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-02

https://openquantumsafe.org/

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

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

Netherlands Crypto Working Group • 2020-02-07

UNIVERSITY OF WATERLOO

IQC Institute for Quantum Computing

CYBER SECURITY AND PRIVACY INSTITUTE
UNIVERSITY OF WATERLOO

Overview

- Design issues in adding hybrid key exchange to Internet protocols
- Open Quantum Safe project
- Compatibility issues of post-quantum & hybrid key exchange and authentication in SSH and TLS
- Performance of post-quantum & hybrid key exchange and authentication in TLS

“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 2019/20
  - 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. **Hybrid key exchange in TLS 1.3**. **Internet-Draft**. Internet Engineering Task Force, February 2020. https://tools.ietf.org/html/draft-stebila-tls-hybrid-design-02

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

Done in all(?) implementations to date.

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

Done in Amazon s2n TLS 1.2

Negotiate pre-defined combinations of algorithms

1. Standardize a name for each desired combination

- Can use existing negotiation data structures and logic

Done in all(?) other implementations to date

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
- Done in Amazon s2n TLS 1.2

2) Concatenate public keys

- For each supported combination, concatenate its public keys / ciphertext into an opaque data structure
- Done in all other implementations to date.

#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

Draft-02
February 2020

draft-stebila-tls-hybrid-design-02

Number of algorithms:

- 2

Negotiation:

- Negotiate pairs of algorithms in combination

Draft-02
February 2020

draft-stebila-tls-hybrid-design-02

Conveying public keys:

- Concatenated public keys
  - But with length encoding
  - Since some algorithms don't have fixed-length public keys / ciphertexts

Combining keying material:

- Concatenate shared secrets then put into TLS 1.3 key schedule
  - Key schedule applies HKDF.Extract
- No length encoding
- Will be approved by NIST in upcoming revision of SP-800-56C

Open questions

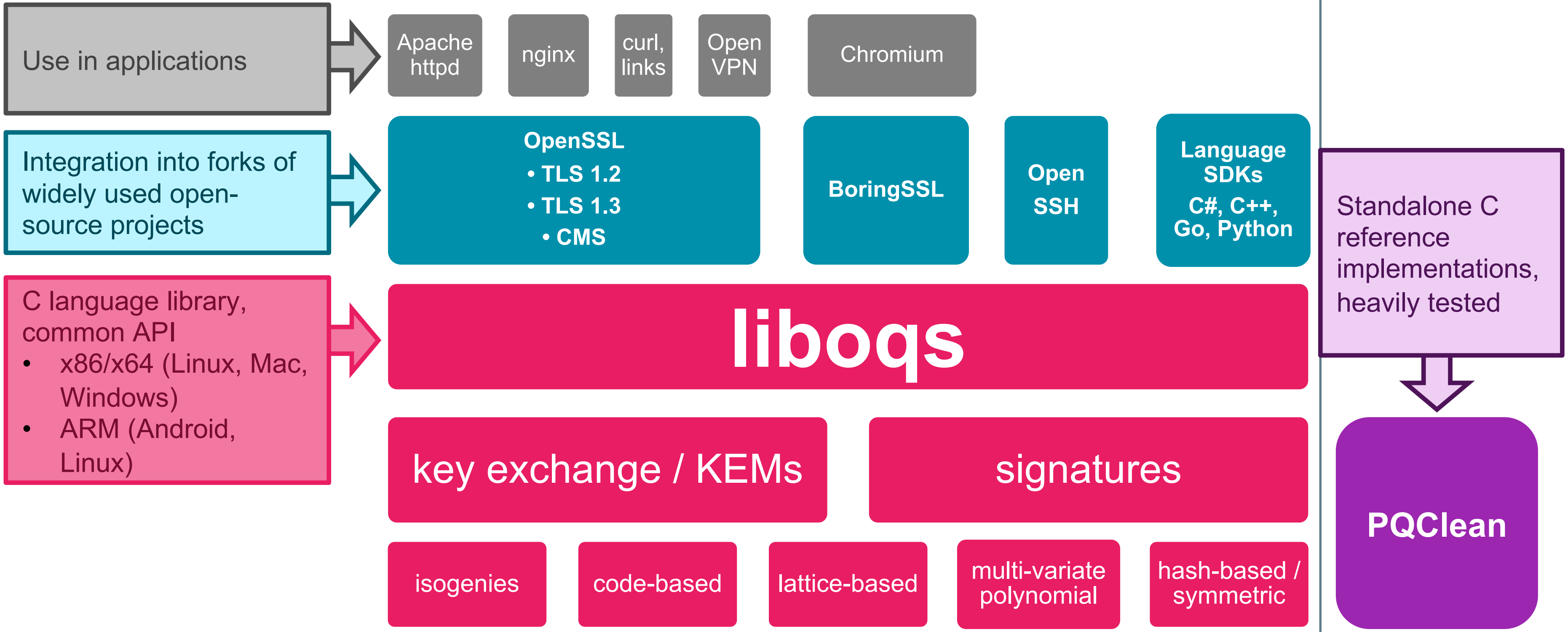
- Still some debate about negotiation and using concatenate public keys / ciphertexts
- Is it safe to use an IND-CPA KEM for ephemeral key exchange in TLS 1.3?
  - Intuitively, seems like it should be safe for **one-time** use keys
    - Some implementations re-use ephemeral keys which wouldn't match IND-CPA
  - But proofs of signed ephemeral DH in TLS 1.2 used an interactive assumption (PRF-ODH) rather than a standard assumption (DDH) (JKSS, C'12); was later shown to be necessary (KraPatWee, C'13)
  - Proofs of signed-DH in TLS 1.3 (BFGS CCS'15, …) also use PRF-ODH; no analysis of whether this is necessary, no generalization to KEMs)

OPEN QUANTUM SAFE

software for prototyping quantum-resistant cryptography

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

- 50 key encapsulation mechanisms from 9 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
- **LEDAcrypt:** LEDAcryptKEM-LT12, LEDAcryptKEM-LT32, LEDAcryptKEM-LT52
- **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
- **ThreeBears:** BabyBear, BabyBearEphem, MamaBear, MamaBearEphem, PapaBear,PapaBearEphem

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

- 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

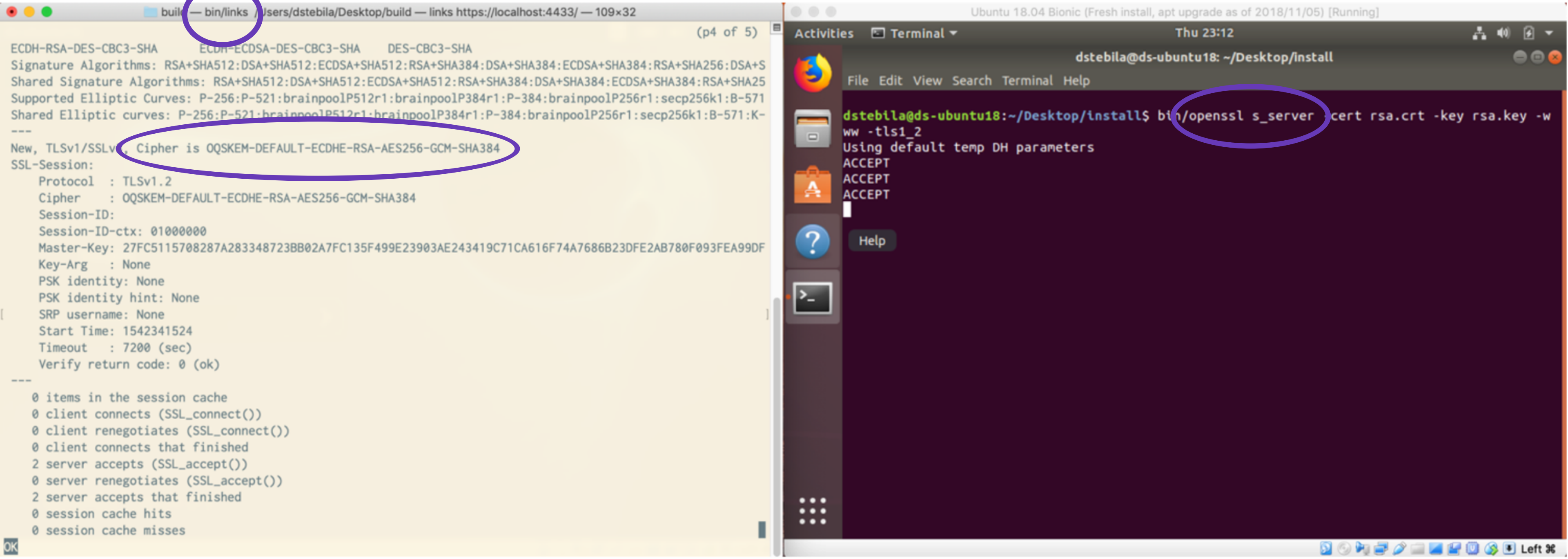
OpenSSL

- 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
  - PQ and hybrid signatures in Cryptographic Message Syntax (CMS)

