Rethinking Internet protocols for post-quantum cryptography

Douglas Stebila

University of Waterloo

NSERC CRSNG

Virginia Tech • 2023-02-21

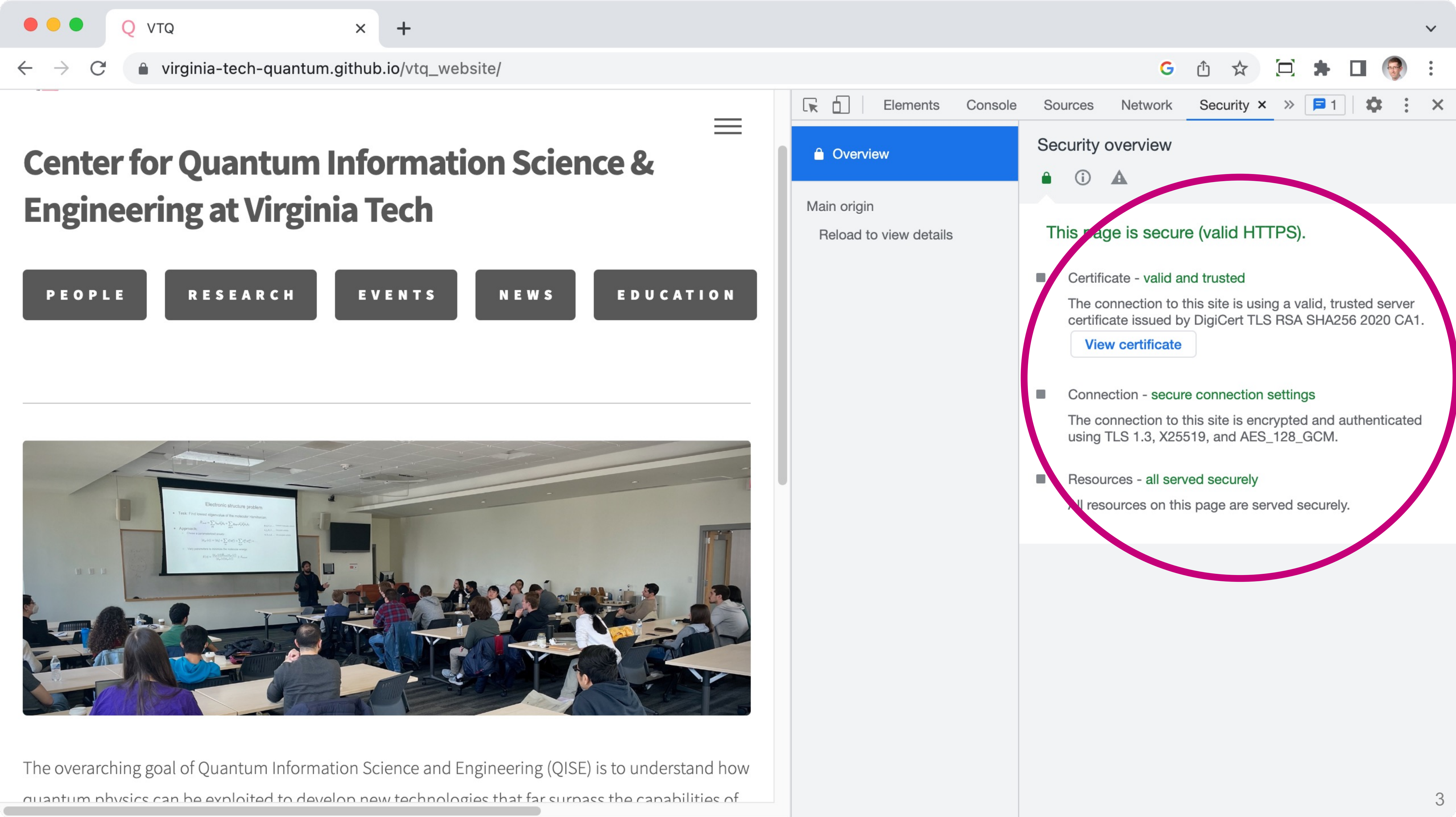
VTQ

virginia-tech-quantum.github.io/vtq_website/

VTQ

Center for Quantum Information Science & Engineering at Virginia Tech

PEOPLE | RESEARCH | EVENTS | NEWS | EDUCATION | OUTREACH | OPENINGS

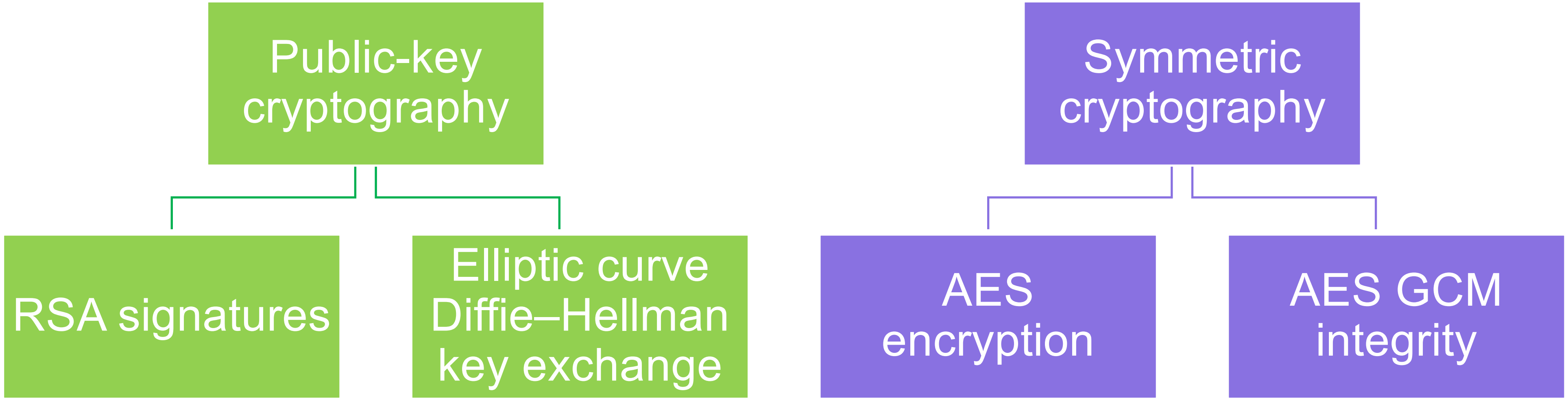

VTQ
virginia-tech-quantum.github.io/vtq_website/
Center for Quantum Information Science & Engineering at Virginia Tech
PEOPLE
RESEARCH
EVENTS
NEWS
EDUCATION
The overarching goal of Quantum Information Science and Engineering (QISE) is to understand how
Elements
Console
Sources
Network
Security
Overview
Main origin
Reload to view details
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 DigiCert TLS RSA SHA256 2020 CA1.
View certificate
Connection - secure connection settings
The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM.
Resources - all served securely
All resources on this page are served securely.

