, ss_e)
accept CHTS ← HKDF.Expand(HS, "c hs traffic", CH..SH)
stage 1
accept SHTS ← HKDF.Expand(HS, "s hs traffic", CH..SH)
stage 2
dHS ← HKDF.Expand(HS, "derived", ∅)
Phase 2: Implicitly authenticated key exchange
{EncryptedExtensions}_stage2
{ServerCertificate}_stage2: cert[pk_S], int. CA cert.
(ss_S, ct_S) ← KEM_s.Encapsulate(pk_S)
{ClientKemCiphertext}_stage1: ct_S
ss_S ← KEM_s.Decapsulate(ct_S, sk_S)
AHS ← HKDF.Extract(dHS, ss_S)
accept CAHTS ← HKDF.Expand(AHS, "c ahs traffic", CH..CKC)
stage 3
accept SAHTS ← HKDF.Expand(AHS, "s ahs traffic", CH..CKC)
stage 4
dAHS ← HKDF.Expand(AHS, "derived", ∅)
MS ← HKDF.Extract(dAHS, 0)
fk_c ← HKDF.Expand(MS, "c finished", ∅)
fk_s ← HKDF.Expand(MS, "s finished", ∅)
{ClientFinished}_stage3: CF ← HMAC(fk_c, CH..CKC)
abort if CF ≠ HMAC(fk_c, CH..CKC)
accept CATS ← HKDF.Expand(MS, "c ap traffic", CH..CF)
stage 5
record layer, AEAD-encrypted with key derived from CATS
Phase 3: Confirmation / explicit authentication
{ServerFinished}_stage4: SF ← HMAC(fk_s, CH..CF)
abort if SF ≠ HMAC(fk_s, CH..CF)
accept SATS ← HKDF.Expand(MS, "s ap traffic", CH..SF)
stage 6
record layer, AEAD-encrypted with key derived from SATS

KEMTLS with client authentication

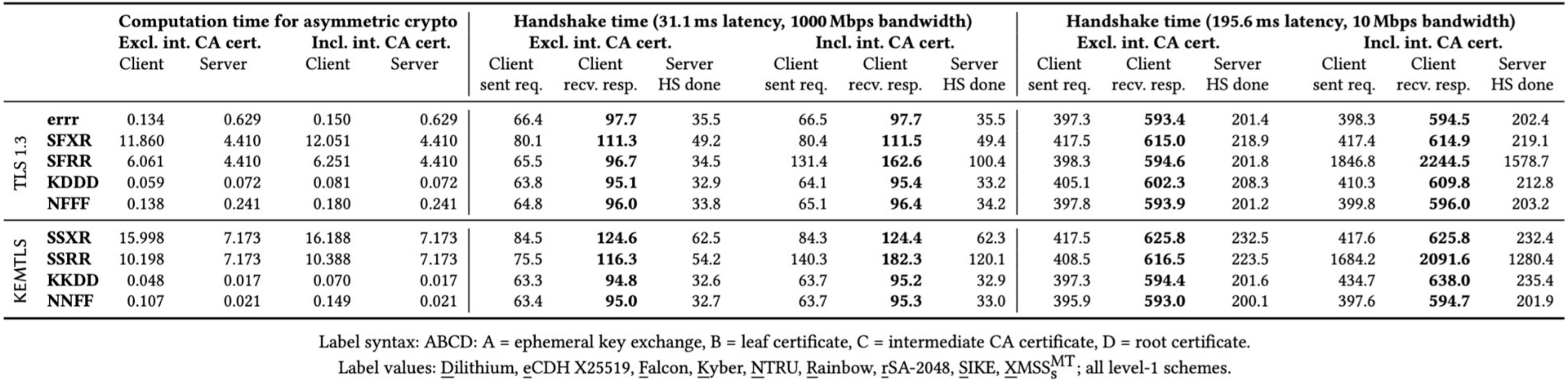


TLS 1.3 and KEMTLS size of public key objects



				Excluding intermediate CA certificate				Including intermediate CA certificate				
		Abbrv.	KEX (pk+ct)	HS auth (ct/sig)	Leaf crt. subject (pk)	Leaf crt. (signature)	Sum excl. int. CA cert.	Int. CA crt. subject (pk)	Int. CA crt. (signature)	Sum incl. int. CA crt.	Root CA (pk)	Sum TCP payloads of TLS HS (incl. int. CA crt.)
	TLS 1.3	errr	ECDH (X25519) 64	RSA-2048 256	RSA-2048 272	RSA-2048 256	**848**	RSA-2048 272	RSA-2048 256	**1376**	RSA-2048 272	2829
TLS 1.3 (Signed KEX)	**Min. incl. int. CA cert.**	SFXR	SIKE 433	Falcon 690	Falcon 897	XMSS_s^{MT} 979	**2999**	XMSS_s^{MT} 32	Rainbow 66	**3097**	Rainbow 161600	5378
	Min. excl. int. CA cert.	SFRR	SIKE 433	Falcon 690	Falcon 897	Rainbow 66	**2086**	Rainbow 60192	Rainbow 66	**62344**	Rainbow 60192	64693
	Assumption: MLWE+MSIS	KDDD	Kyber 1568	Dilithium 2420	Dilithium 1312	Dilithium 2420	**7720**	Dilithium 1312	Dilithium 2420	**11452**	Dilithium 1312	12639
	Assumption: NTRU	NFFF	NTRU 1398	Falcon 690	Falcon 897	Falcon 690	**3675**	Falcon 897	Falcon 690	**5262**	Falcon 897	6524
KEMTLS	**Min. incl. int. CA cert.**	SSXR	SIKE 433	SIKE 236	SIKE 197	XMSS_s^{MT} 979	**1845**	XMSS_s^{MT} 32	Rainbow 66	**1943**	Rainbow 60192	4252
	Min. excl. int. CA cert.	SSRR	SIKE 433	SIKE 236	SIKE 197	Rainbow 66	**932**	Rainbow 60192	Rainbow 66	**61190**	Rainbow 60192	63568
	Assumption: MLWE+MSIS	KKDD	Kyber 1568	Kyber 768	Kyber 800	Dilithium 2420	**5556**	Dilithium 1312	Dilithium 2420	**9288**	Dilithium 1312	10471
	Assumption: NTRU	NNFF	NTRU 1398	NTRU 699	NTRU 699	Falcon 690	**3486**	Falcon 897	Falcon 690	**5073**	Falcon 897	6359