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

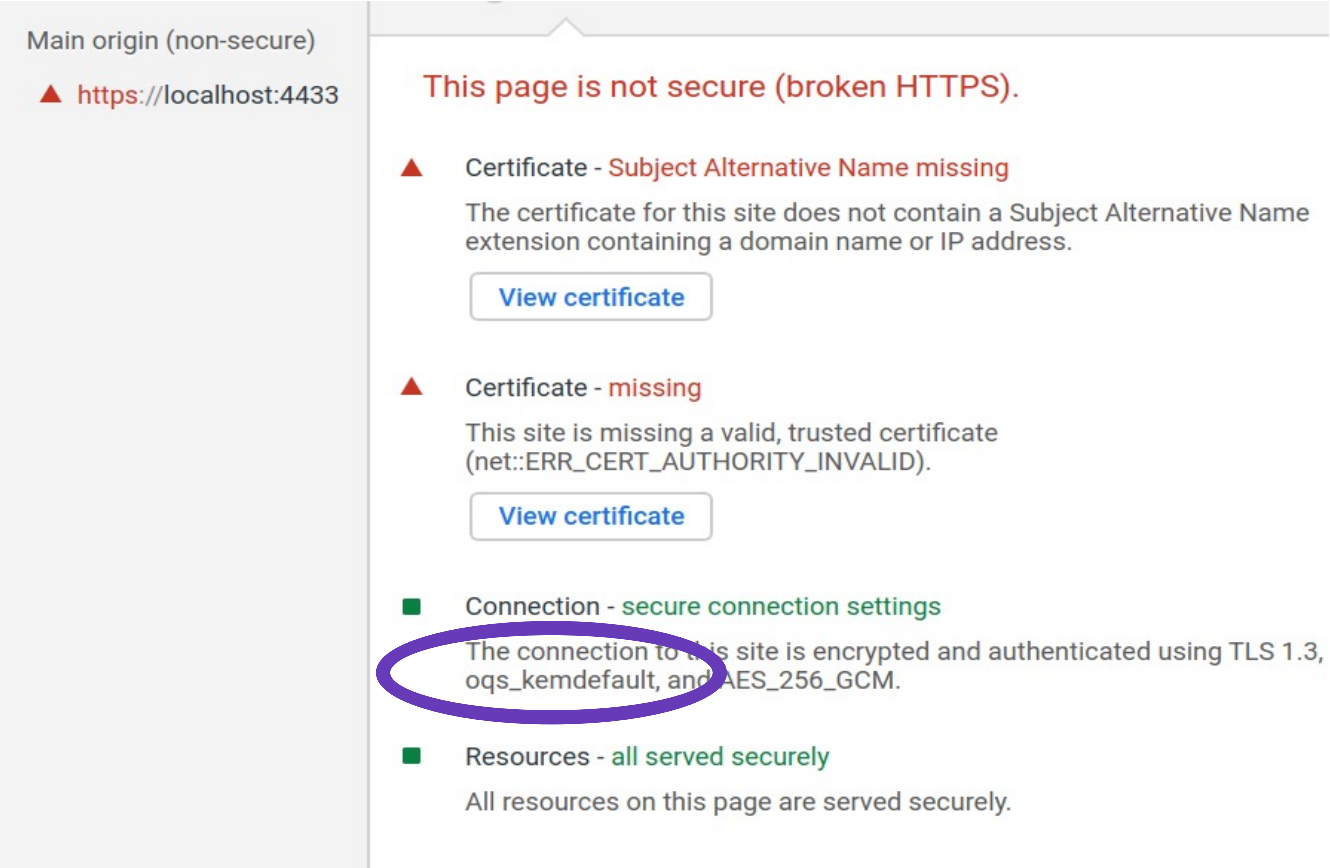
OQS demo: OpenSSL



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!

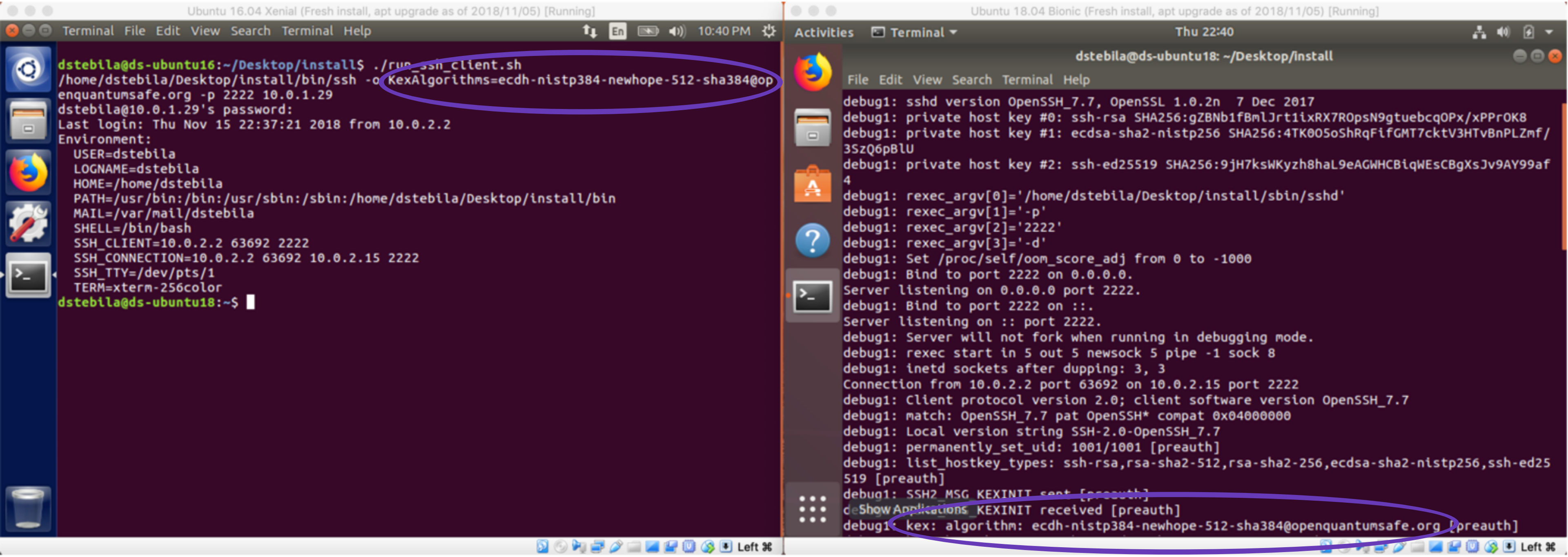
OQS demo: Chromium with BoringSSL talking to Apache



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**. In *PQCrypto 2020*, to appear. https://eprint.iacr.org/2019/1447

Goals

- 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 network bandwidth / throughput

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

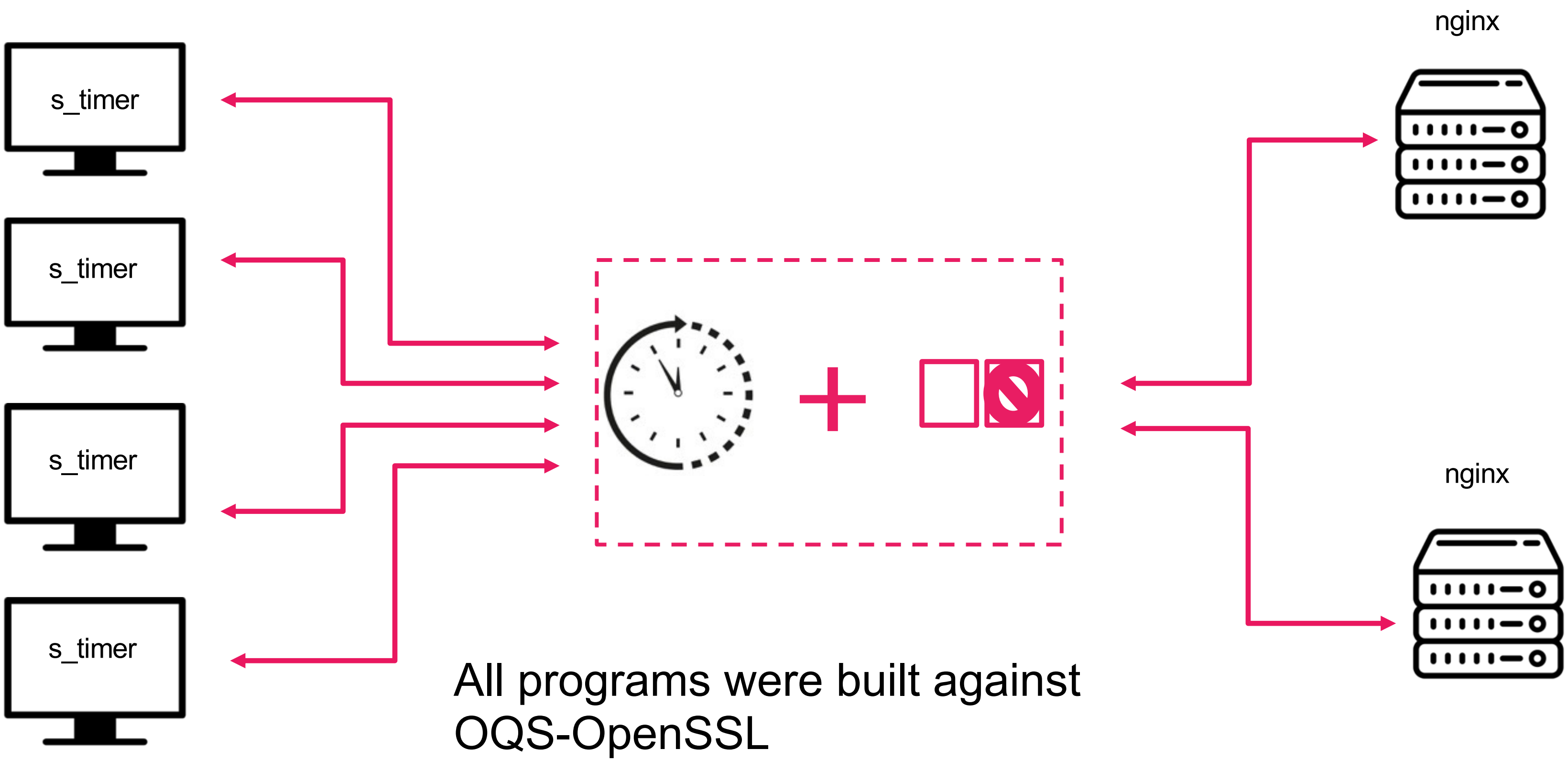
+ easier to isolate effects of network characteristics