Certificate - valid and trusted

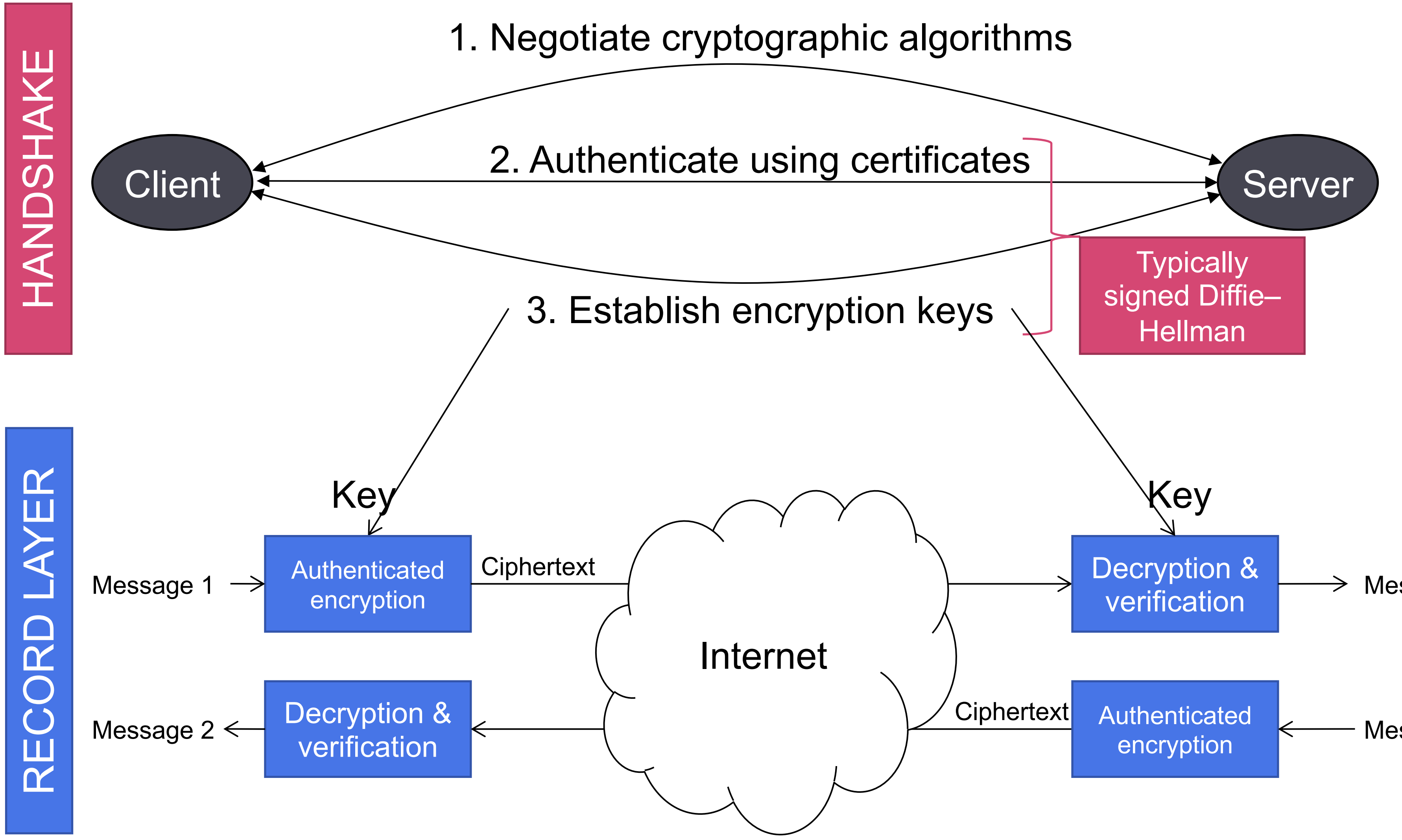
The connection to this site is using a valid, trusted server certificate issued by DigiCert TLS RSA SHA256 2020 CA1.