TLS 1.3 and KEMTLS crypto & handshake time

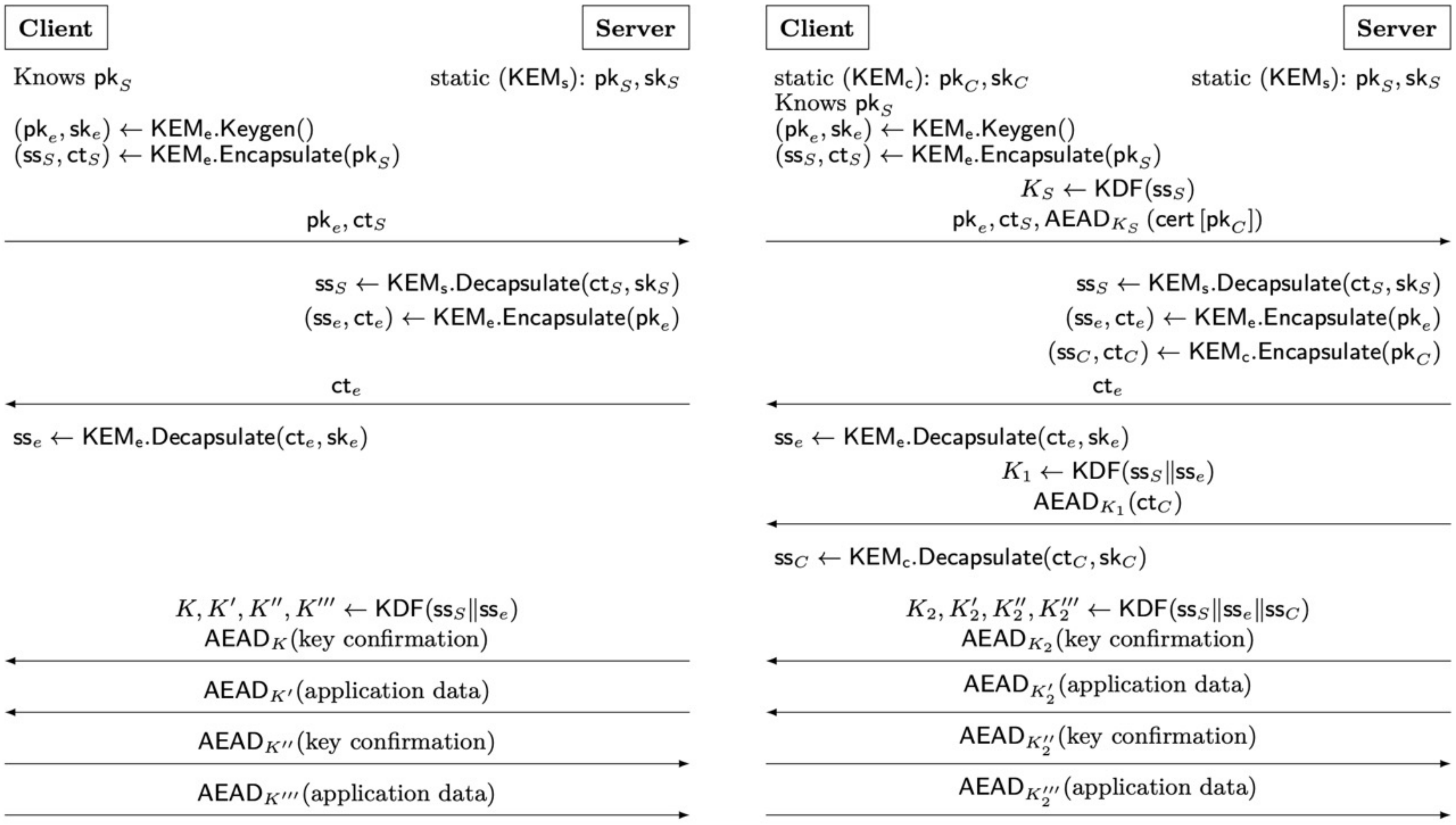


		Computation time for asymmetric crypto				Handshake time (31.1 ms latency, 1000 Mbps bandwidth)						Handshake time (195.6 ms latency, 10 Mbps bandwidth)					
		Excl. int. CA cert.		Incl. int. CA cert.		Excl. int. CA cert.			Incl. int. CA cert.			Excl. int. CA cert.			Incl. int. CA cert.		
		Client	Server	Client	Server	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done
TLS 1.3	**errr**	0.134	0.629	0.150	0.629	66.4	**97.7**	35.5	66.5	**97.7**	35.5	397.3	**593.4**	201.4	398.3	**594.5**	202.4
	SFXR	11.860	4.410	12.051	4.410	80.1	**111.3**	49.2	80.4	**111.5**	49.4	417.5	**615.0**	218.9	417.4	**614.9**	219.1
	SFRR	6.061	4.410	6.251	4.410	65.5	**96.7**	34.5	131.4	**162.6**	100.4	398.3	**594.6**	201.8	1846.8	**2244.5**	1578.7
	KDDD	0.059	0.072	0.081	0.072	63.8	**95.1**	32.9	64.1	**95.4**	33.2	405.1	**602.3**	208.3	410.3	**609.8**	212.8
	NFFF	0.138	0.241	0.180	0.241	64.8	**96.0**	33.8	65.1	**96.4**	34.2	397.8	**593.9**	201.2	399.8	**596.0**	203.2
KEMTLS	**SSXR**	15.998	7.173	16.188	7.173	84.5	**124.6**	62.5	84.3	**124.4**	62.3	417.5	**625.8**	232.5	417.6	**625.8**	232.4
	SSRR	10.198	7.173	10.388	7.173	75.5	**116.3**	54.2	140.3	**182.3**	120.1	408.5	**616.5**	223.5	1684.2	**2091.6**	1280.4
	KKDD	0.048	0.017	0.070	0.017	63.3	**94.8**	32.6	63.7	**95.2**	32.9	397.3	**594.4**	201.6	434.7	**638.0**	235.4
	NNFF	0.107	0.021	0.149	0.021	63.4	**95.0**	32.7	63.7	**95.3**	33.0	395.9	**593.0**	200.1	397.6	**594.7**	201.9

Label syntax: ABCD: A = ephemeral key exchange, B = leaf certificate, C = intermediate CA certificate, D = root certificate.

Label values: Dilithium, eCDH X25519, Falcon, Kyber, NTRU, Rainbow, rSA-2048, SIKE, XMSS$_s^{MT}$; all level-1 schemes.

KEMTLS-PDK overview



Client — **Server**

Knows pk_S — static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$

$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$

$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_S)$

$\xrightarrow{\mathsf{pk}_e, \mathsf{ct}_S}$

$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$

$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$

$\xleftarrow{\mathsf{ct}_e}$

$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$

$K, K', K'', K''' \leftarrow \mathsf{KDF}(\mathsf{ss}_S \| \mathsf{ss}_e)$

$\xleftarrow{\mathsf{AEAD}_K(\text{key confirmation})}$

$\xleftarrow{\mathsf{AEAD}_{K'}(\text{application data})}$

$\xrightarrow{\mathsf{AEAD}_{K''}(\text{key confirmation})}$

$\xrightarrow{\mathsf{AEAD}_{K'''}(\text{application data})}$

(a) Unilaterally authenticated

Client — **Server**

static ($\mathsf{KEM_c}$): $\mathsf{pk}_C, \mathsf{sk}_C$ — static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$

Knows pk_S

$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$

$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_S)$

$K_S \leftarrow \mathsf{KDF}(\mathsf{ss}_S)$

$\xrightarrow{\mathsf{pk}_e, \mathsf{ct}_S, \mathsf{AEAD}_{K_S}(\mathsf{cert}[\mathsf{pk}_C])}$

$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$

$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$

$(\mathsf{ss}_C, \mathsf{ct}_C) \leftarrow \mathsf{KEM_c.Encapsulate}(\mathsf{pk}_C)$

$\xleftarrow{\mathsf{ct}_e}$

$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$

$K_1 \leftarrow \mathsf{KDF}(\mathsf{ss}_S \| \mathsf{ss}_e)$

$\xleftarrow{\mathsf{AEAD}_{K_1}(\mathsf{ct}_C)}$

$\mathsf{ss}_C \leftarrow \mathsf{KEM_c.Decapsulate}(\mathsf{ct}_C, \mathsf{sk}_C)$

$K_2, K_2', K_2'', K_2''' \leftarrow \mathsf{KDF}(\mathsf{ss}_S \| \mathsf{ss}_e \| \mathsf{ss}_C)$