– loss in realism

Network emulation setup

- Linux kernel **network namespaces**
  - Independent copies of the kernel's network stack, each having its own routes, addresses, firewall rules, etc.
- **Virtual ethernet devices** created in pairs – one outgoing, one incoming
- **netem (network emulation)** kernel module
  - Can instruct kernel to apply certain delay to packets
  - Can instruct kernel to randomly drop packets with a certain rate

Experiment setup



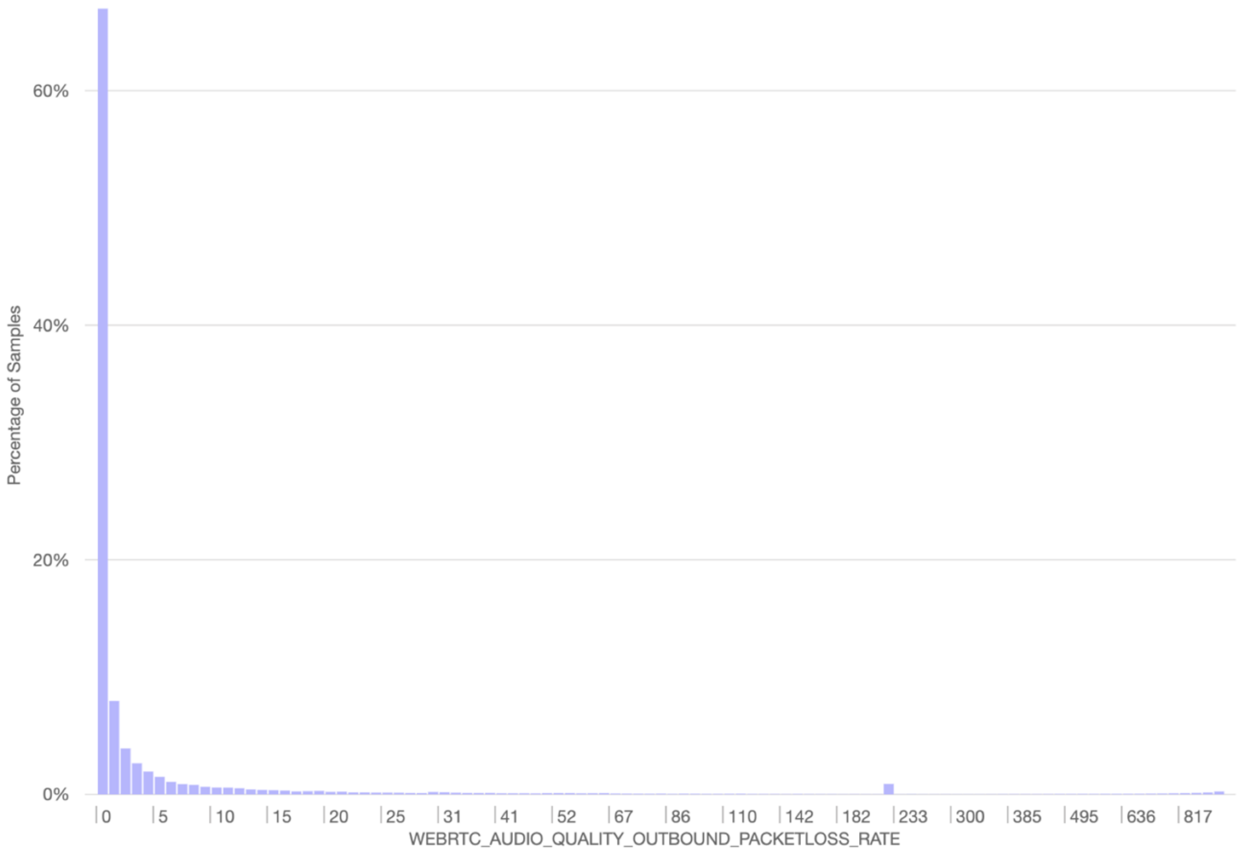
All programs were built against OQS-OpenSSL

Network latencies

Virtual machine	Azure region	Round-trip time
Client	East US 2 (Virginia)	–
Server – near	East US (Virginia)	6.193 ms
Server – medium	Central US (Iowa)	30.906 ms
Server – far	North Europe (Ireland)	70.335 ms
Server – worst-case	Australia East (New South Wales)	198.707 ms

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.

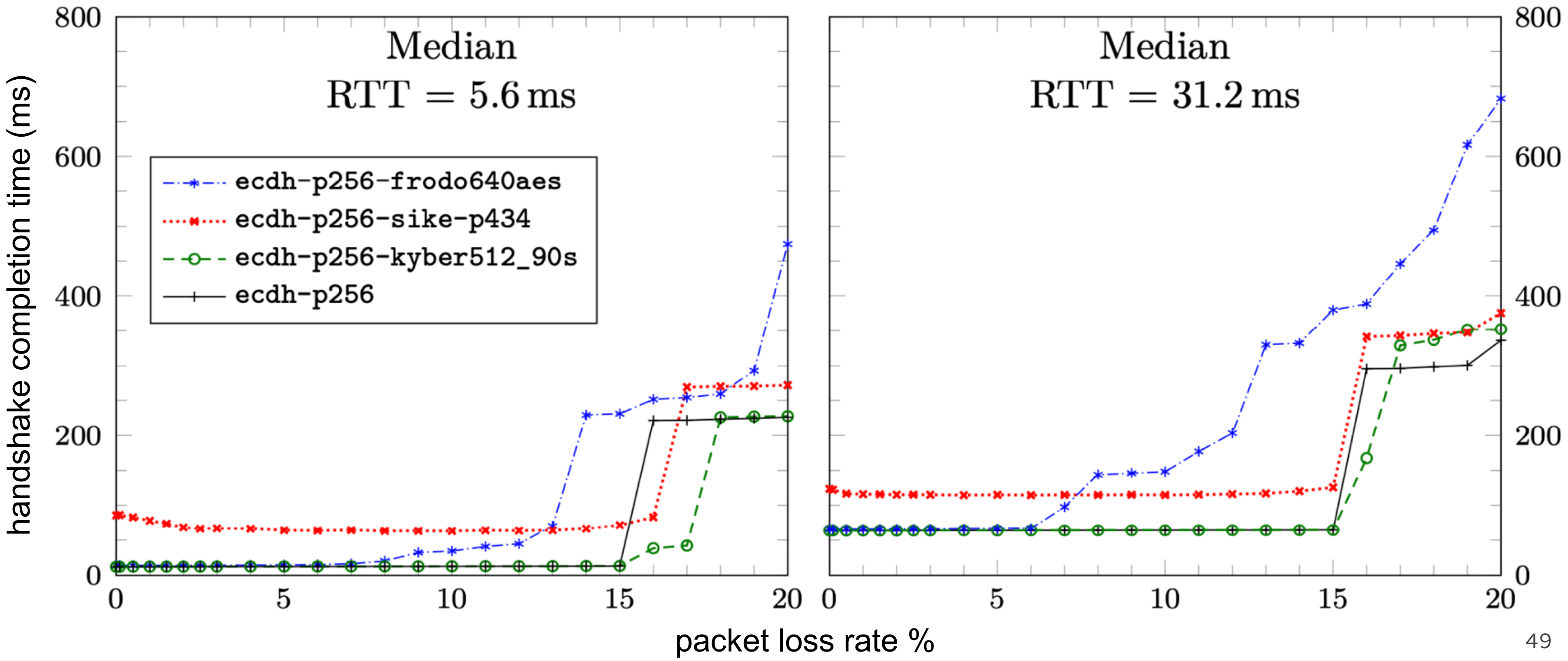
Export CSV Export JSON

https://telemetry.mozilla.org

Algorithms in experiment

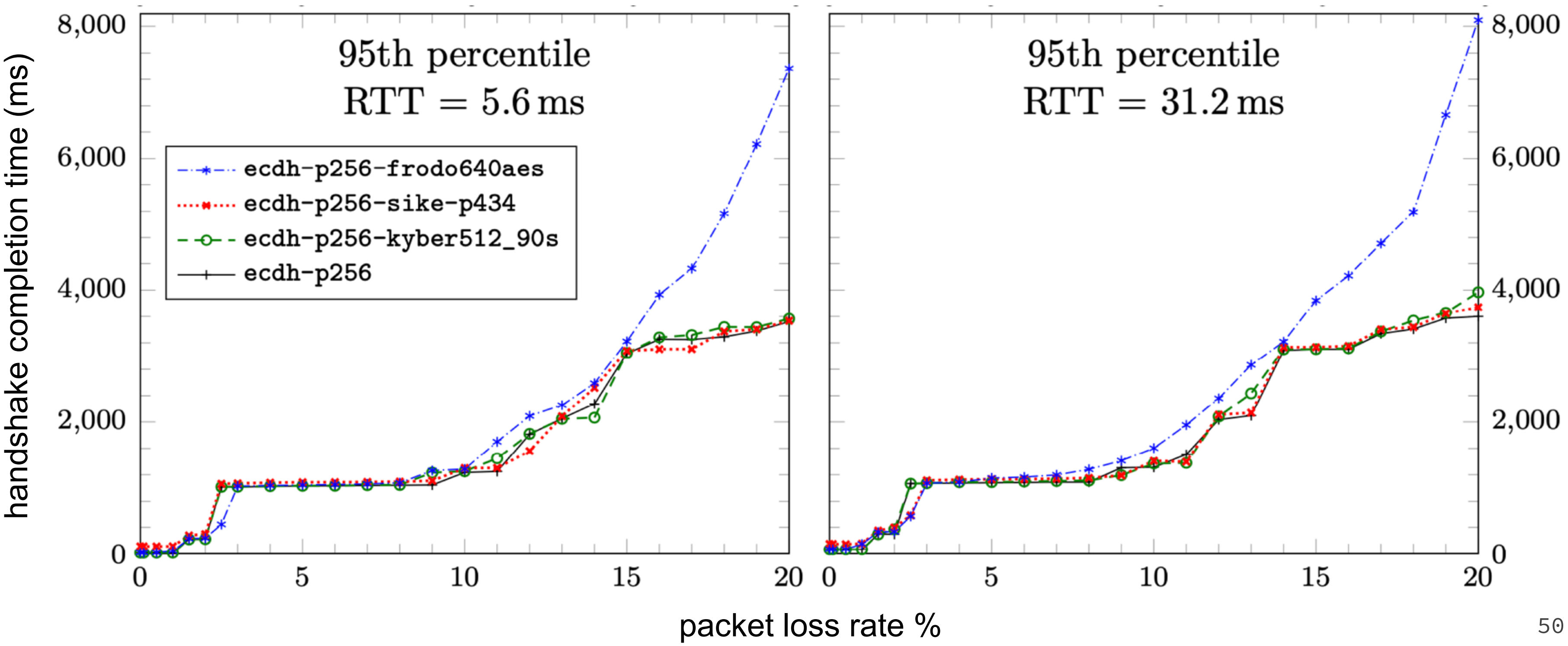
Notation	Hybrid	Family	Variant	Implementation
Key exchange				
`ecdh-p256`	×	Elliptic-curve	NIST P-256	OpenSSL optimized
`ecdh-p256-sike-p434`	✓	Supersingular isogeny	SIKE p434 [JAC+19]	Assembly optimized
`ecdh-p256-kyber512_90s`	✓	Module LWE	Kyber 90s level 1 [SAB+19]	AVX2 optimized
`ecdh-p256-frodo640aes`	✓	Plain LWE	Frodo-640-AES [NAB+19]	C with AES-NI
Signatures				
`ecdsa-p256`	×	Elliptic curve	NIST P-256	OpenSSL optimized
`dilithium2`	×	Module LWE/SIS	Dilithium2 [LDK+19]	AVX2 optimized
`qtesla-p-i`	×	Ring LWE/SIS	qTESLA provable 1 [BAA+19]	AVX2 optimized
`picnic-l1-fs`	×	Symmetric	Picnic-L1-FS [ZCD+19]	AVX2 optimized