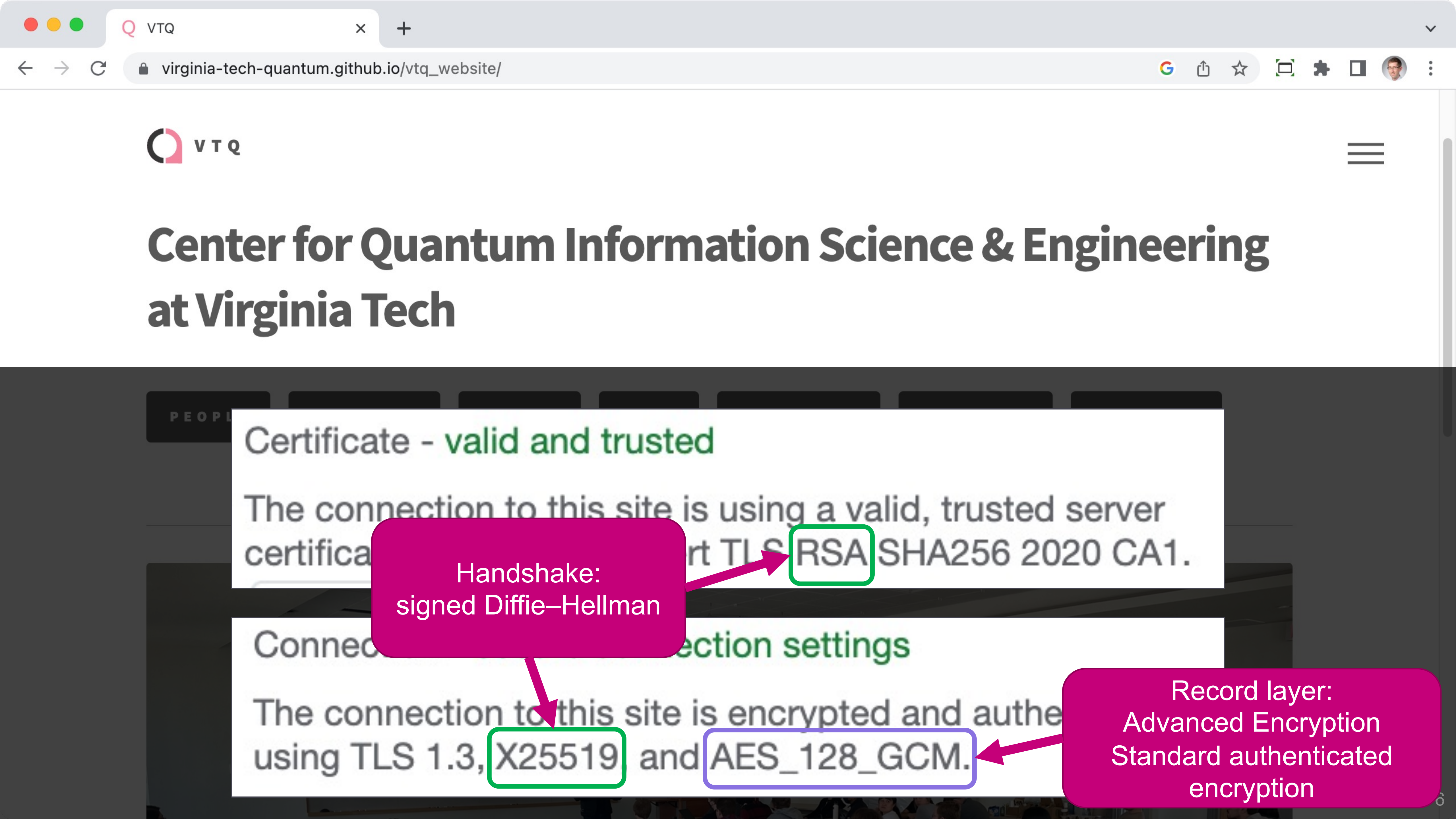
Connection - secure connection settings

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



SSL/TLS Protocol




VTQ
virginia-tech-quantum.github.io/vtq_website/
VTQ
Center for Quantum Information Science & Engineering at Virginia Tech
Certificate - valid and trusted
The connection to this site is using a valid, trusted server
certifica
rt TLS RSA SHA256 2020 CA1.
Handshake:
signed Diffie–Hellman
Connec
ection settings
The connection to this site is encrypted and authe
using TLS 1.3, X25519 and AES_128_GCM.
Record layer:
Advanced Encryption Standard authenticated encryption

Diffie–Hellman key exchange

Alice		**Bob**
$x \in_R \mathbb{Z}_q$		$y \in_R \mathbb{Z}_q$
$X \leftarrow g^x$	send X $\longrightarrow$	$Y \leftarrow g^y$
	$\leftarrow$ send Y	
$k \leftarrow Y^x = g^{xy}$		$k \leftarrow X^y = g^{xy}$

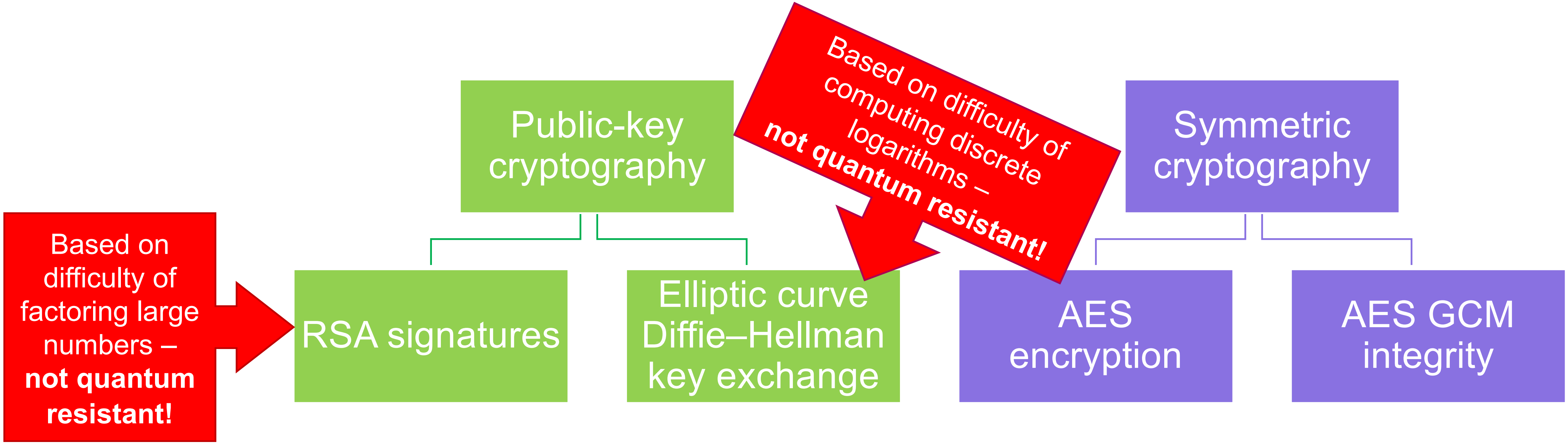
g is a generator of an abelian group of prime order q

Certificate - valid and trusted

The connection to this site is using a valid, trusted server certificate issued by DigiCert TLS RSA SHA256 2020 CA1.

Connection - secure connection settings

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



Post-quantum cryptography

a.k.a. quantum-resistant algorithms

Cryptography based on computational assumptions believed to be resistant to attacks by quantum computers

Uses only classical (non-quantum) operations to implement

Quantum key distribution

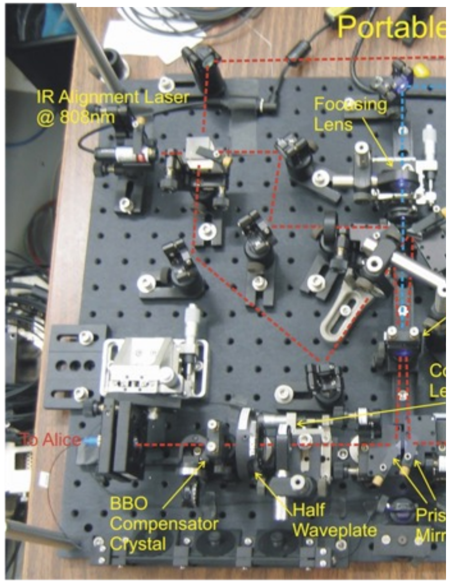
Also provides quantum-resistant confidentiality

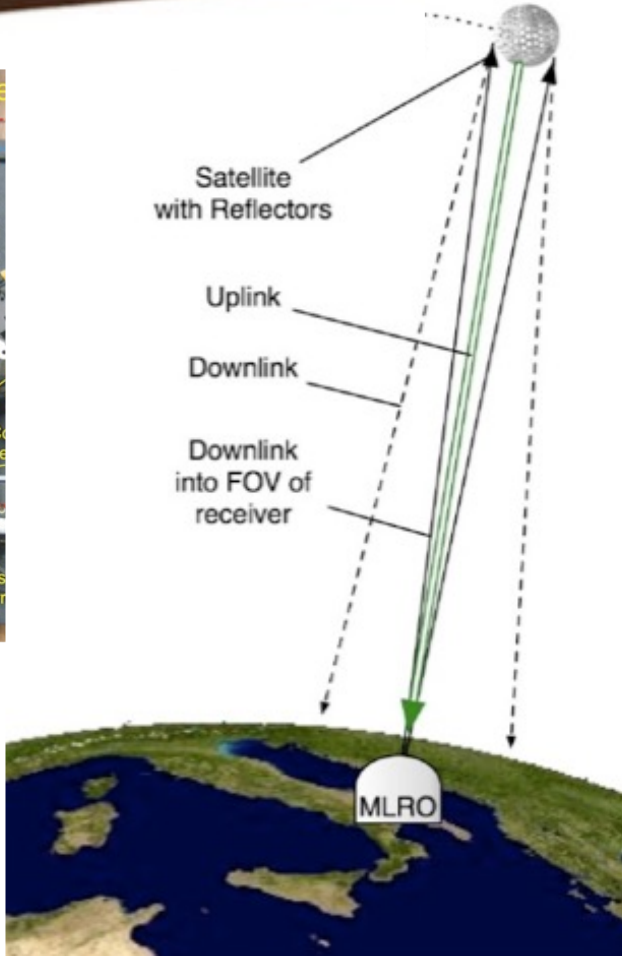
Uses quantum mechanics to protect information