$\xleftarrow{\mathsf{AEAD}_{K_2}(\text{key confirmation})}$

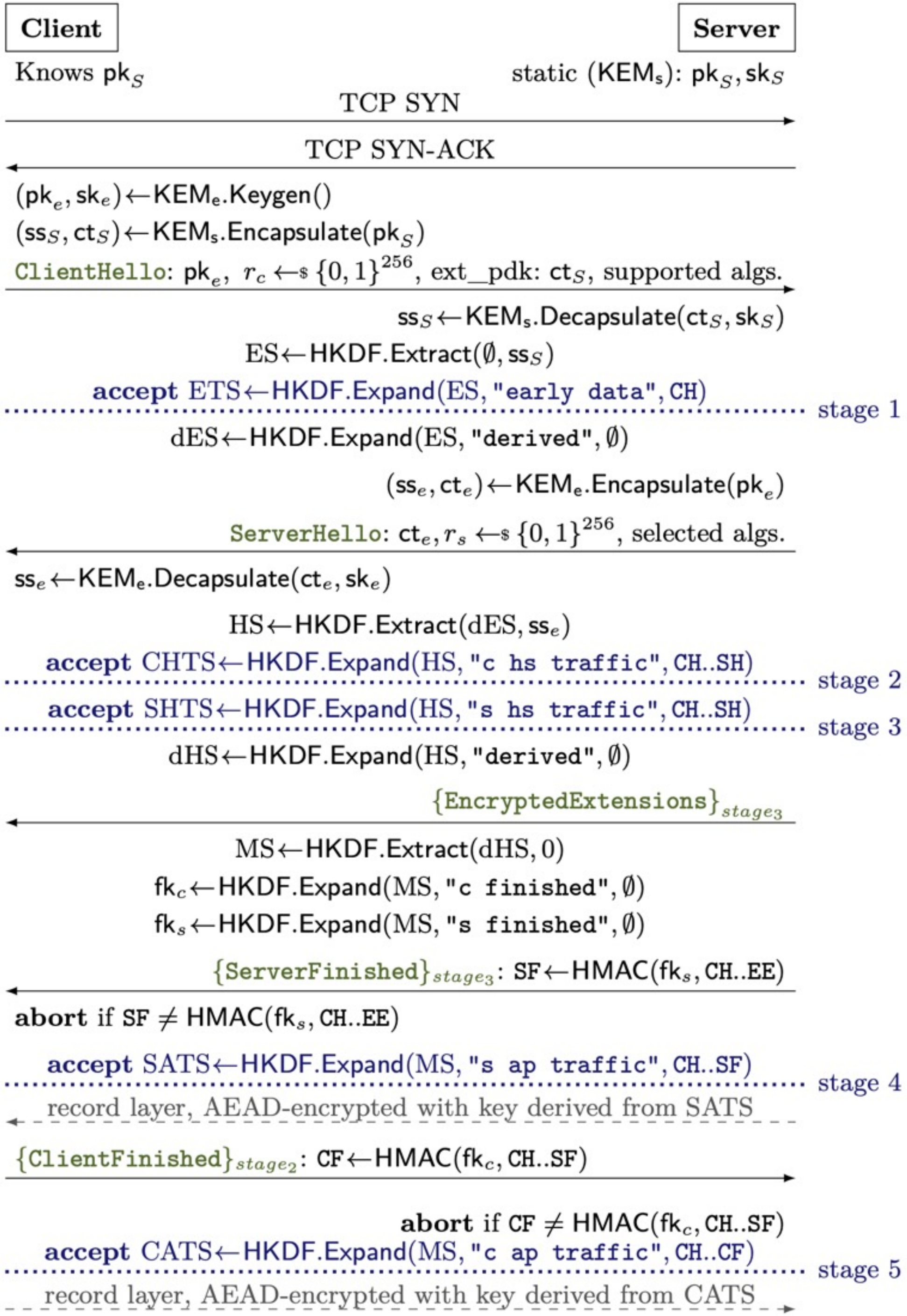
$\xleftarrow{\mathsf{AEAD}_{K_2'}(\text{application data})}$

$\xrightarrow{\mathsf{AEAD}_{K_2''}(\text{key confirmation})}$

$\xrightarrow{\mathsf{AEAD}_{K_2'''}(\text{application data})}$

(b) With proactive client authentication

KEMTLS-PDK



Client | **Server**

Knows pk_S | static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$

TCP SYN

TCP SYN-ACK

$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$

$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_s.Encapsulate}(\mathsf{pk}_S)$

`ClientHello`: pk_e, $r_c \leftarrow_\$ \{0,1\}^{256}$, ext_pdk: ct_S, supported algs.

$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$

$\mathrm{ES} \leftarrow \mathsf{HKDF.Extract}(\emptyset, \mathsf{ss}_S)$

accept $\mathrm{ETS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \texttt{"early data"}, \texttt{CH})$ — stage 1

$\mathrm{dES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \texttt{"derived"}, \emptyset)$

$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$

`ServerHello`: $\mathsf{ct}_e, r_s \leftarrow_\$ \{0,1\}^{256}$, selected algs.

$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$

$\mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dES}, \mathsf{ss}_e)$

accept $\mathrm{CHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"c hs traffic"}, \texttt{CH..SH})$ — stage 2

accept $\mathrm{SHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"s hs traffic"}, \texttt{CH..SH})$ — stage 3

$\mathrm{dHS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"derived"}, \emptyset)$

$\{\texttt{EncryptedExtensions}\}_{stage_3}$

$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dHS}, 0)$

$\mathsf{fk}_c \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"c finished"}, \emptyset)$

$\mathsf{fk}_s \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"s finished"}, \emptyset)$

$\{\texttt{ServerFinished}\}_{stage_3}$: $\texttt{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_s, \texttt{CH..EE})$

abort if $\texttt{SF} \neq \mathsf{HMAC}(\mathsf{fk}_s, \texttt{CH..EE})$

accept $\mathrm{SATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"s ap traffic"}, \texttt{CH..SF})$ — stage 4