Key exchange
median, lower network latencies

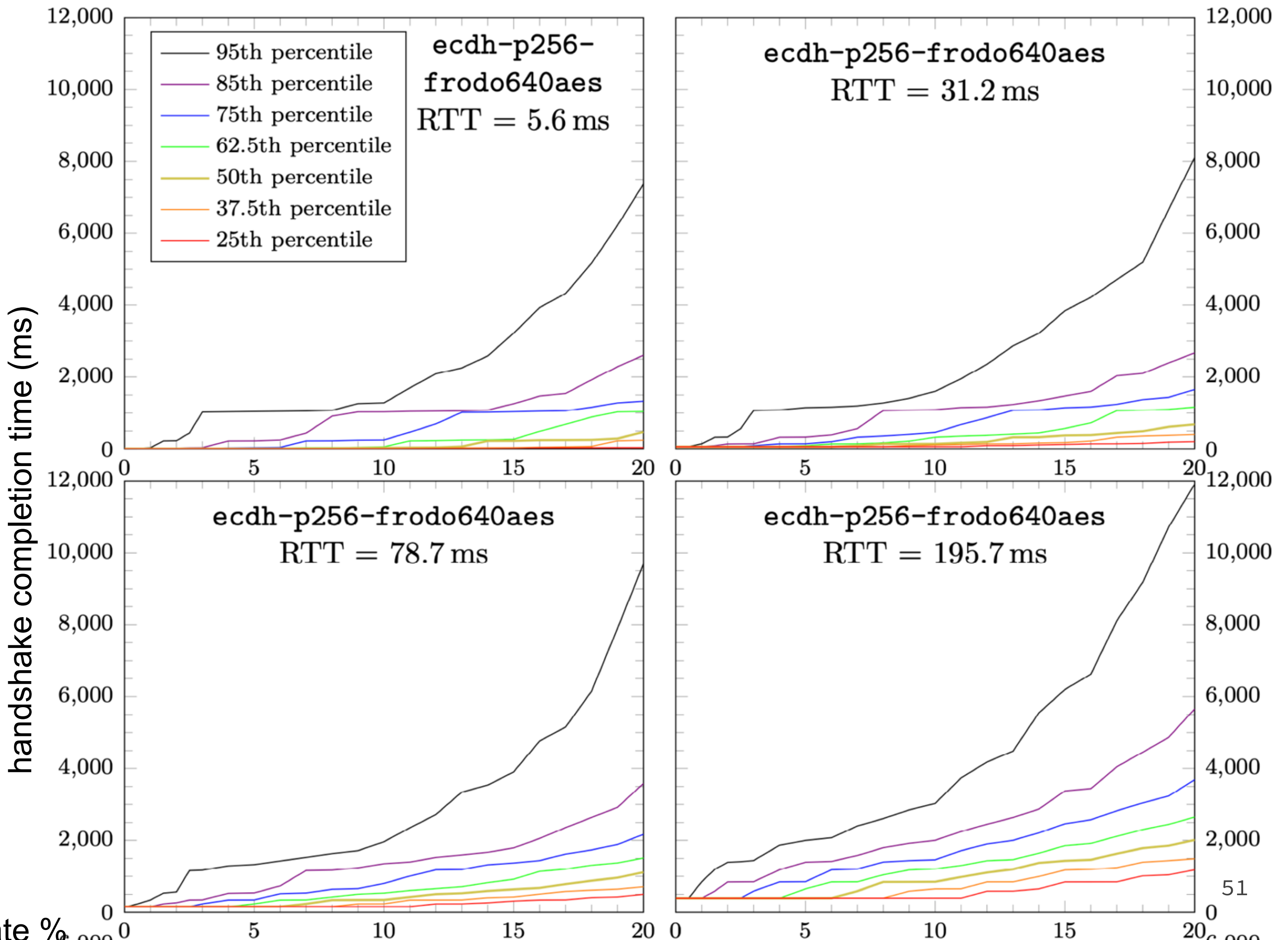


Key exchange

95th percentile, lower network latencies

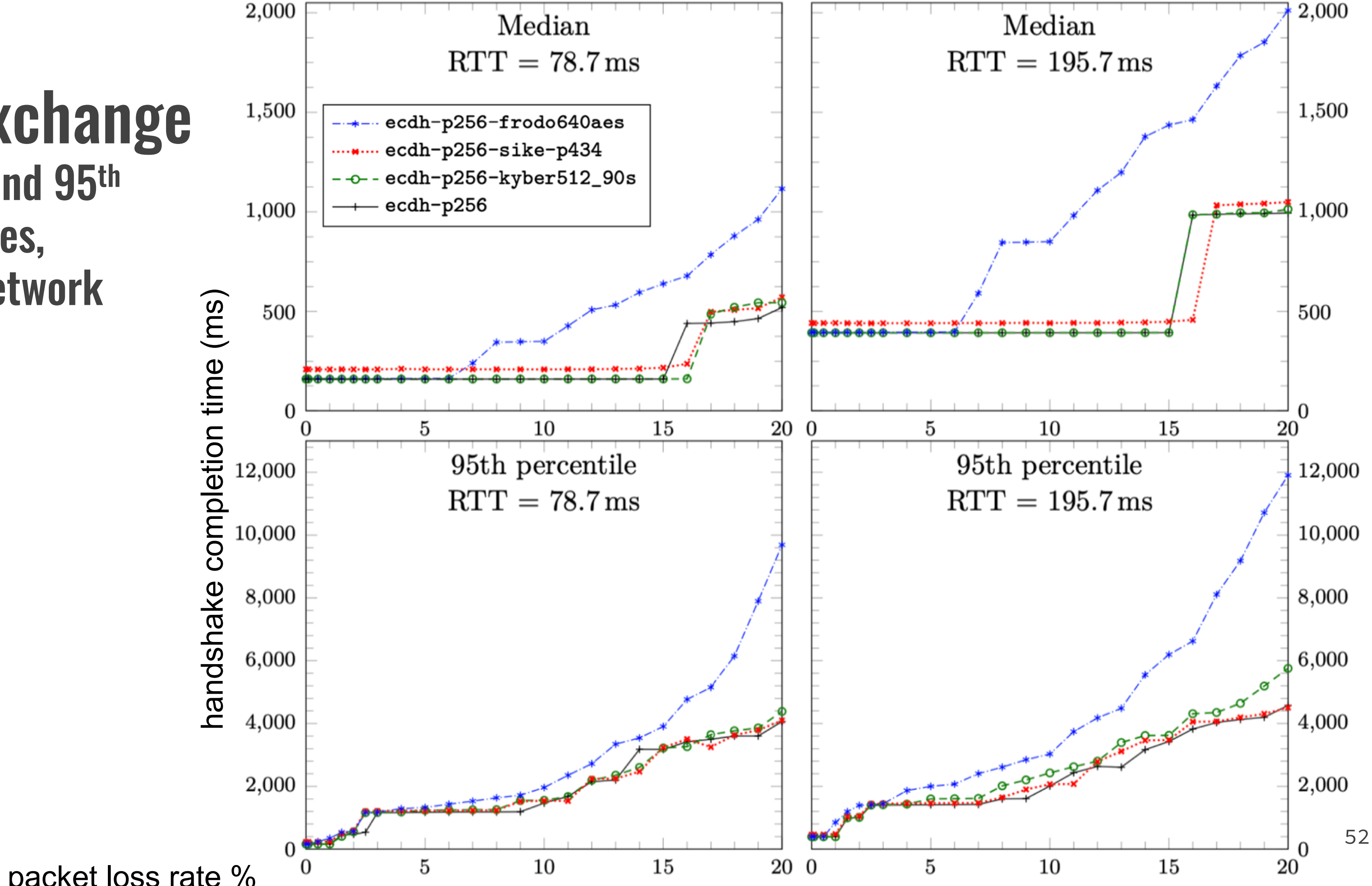


Key exchange
percentiles, FrodoKEM-640-AES



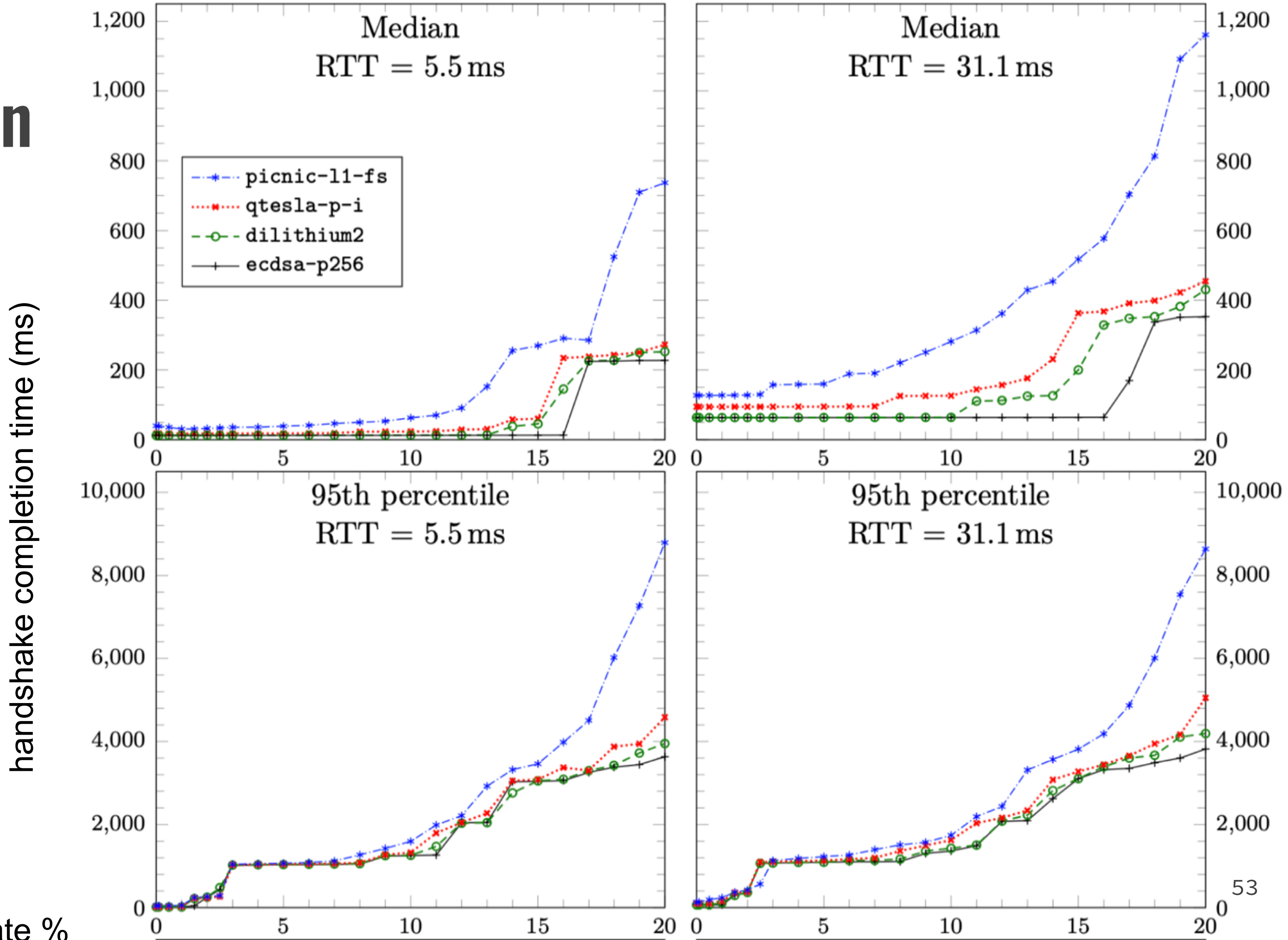
Key exchange

median and 95th percentiles, higher network latencies



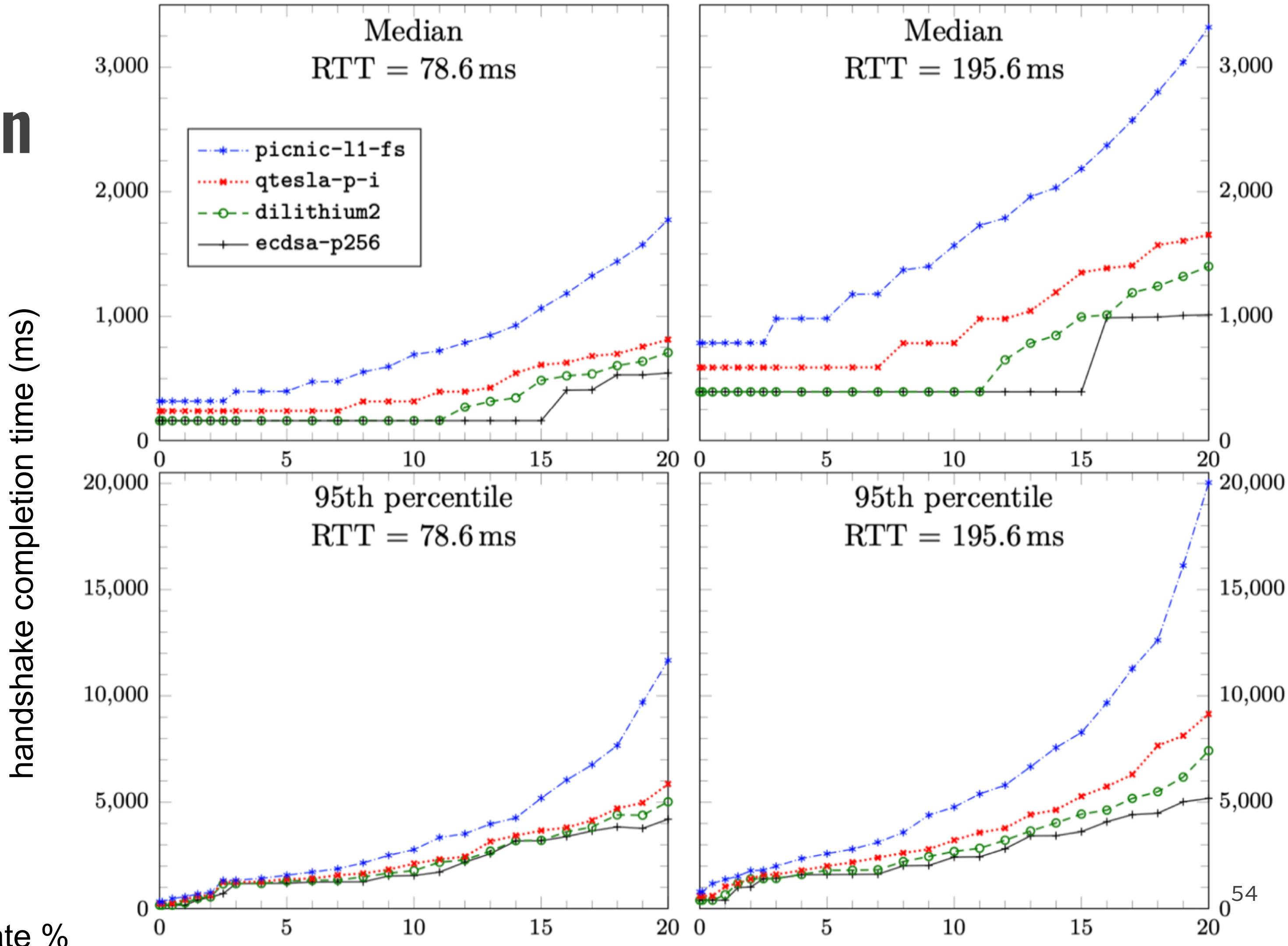
Authentication