Doesn't require a full quantum computer

But does require quantum communication devices and channels

=> Not the subject of this talk





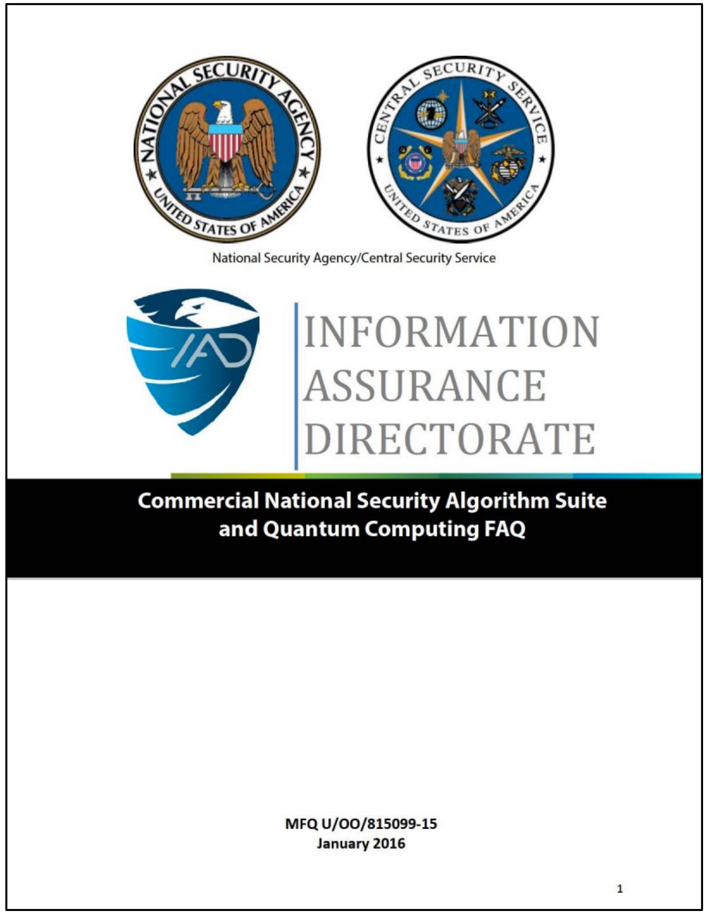
Post-quantum	QKD
Security depends on computational assumptions	Can be information-theoretically secure
Works on existing infrastructure	Requires new devices and communication channels
No limitations on communication distance	Limits on communication distance without new technology (repeaters) or additional trusts assumptions

Outline

1. Status of post-quantum cryptography standardization
2. Making Internet protocols post-quantum
    - Challenges
    - Hybrid
    - New protocol designs
    - New transport designs
3. Next steps

Standardization of PQ cryptography

Standardizing post-quantum cryptography



Aug. 2015 (Jan. 2016)

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

– NSA Information Assurance Directorate, Aug. 2015

Primary goals for post-quantum crypto

Confidentiality in the public key setting

- **Public key encryption schemes**
- Alternatively: key encapsulation mechanisms
  - KEMs are a generalization of two-party Diffie–Hellman-style key exchange
  - Easy to convert KEM into PKE and vice versa

Authentication & integrity in the public key setting

- **Digital signature schemes**

Families of post-quantum cryptography

Hash- & symmetric-based

- Can only be used to make signatures, not public key encryption
- Very high confidence in hash-based signatures, but large signatures required for many signature-systems

Code-based

- Long-studied cryptosystems with moderately high confidence for some code families
- Challenges in communication sizes

Multivariate quadratic

- Variety of systems with various levels of confidence and trade-offs
- Substantial break of Rainbow algorithm in Round 3

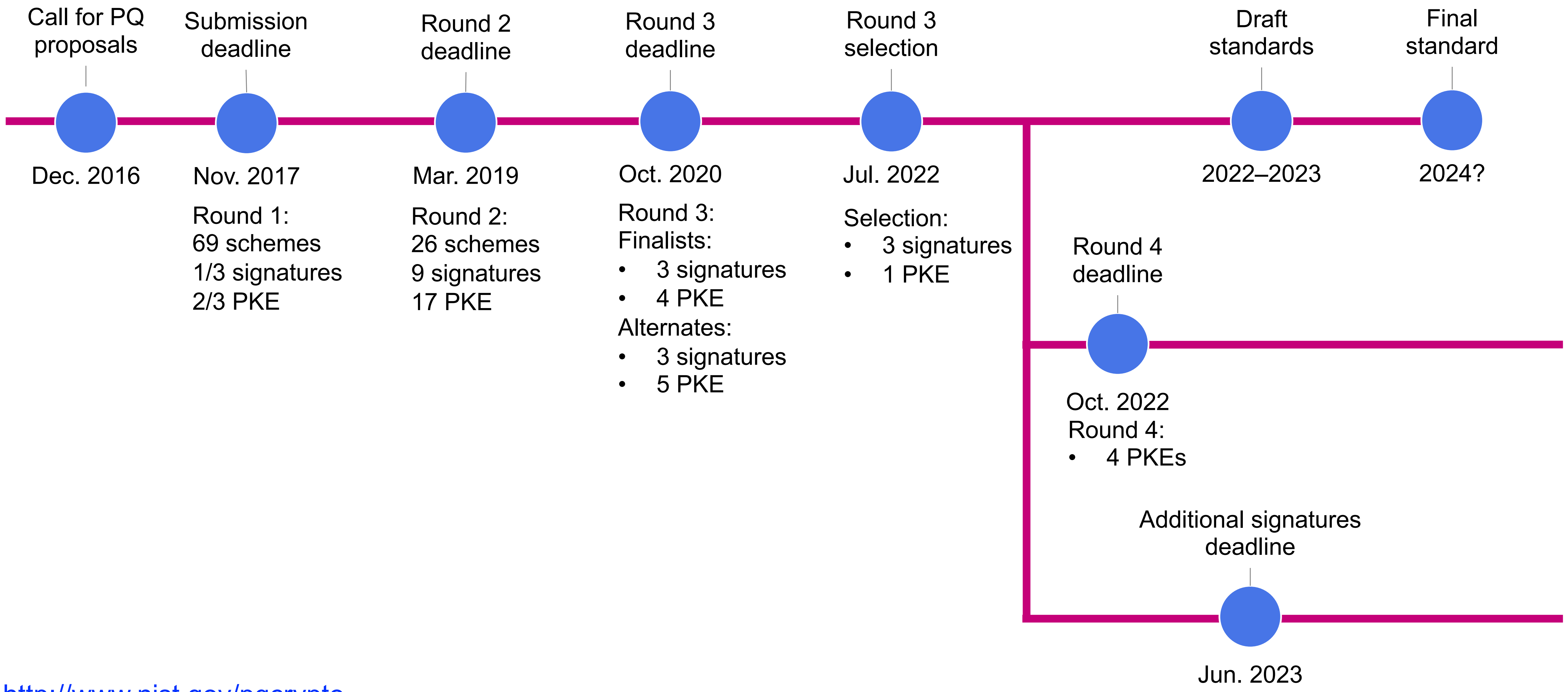
Lattice-based

- High level of academic interest in this field, flexible constructions
- Can achieve reasonable communication sizes