record layer, AEAD-encrypted with key derived from SATS

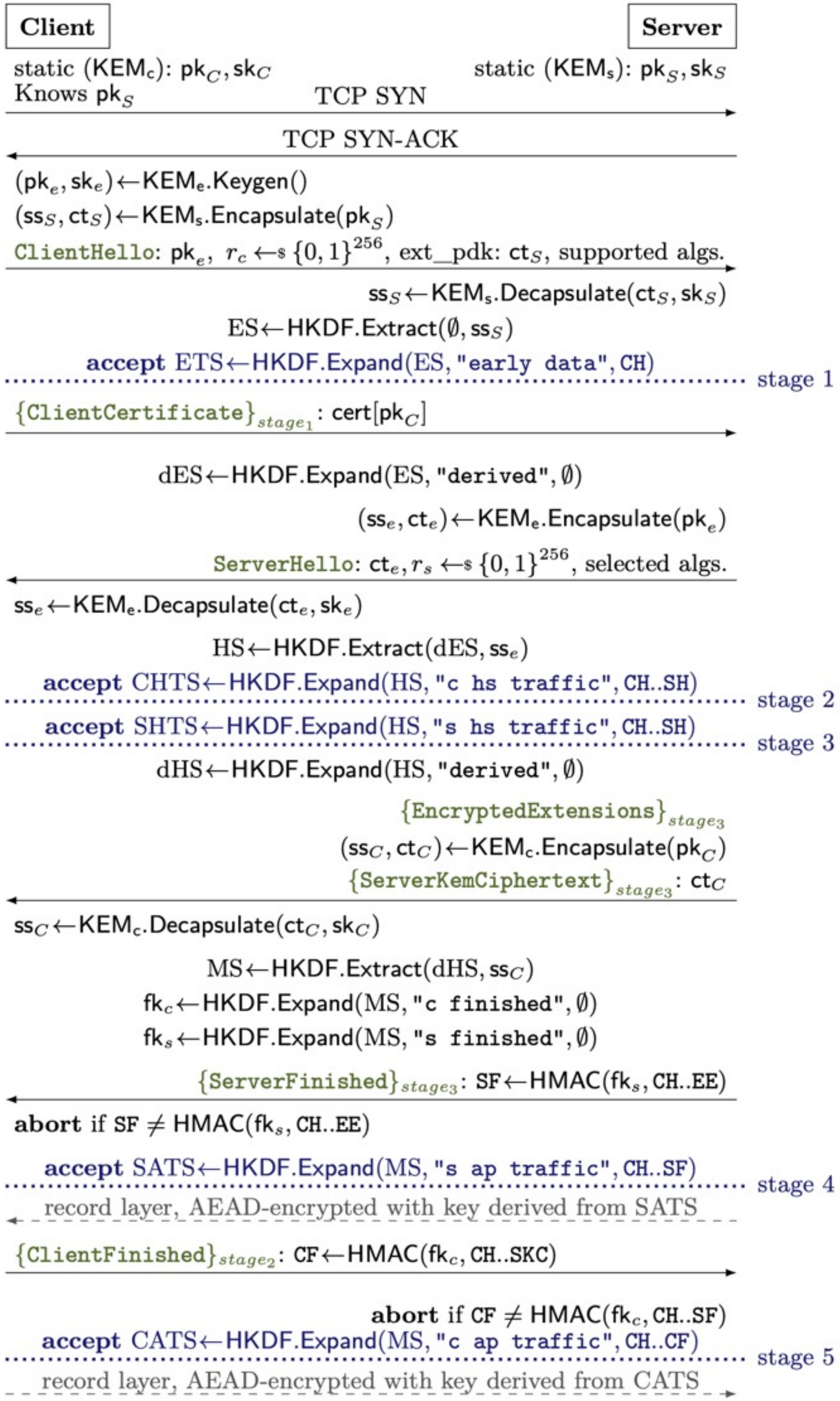
$\{\texttt{ClientFinished}\}_{stage_2}$: $\texttt{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_c, \texttt{CH..SF})$

abort if $\texttt{CF} \neq \mathsf{HMAC}(\mathsf{fk}_c, \texttt{CH..SF})$

accept $\mathrm{CATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"c ap traffic"}, \texttt{CH..CF})$ — stage 5

record layer, AEAD-encrypted with key derived from CATS

KEMTLS-PDK with proactive client authentication



Client **Server**

static ($\mathsf{KEM_c}$): $\mathsf{pk}_C, \mathsf{sk}_C$ static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$

Knows pk_S

TCP SYN

TCP SYN-ACK

$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$

$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_s.Encapsulate}(\mathsf{pk}_S)$

`ClientHello`: $\mathsf{pk}_e,\ r_c \leftarrow\$ \{0,1\}^{256}$, ext_pdk: ct_S, supported algs.

$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$

$\mathrm{ES} \leftarrow \mathsf{HKDF.Extract}(\emptyset, \mathsf{ss}_S)$

accept $\mathrm{ETS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \texttt{"early data"}, \texttt{CH})$

stage 1

$\{\texttt{ClientCertificate}\}_{stage_1}$: $\mathsf{cert}[\mathsf{pk}_C]$

$\mathrm{dES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \texttt{"derived"}, \emptyset)$

$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$

`ServerHello`: $\mathsf{ct}_e, r_s \leftarrow\$ \{0,1\}^{256}$, selected algs.

$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$

$\mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dES}, \mathsf{ss}_e)$

accept $\mathrm{CHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"c hs traffic"}, \texttt{CH..SH})$

stage 2

accept $\mathrm{SHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"s hs traffic"}, \texttt{CH..SH})$

stage 3

$\mathrm{dHS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"derived"}, \emptyset)$

$\{\texttt{EncryptedExtensions}\}_{stage_3}$

$(\mathsf{ss}_C, \mathsf{ct}_C) \leftarrow \mathsf{KEM_c.Encapsulate}(\mathsf{pk}_C)$

$\{\texttt{ServerKemCiphertext}\}_{stage_3}$: ct_C

$\mathsf{ss}_C \leftarrow \mathsf{KEM_c.Decapsulate}(\mathsf{ct}_C, \mathsf{sk}_C)$

$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dHS}, \mathsf{ss}_C)$

$\mathsf{fk}_c \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"c finished"}, \emptyset)$

$\mathsf{fk}_s \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"s finished"}, \emptyset)$