median and 95th percentiles, lower network latencies



Authentication

median and 95th percentiles, higher network latencies



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-02

https://openquantumsafe.org/

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

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

Netherlands Crypto Working Group • 2020-02-07

Appendix

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

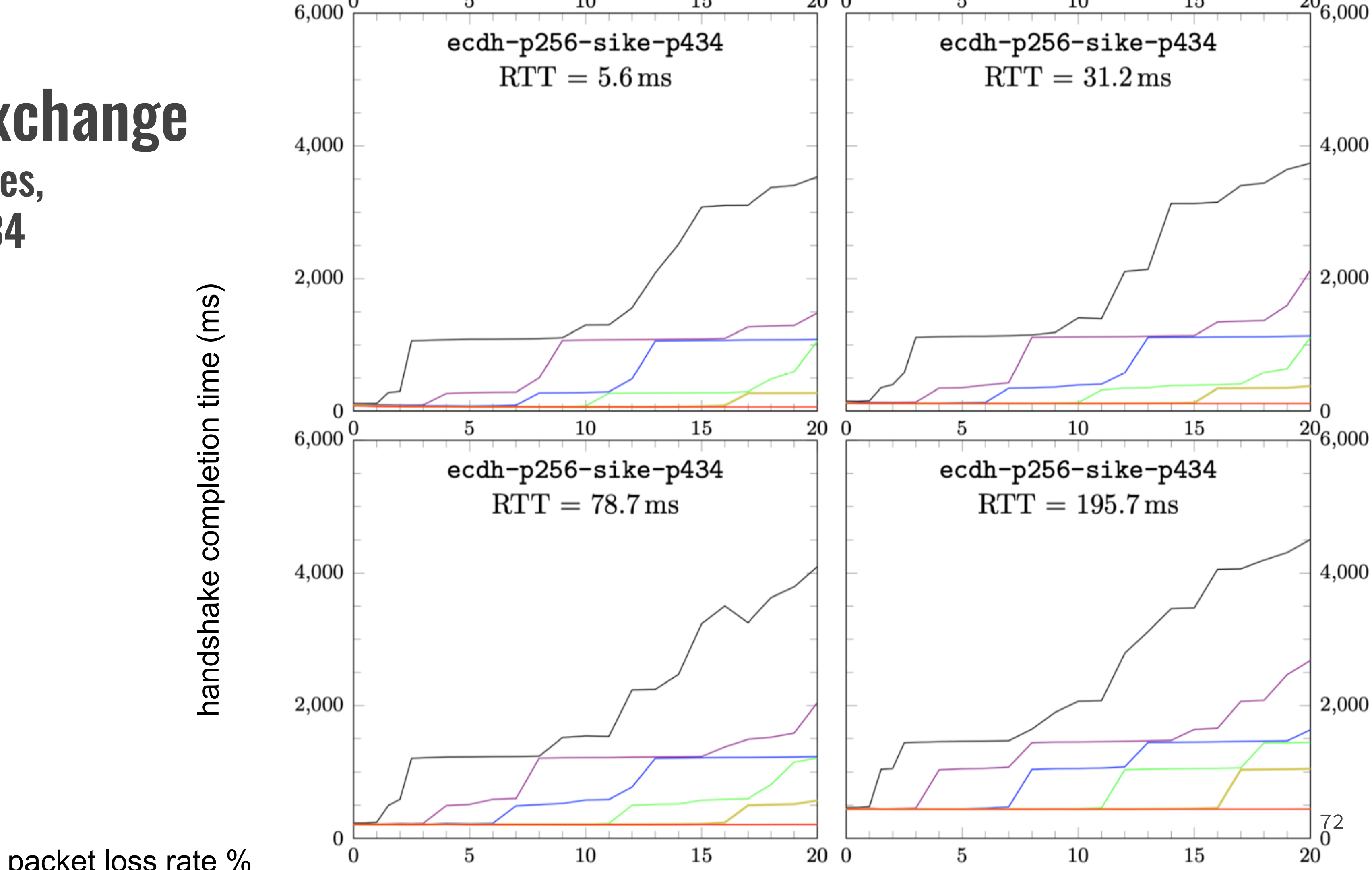
Results in combinatorial explosion of NamedGroups

Duplicate key shares will be sent

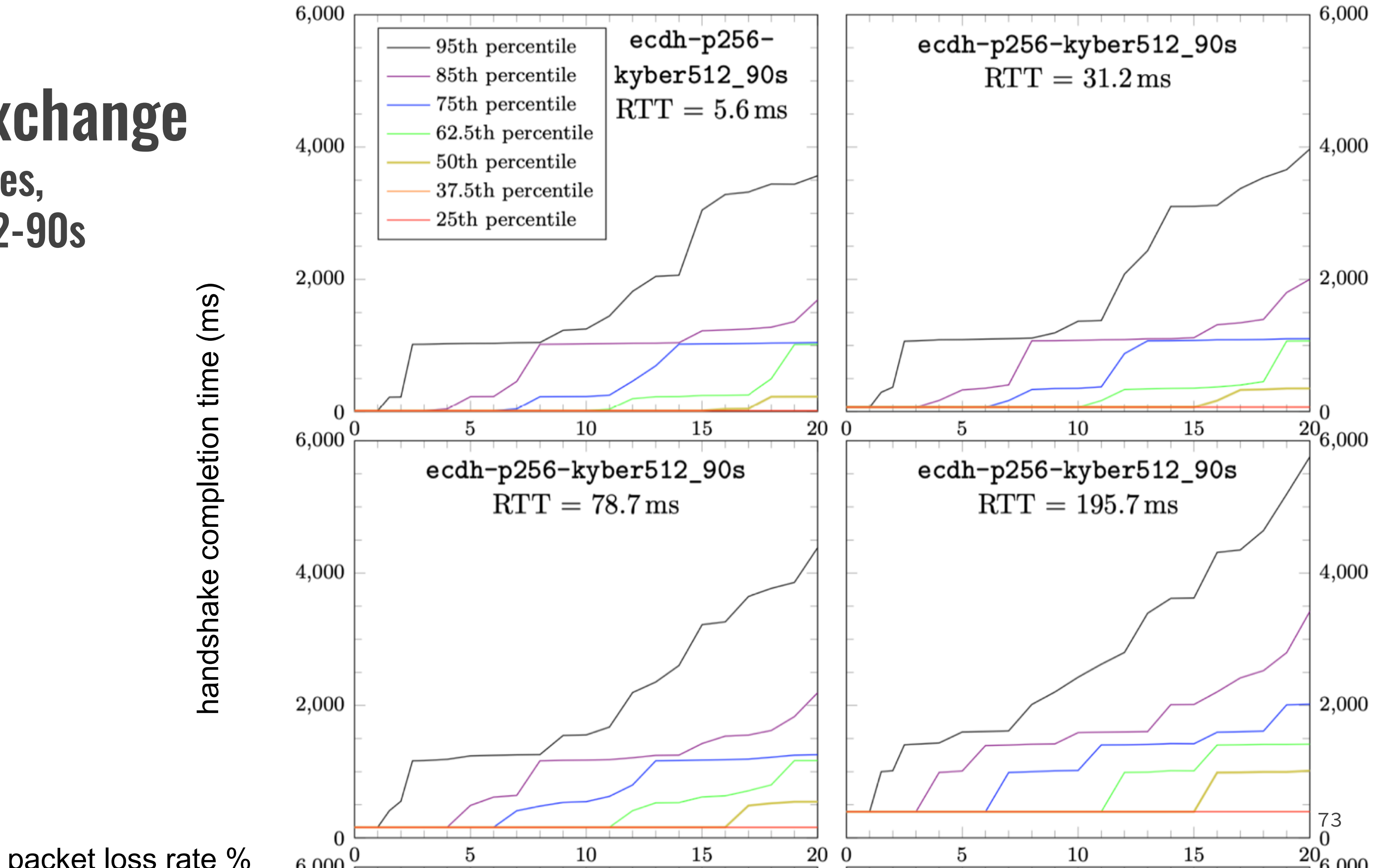
Benchmarking PQ crypto in TLS

Christian Paquin, Douglas Stebila, Goutam Tamvada. **Benchmarking post-quantum cryptography in TLS**. In *PQCrypto 2020*, to appear. https://eprint.iacr.org/2019/1447