Elliptic curve isogenies

- Newest mathematical construction
- Small communication, slower computation
- Substantial break of SIKE in Round 4

NIST Post-quantum Crypto Project timeline



http://www.nist.gov/pqcrypto

NIST Round 3 selections and Round 4

Selections

Key encapsulation mechanisms

- Lattice-based: Kyber

Signatures

- Lattice-based: Dilithium, Falcon
- Hash-based: SPHINCS+

Round 4

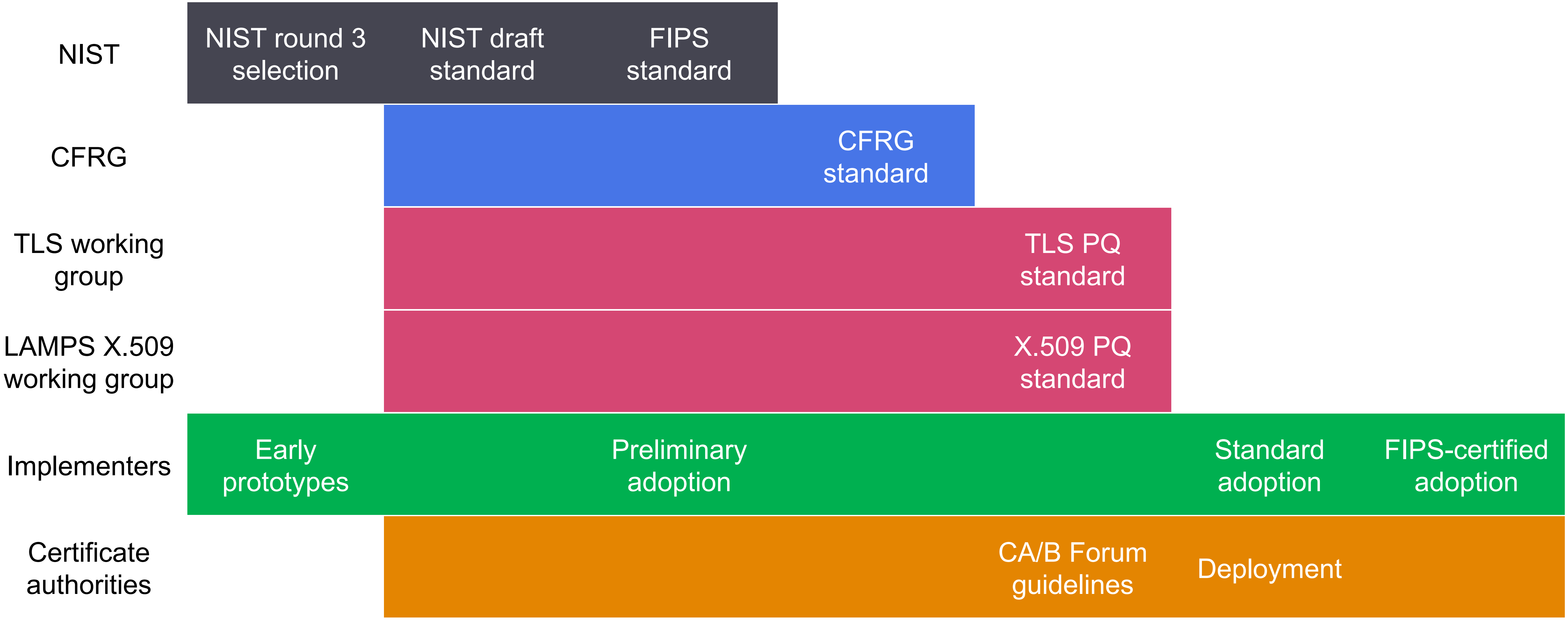
Key encapsulation mechanisms