$\{\texttt{ServerFinished}\}_{stage_3}$: $\texttt{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_s, \texttt{CH..EE})$

abort if $\texttt{SF} \neq \mathsf{HMAC}(\mathsf{fk}_s, \texttt{CH..EE})$

accept $\mathrm{SATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"s ap traffic"}, \texttt{CH..SF})$

stage 4

record layer, AEAD-encrypted with key derived from SATS

$\{\texttt{ClientFinished}\}_{stage_2}$: $\texttt{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_c, \texttt{CH..SKC})$

abort if $\texttt{CF} \neq \mathsf{HMAC}(\mathsf{fk}_c, \texttt{CH..SF})$

accept $\mathrm{CATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"c ap traffic"}, \texttt{CH..CF})$

stage 5

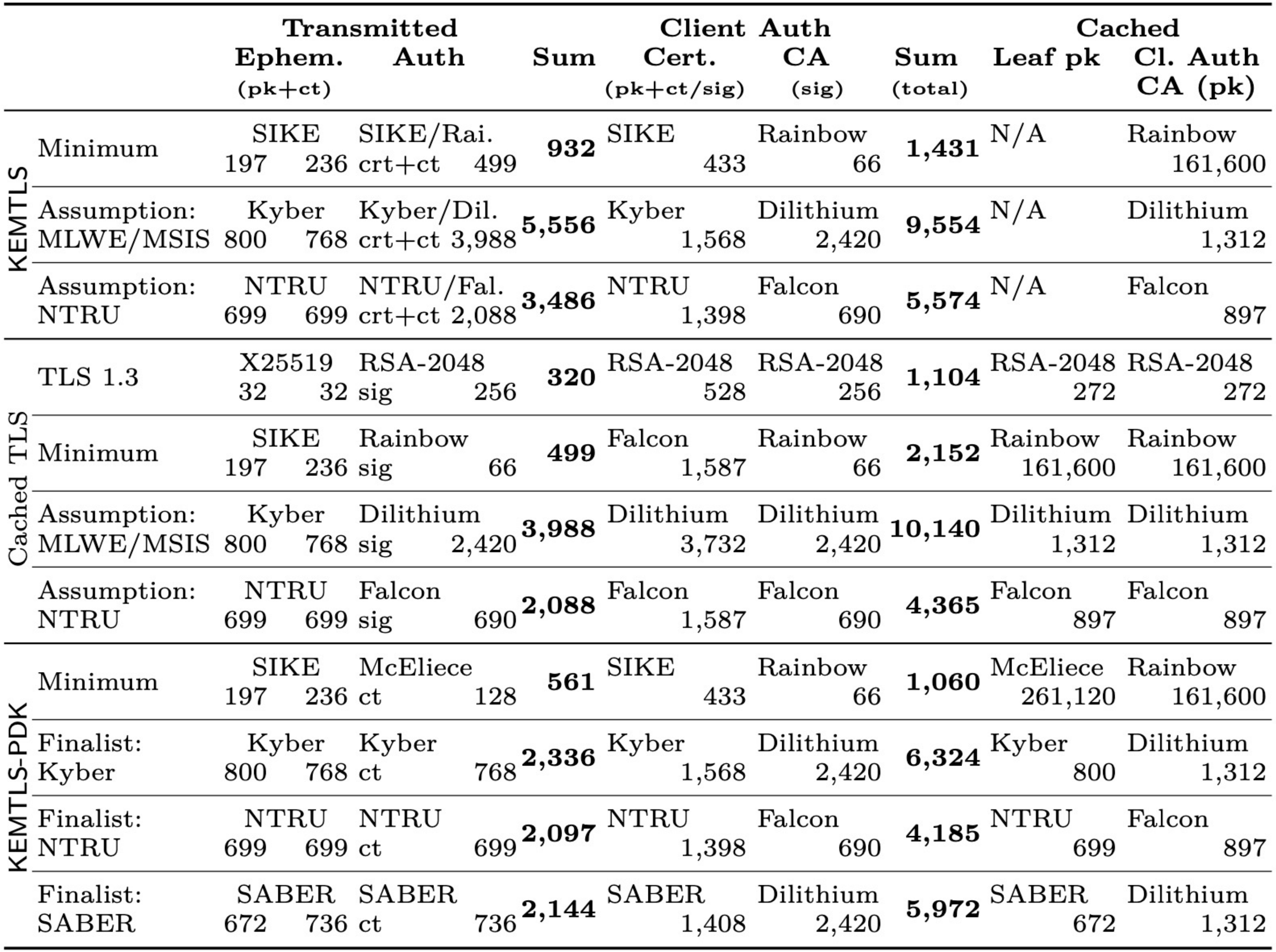
record layer, AEAD-encrypted with key derived from CATS