Key exchange
percentiles, SIKE-p434

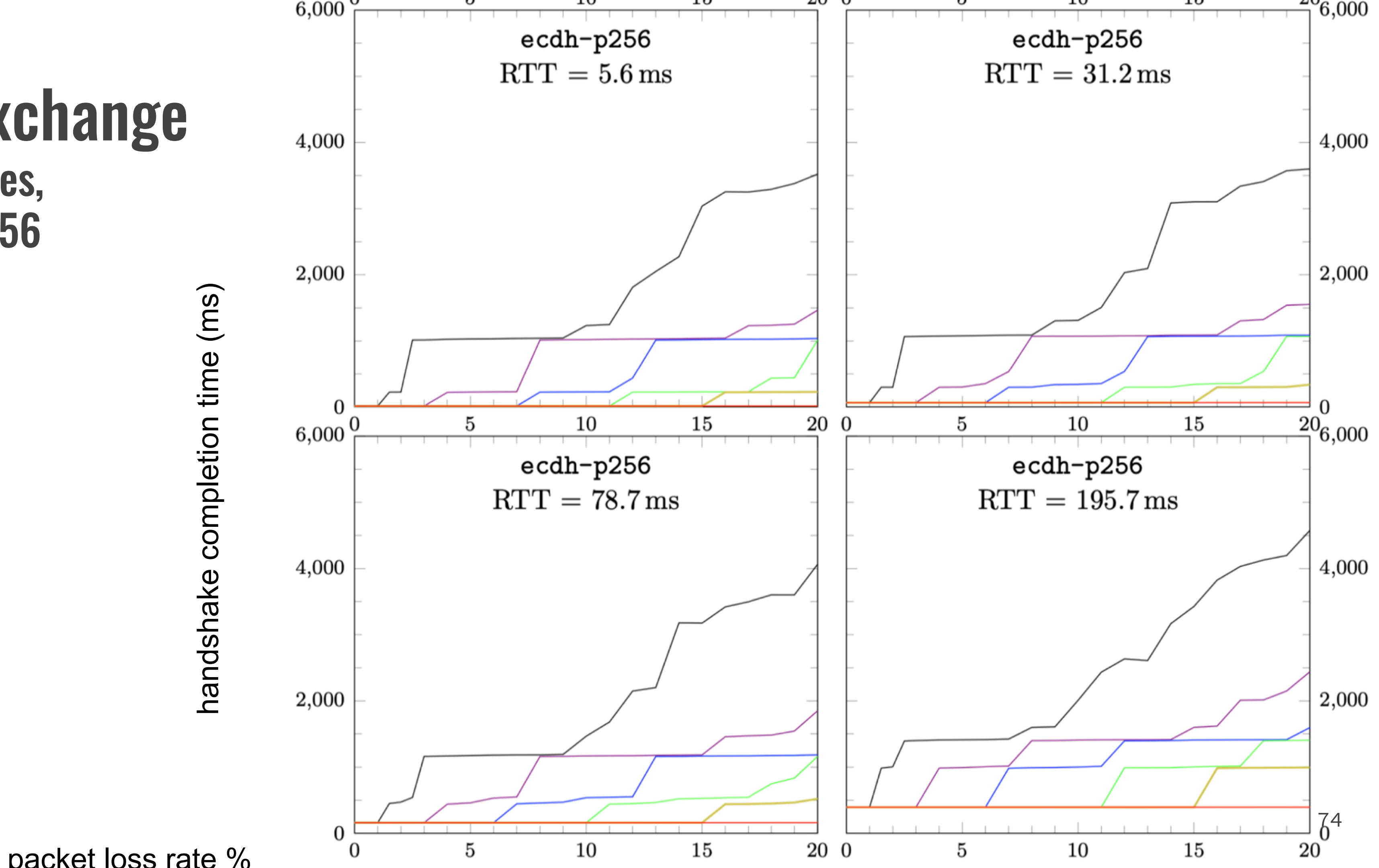


Key exchange
percentiles, Kyber512-90s

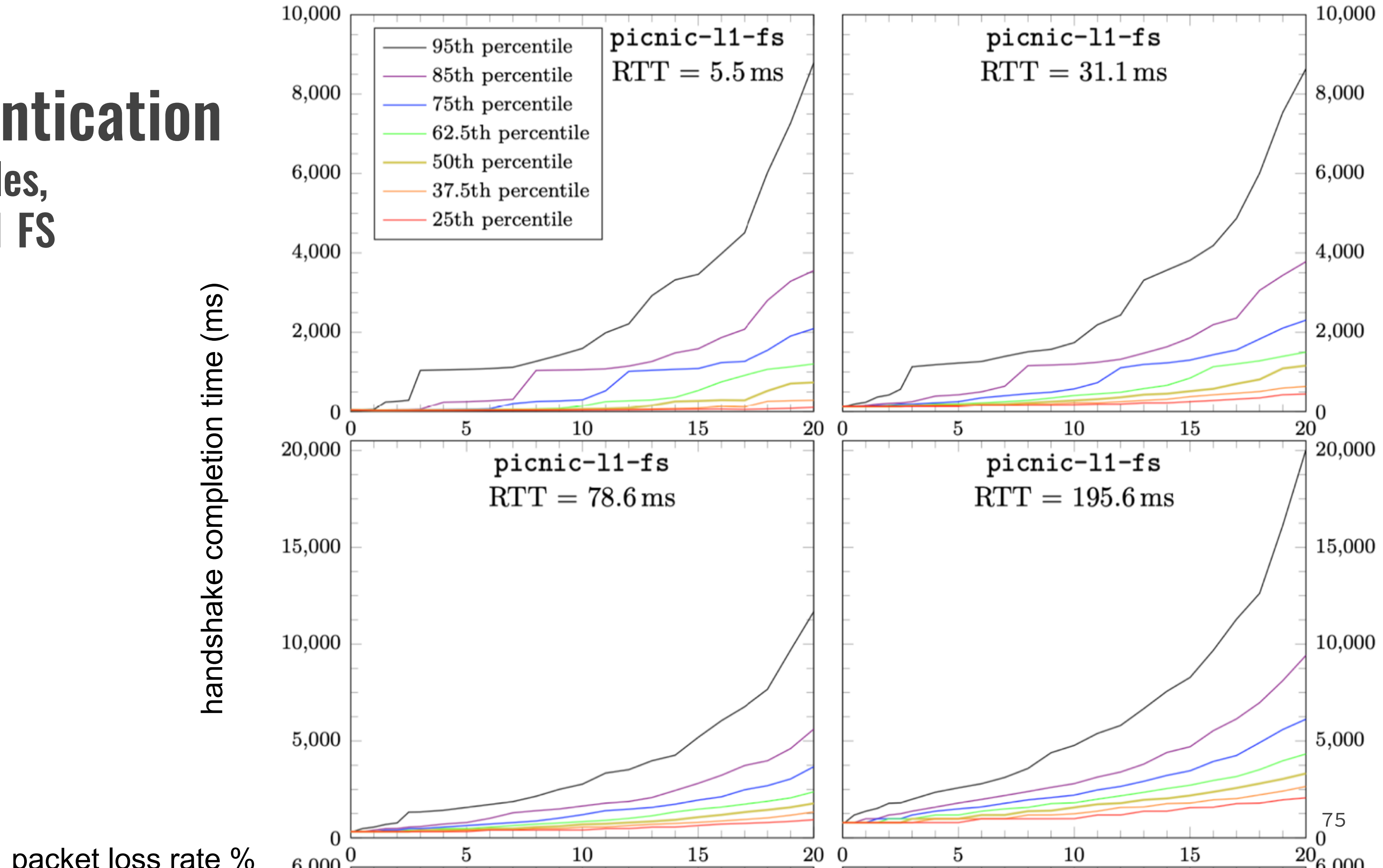


Key exchange

percentiles, ECDH-p256

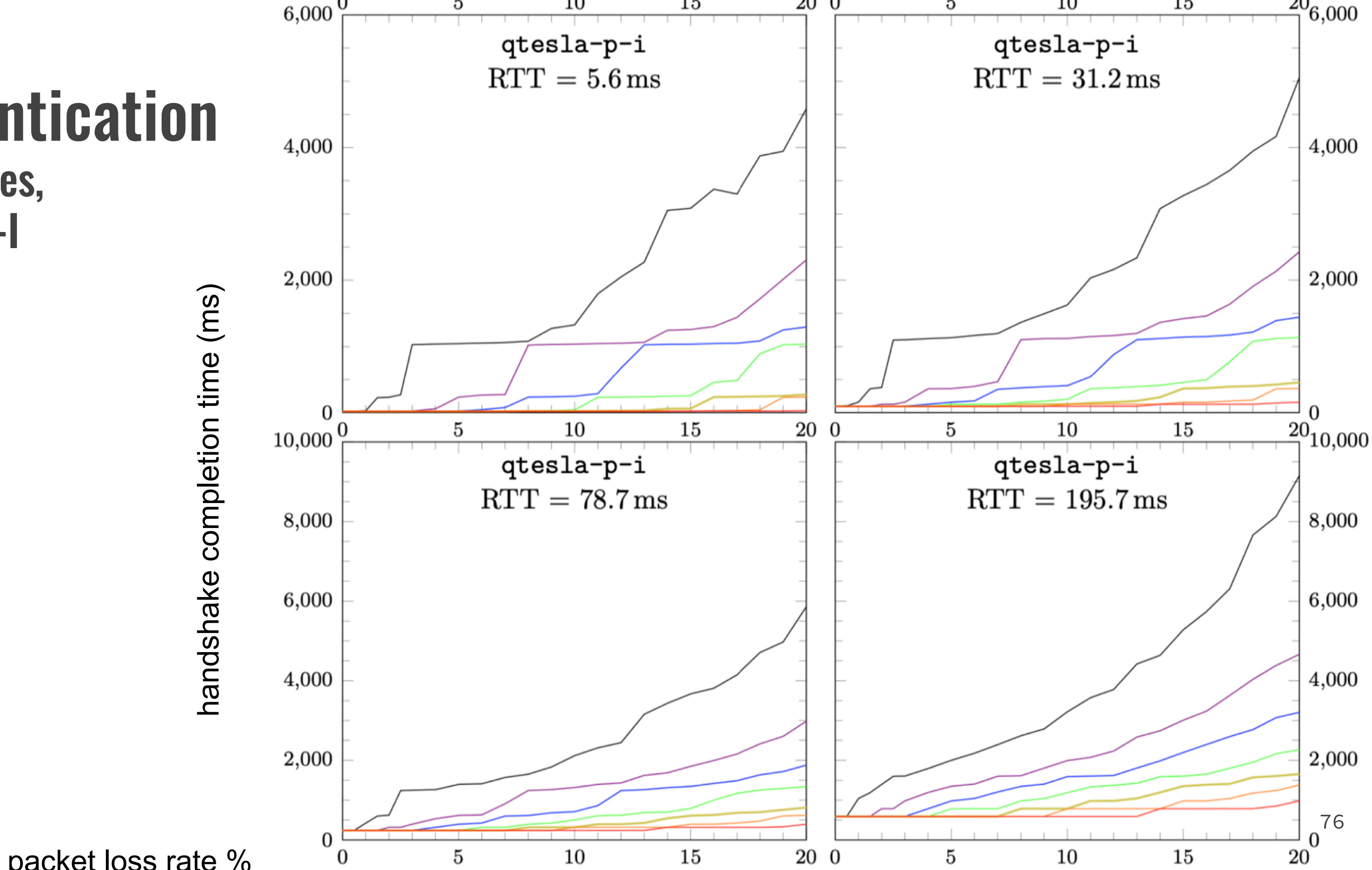


Authentication
percentiles,
Picnic L1 FS

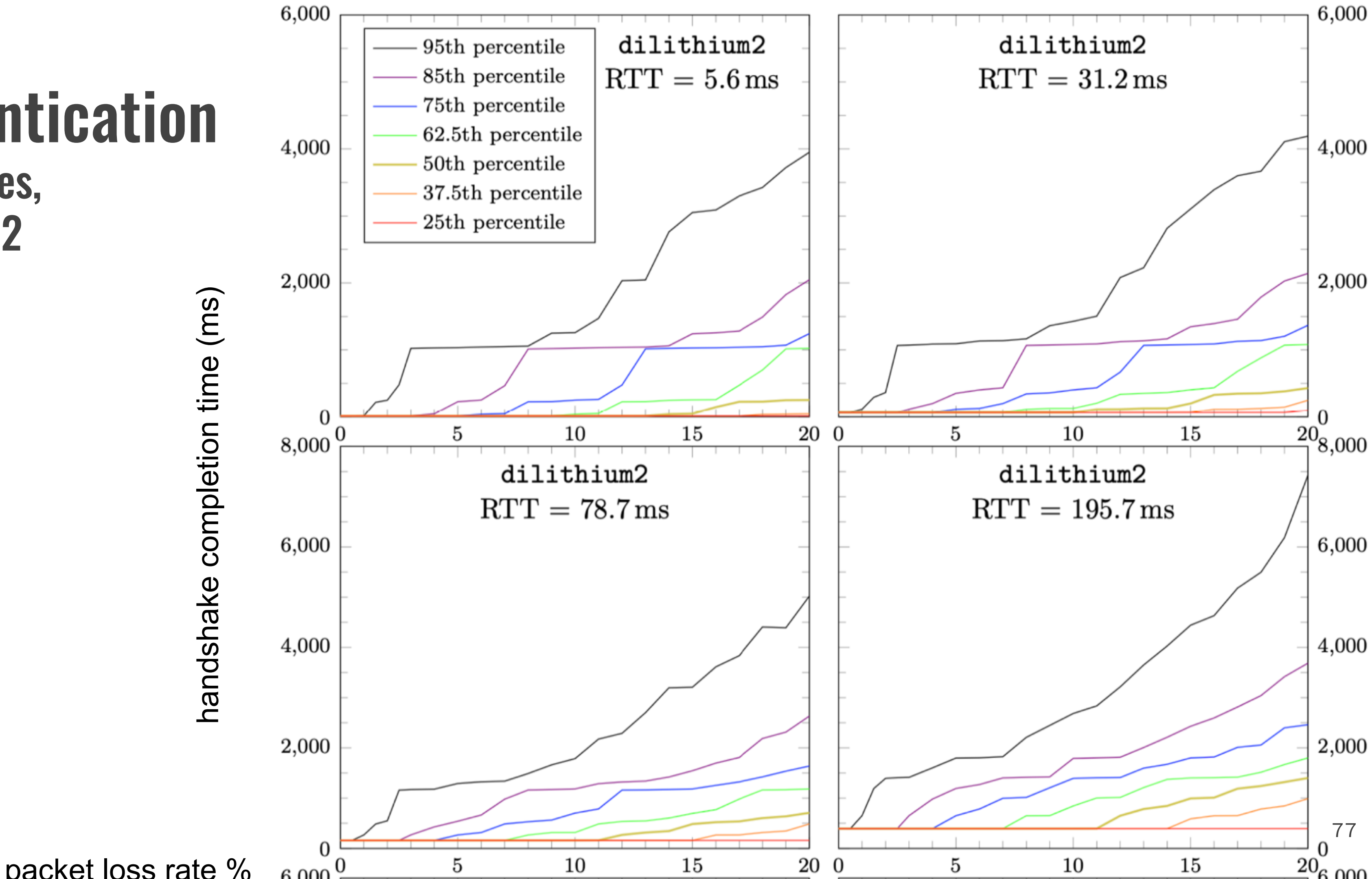


Authentication