- Code-based: BIKE, Classic McEliece, HQC
- ~~Isogeny-based: SIKE~~

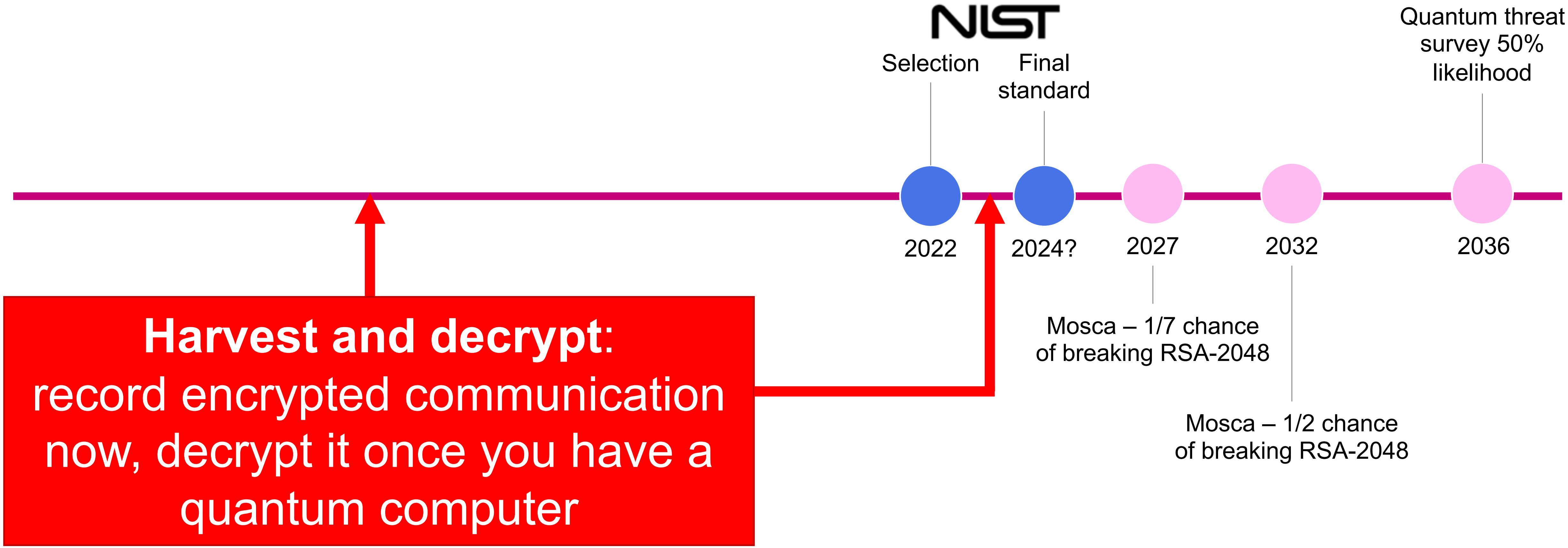
Signatures

- Call for additional signature schemes

Paths to standardization and adoption

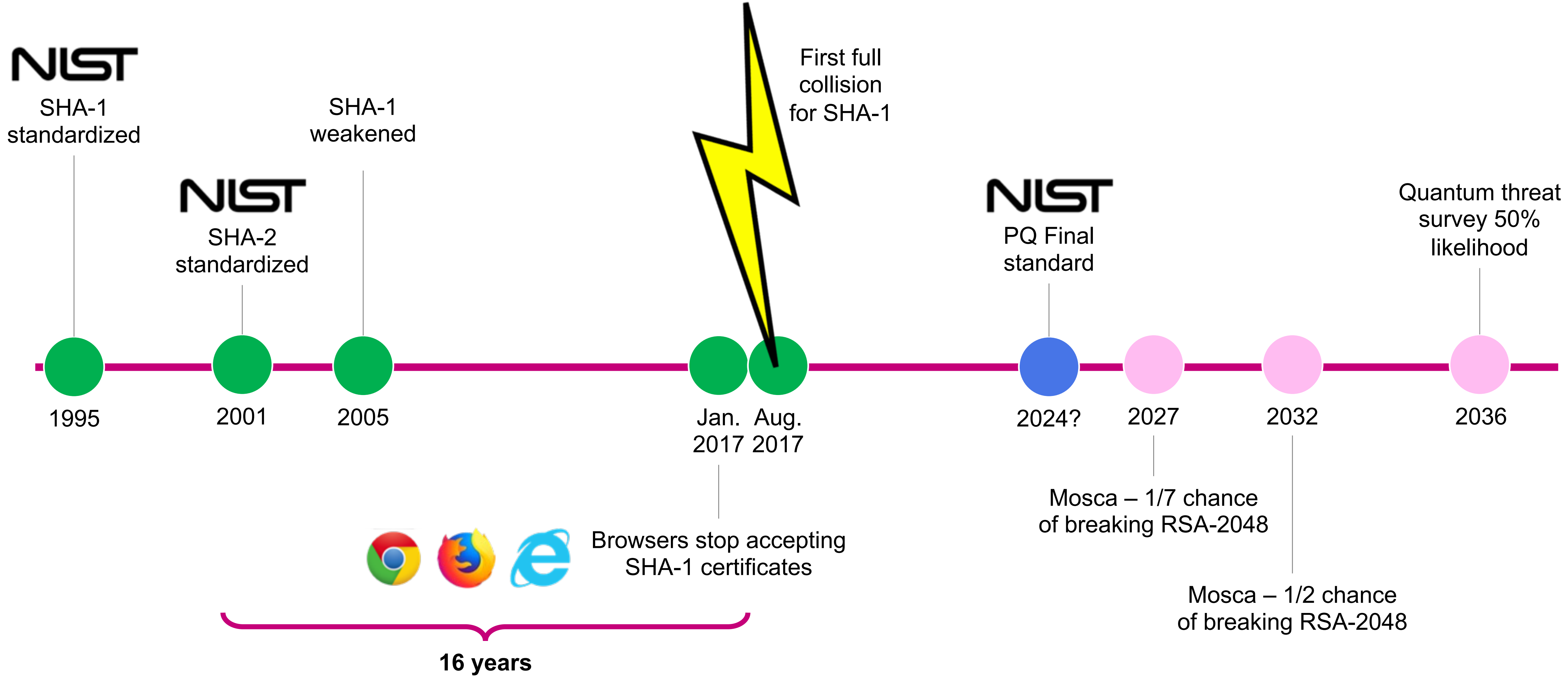


Will we be ready in time?



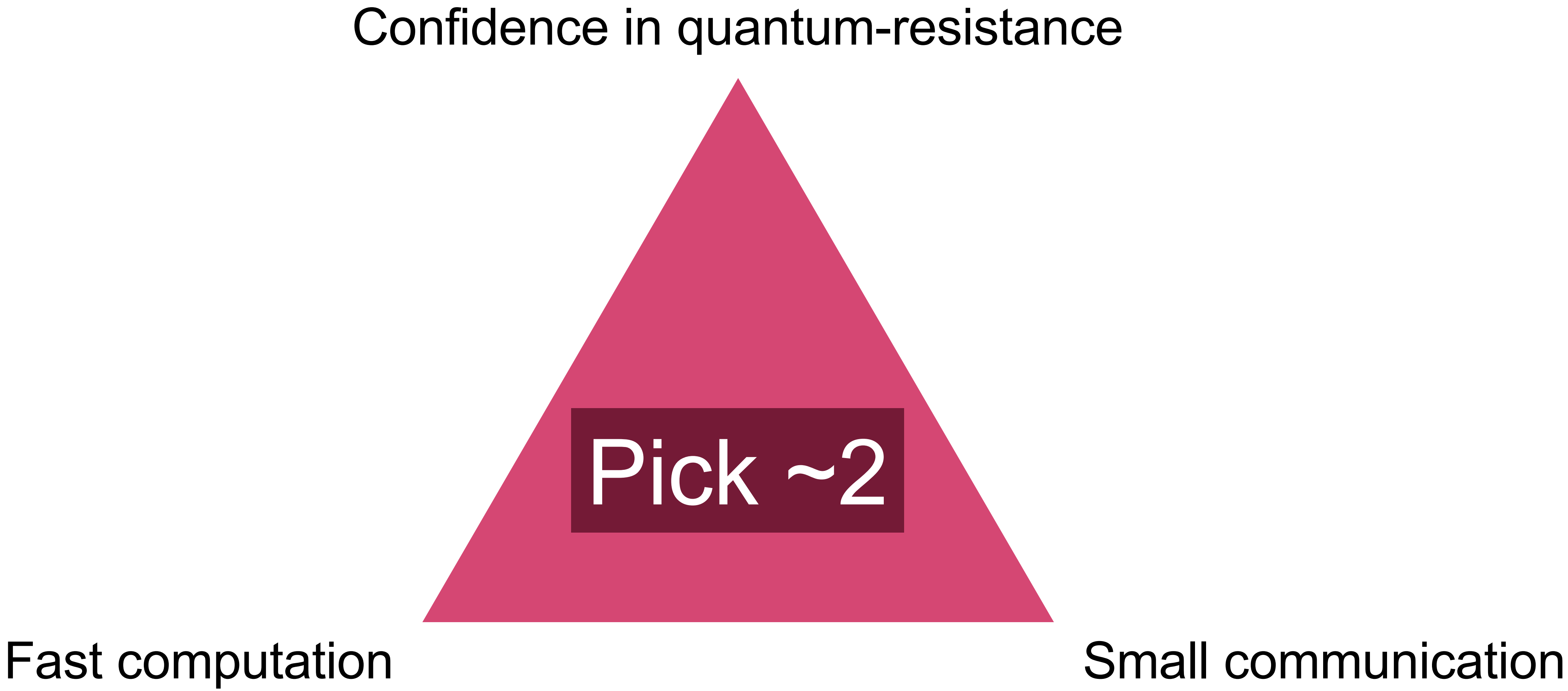
[Mosca] IEEE Security & Privacy 16(5):38–41, Sep/Oct 2018. https://doi.org/10.1109/MSP.2018.3761723
[Quantum threat] https://evolutionq.com/quantum-threat-timeline-2021.html