Communication sizes

KEMTLS

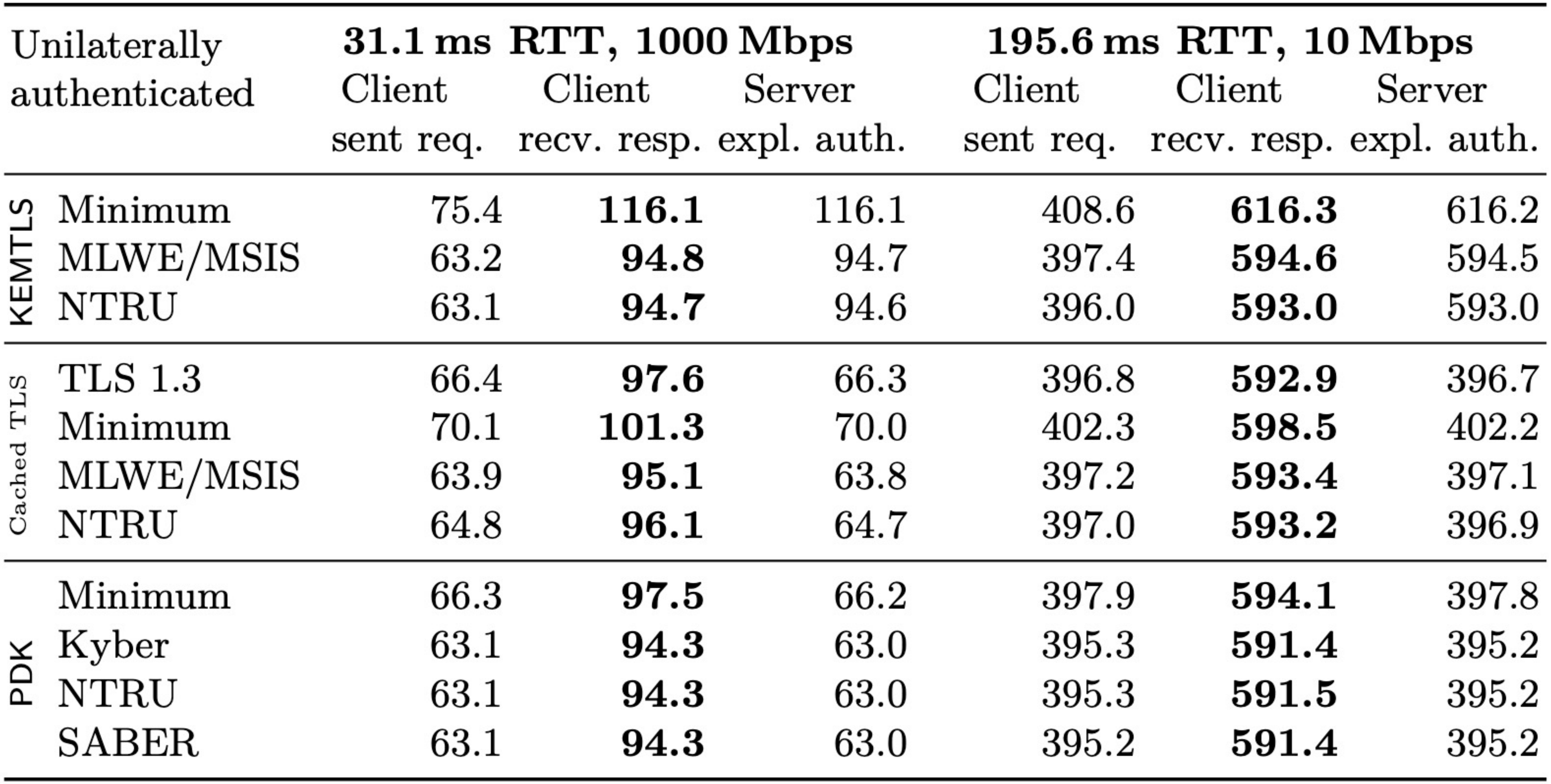
TLS 1.3 w/cached server certs

KEMTLS-PDK



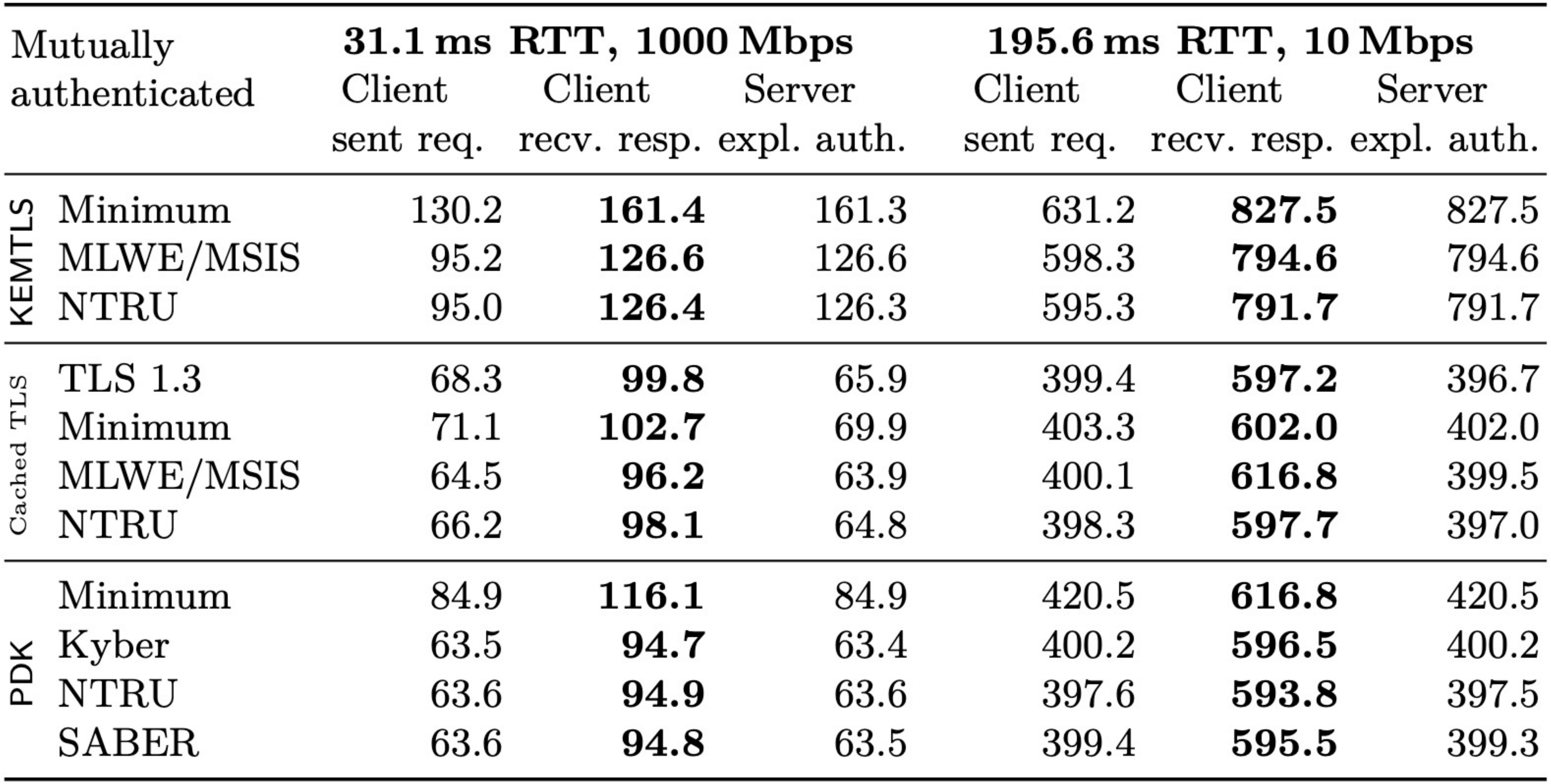
		Transmitted		Sum	Client Auth		Sum	Cached	
		Ephem. (pk+ct)	Auth		Cert. (pk+ct/sig)	CA (sig)	(total)	Leaf pk	Cl. Auth CA (pk)
KEMTLS	Minimum	SIKE 197 236	SIKE/Rai. crt+ct 499	**932**	SIKE 433	Rainbow 66	**1,431**	N/A	Rainbow 161,600
	Assumption: MLWE/MSIS	Kyber 800 768	Kyber/Dil. crt+ct 3,988	**5,556**	Kyber 1,568	Dilithium 2,420	**9,554**	N/A	Dilithium 1,312
	Assumption: NTRU	NTRU 699 699	NTRU/Fal. crt+ct 2,088	**3,486**	NTRU 1,398	Falcon 690	**5,574**	N/A	Falcon 897
Cached TLS	TLS 1.3	X25519 32 32	RSA-2048 sig 256	**320**	RSA-2048 528	RSA-2048 256	**1,104**	RSA-2048 272	RSA-2048 272