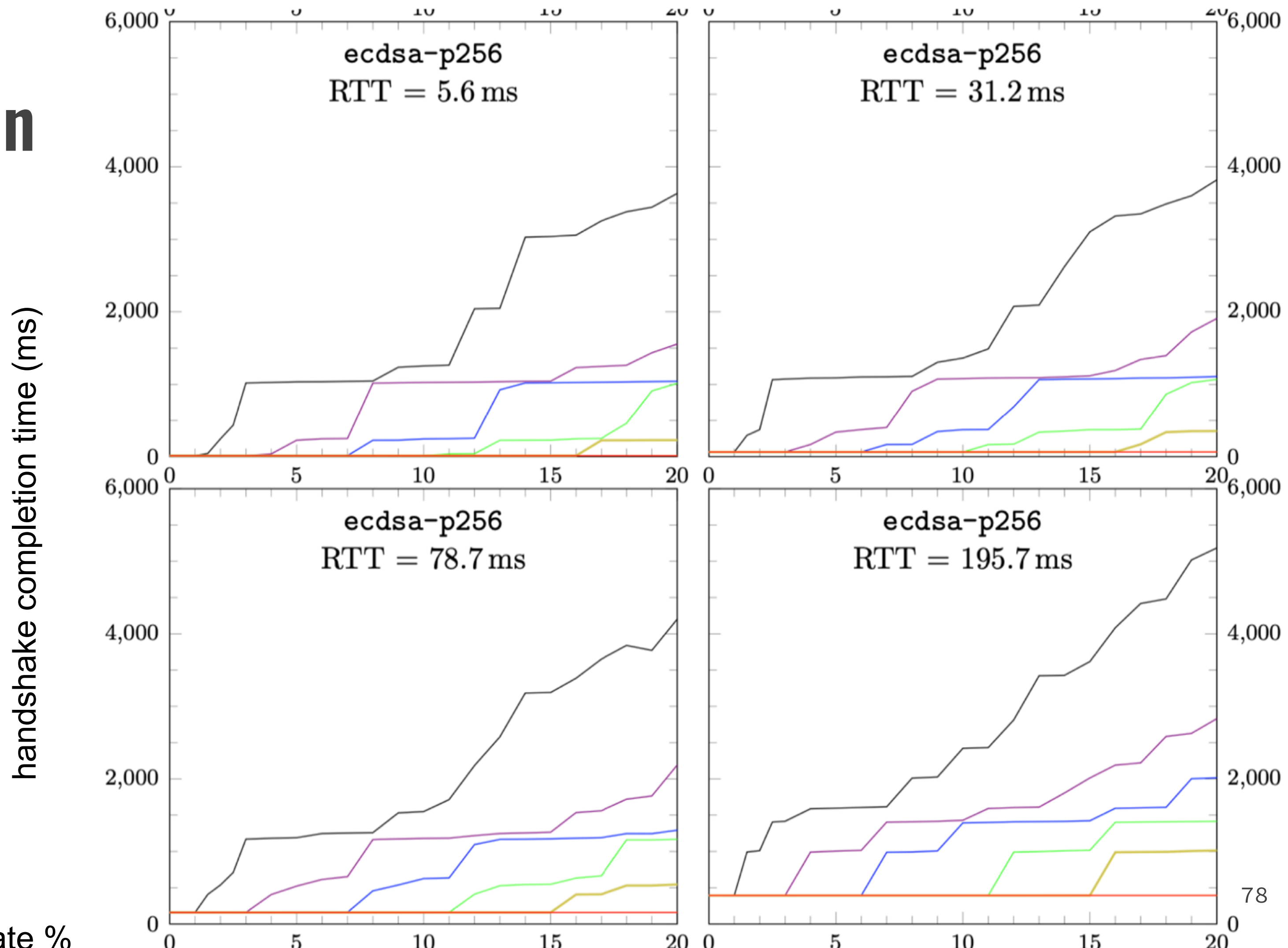
percentiles, qTesla-P-I



Authentication
percentiles, Dilithium2

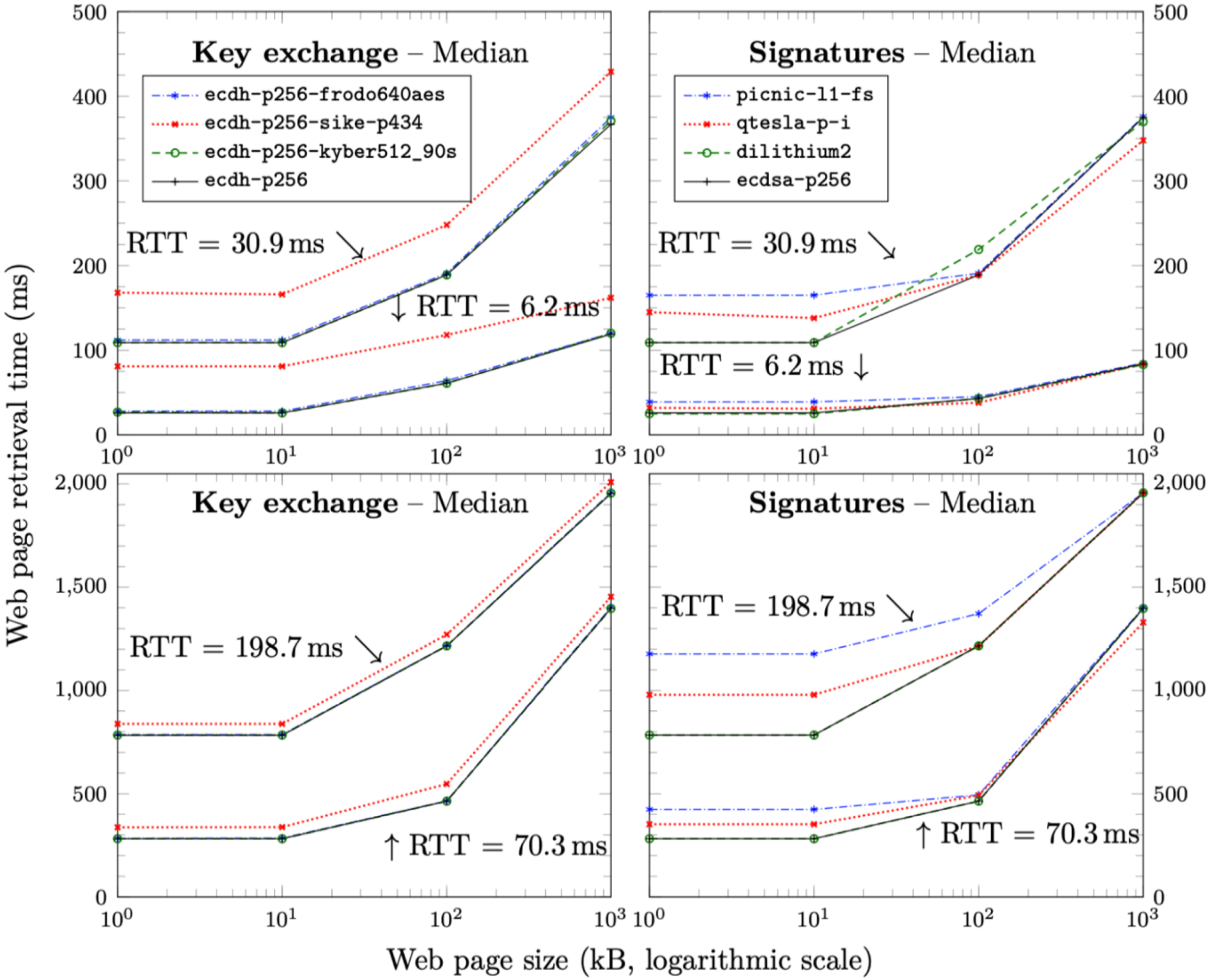


Authentication
percentiles, ECDSA-p256



Data-centre-to-data-centre

web page latency as a function of page size, median



Data-centre-to-data-centre

web page latency as a function of page size, 95th percentile