Timeline to replace cryptographic algorithms



Challenges

Trade-offs with post-quantum crypto

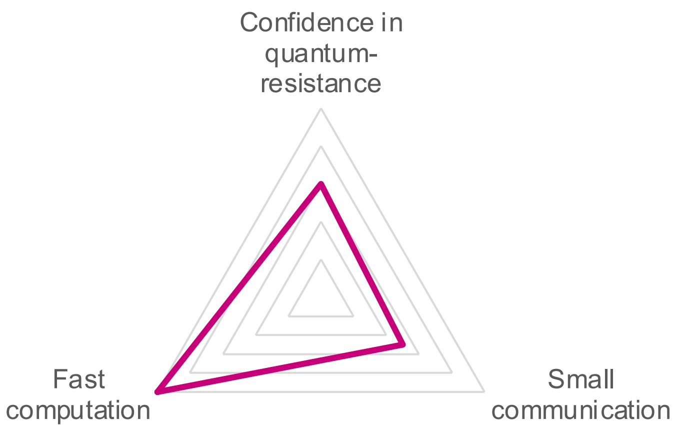


Trade-offs with post-quantum crypto

RSA and elliptic curves

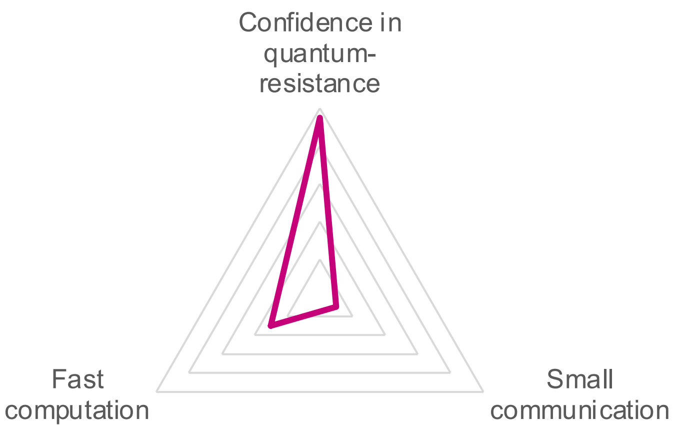
TLS handshake:
1.3 KB

Lattice-based cryptography



TLS handshake:
11.2 KB

Hash-based signatures



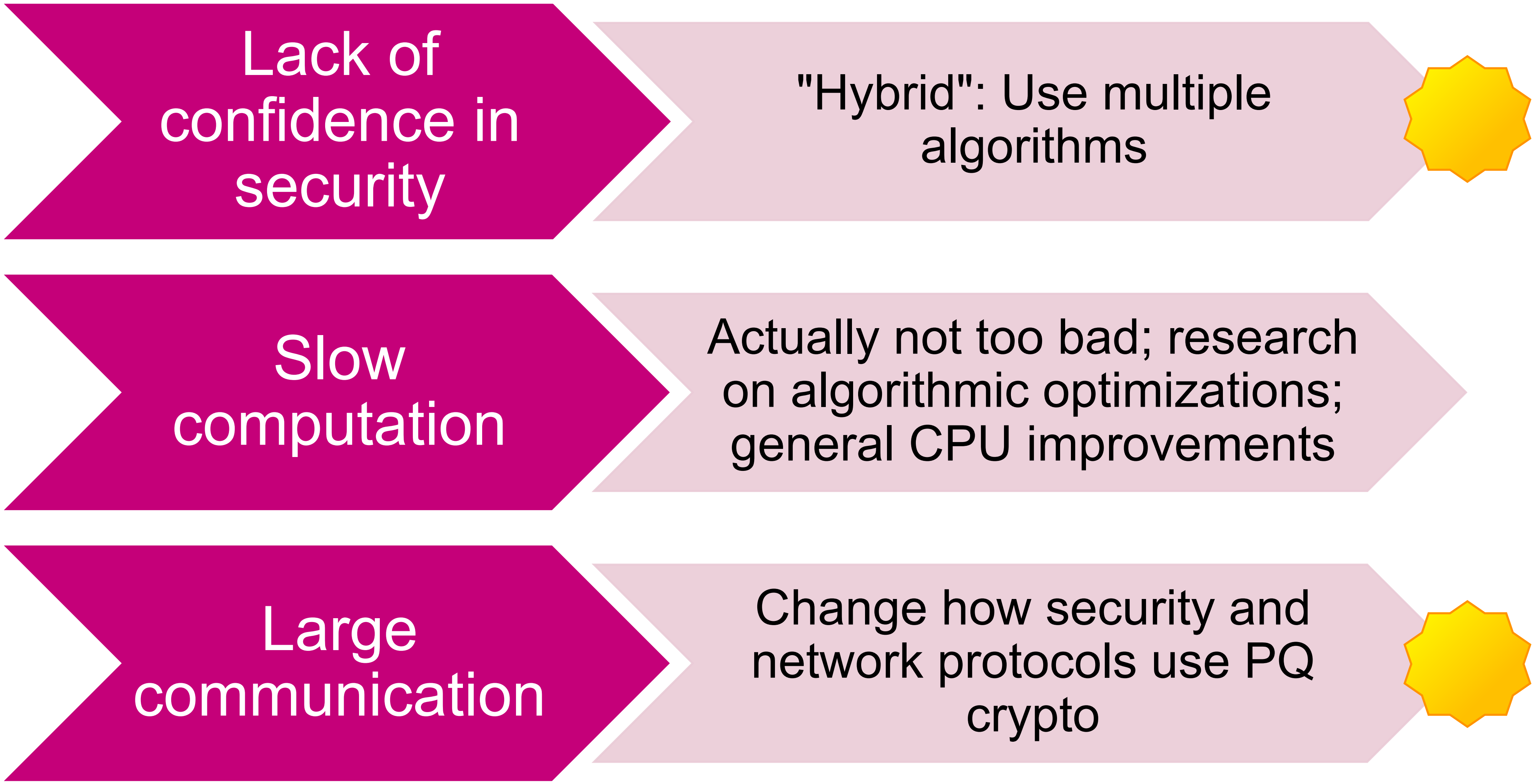
TLS handshake:
24.6 KB

Addressing the challenges of using PQ crypto

- Lack of confidence in security
- Slow computation
- Large communication

Make better PQ crypto

Addressing the challenges of using PQ crypto



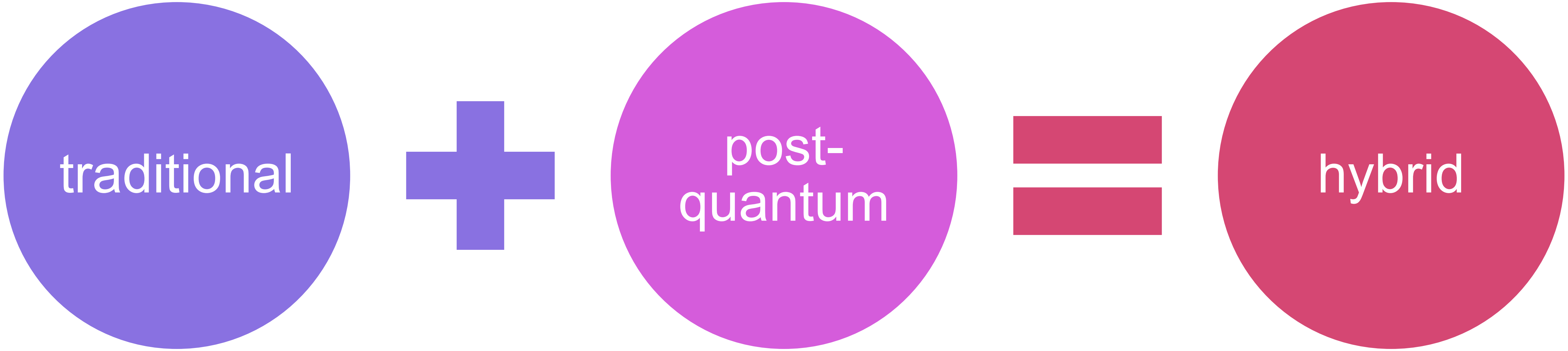
Increasing confidence in security

Hybrid: Classical + PQ

Douglas Stebila, Scott Fluhrer, Shay Gueron
https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/

Panos Kampanakis, Douglas Stebila, Torben Hansen
https://datatracker.ietf.org/doc/draft-kampanakis-curdle-ssh-pq-ke/

Hybrid approach: use traditional and post-quantum simultaneously such that successful attack needs to break both



Why use two (or more) algorithms?

1. Reduce risk from break of one algorithm

2. Ease transition with improved backwards compatibility

3. Standards compliance during transition

Why use two (or more) algorithms?

1. Reduce risk from break of one algorithm

- Enable early adopters to get post-quantum security without abandoning security of existing algorithms
- Retain security as long as at least one algorithm is not broken
- Uncertainty re: long-term security of existing cryptographic assumptions
- Uncertainty re: newer cryptographic assumptions

2. Ease transition with improved backwards compatibility

3. Standards compliance during transition