	Minimum	SIKE 197 236	Rainbow sig 66	**499**	Falcon 1,587	Rainbow 66	**2,152**	Rainbow 161,600	Rainbow 161,600
	Assumption: MLWE/MSIS	Kyber 800 768	Dilithium sig 2,420	**3,988**	Dilithium 3,732	Dilithium 2,420	**10,140**	Dilithium 1,312	Dilithium 1,312
	Assumption: NTRU	NTRU 699 699	Falcon sig 690	**2,088**	Falcon 1,587	Falcon 690	**4,365**	Falcon 897	Falcon 897
KEMTLS-PDK	Minimum	SIKE 197 236	McEliece ct 128	**561**	SIKE 433	Rainbow 66	**1,060**	McEliece 261,120	Rainbow 161,600
	Finalist: Kyber	Kyber 800 768	Kyber ct 768	**2,336**	Kyber 1,568	Dilithium 2,420	**6,324**	Kyber 800	Dilithium 1,312
	Finalist: NTRU	NTRU 699 699	NTRU ct 699	**2,097**	NTRU 1,398	Falcon 690	**4,185**	NTRU 699	Falcon 897
	Finalist: SABER	SABER 672 736	SABER ct 736	**2,144**	SABER 1,408	Dilithium 2,420	**5,972**	SABER 672	Dilithium 1,312

Handshake times, unilateral authentication



	Unilaterally authenticated	31.1 ms RTT, 1000 Mbps Client sent req.	 Client recv. resp.	 Server expl. auth.	195.6 ms RTT, 10 Mbps Client sent req.	 Client recv. resp.	 Server expl. auth.
KEMTLS	Minimum	75.4	**116.1**	116.1	408.6	**616.3**	616.2
	MLWE/MSIS	63.2	**94.8**	94.7	397.4	**594.6**	594.5
	NTRU	63.1	**94.7**	94.6	396.0	**593.0**	593.0
Cached TLS	TLS 1.3	66.4	**97.6**	66.3	396.8	**592.9**	396.7
	Minimum	70.1	**101.3**	70.0	402.3	**598.5**	402.2
	MLWE/MSIS	63.9	**95.1**	63.8	397.2	**593.4**	397.1
	NTRU	64.8	**96.1**	64.7	397.0	**593.2**	396.9
PDK	Minimum	66.3	**97.5**	66.2	397.9	**594.1**	397.8
	Kyber	63.1	**94.3**	63.0	395.3	**591.4**	395.2
	NTRU	63.1	**94.3**	63.0	395.3	**591.5**	395.2
	SABER	63.1	**94.3**	63.0	395.2	**591.4**	395.2

Handshake times, mutual authentication



	Mutually authenticated	31.1 ms RTT, 1000 Mbps			195.6 ms RTT, 10 Mbps		
		Client sent req.	Client recv. resp.	Server expl. auth.	Client sent req.	Client recv. resp.	Server expl. auth.
KEMTLS	Minimum	130.2	**161.4**	161.3	631.2	**827.5**	827.5
	MLWE/MSIS	95.2	**126.6**	126.6	598.3	**794.6**	794.6
	NTRU	95.0	**126.4**	126.3	595.3	**791.7**	791.7
Cached TLS	TLS 1.3	68.3	**99.8**	65.9	399.4	**597.2**	396.7
	Minimum	71.1	**102.7**	69.9	403.3	**602.0**	402.0
	MLWE/MSIS	64.5	**96.2**	63.9	400.1	**616.8**	399.5
	NTRU	66.2	**98.1**	64.8	398.3	**597.7**	397.0
PDK	Minimum	84.9	**116.1**	84.9	420.5	**616.8**	420.5
	Kyber	63.5	**94.7**	63.4	400.2	**596.5**	400.2
	NTRU	63.6	**94.9**	63.6	397.6	**593.8**	397.5
	SABER	63.6	**94.8**	63.5	399.4	**595.5**	399.3

OPEN QUANTUM SAFE

software for prototyping quantum-resistant cryptography

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 0.7.0 released August 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