Why use two (or more) algorithms?

1. Reduce risk from break of one algorithm

2. Ease transition with improved backwards compatibility

- Design backwards-compatible data structures with old algorithms that can be recognized by systems that haven't been upgraded, but new implementations will use new algorithms
- May not be necessary for negotiated protocols like TLS

3. Standards compliance during transition

Why use two (or more) algorithms?

1. Reduce risk from break of one algorithm

2. Ease transition with improved backwards compatibility

3. Standards compliance during transition

- Early adopters may want to use post-quantum before standards-compliant (FIPS-)certified implementations are available
- Possible to combine (in a certified way) keying material from FIPS-certified (non-PQ) implementation with non-certified keying material

Hybrid key exchange

- Use two (or more) key exchange methods
- Transmit both public keys
- Combine shared secrets using hash function / key derivation function
  - Some questions on designing secure dual PRFs in the standard model
- Fairly well understood
- Seems likely to be broadly adopted in first phase of PQ transition

Hybrid authentication

- Use two (or more) authentication methods
- Transmit both public keys and signatures

- Significant debate of merits of and need for hybrid authentication
  - Seems unnecessary in the context of interactive / negotiated protocols
  - May be relevant for long-term scenarios like firmware updates and document signing
    - Counterargument: just use hash-based signatures

Post-quantum TLS

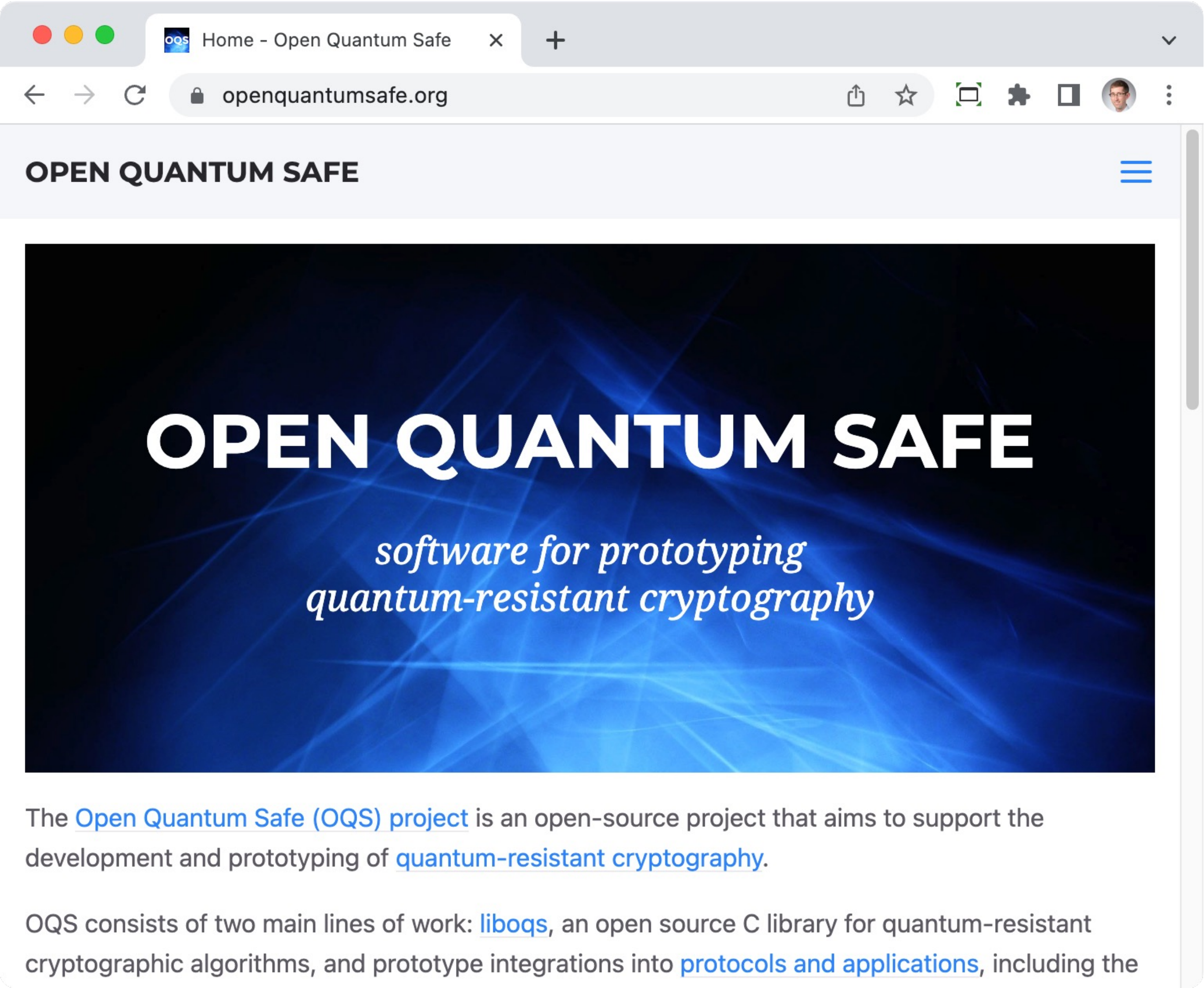
Three dimensions of "post-quantum TLS"

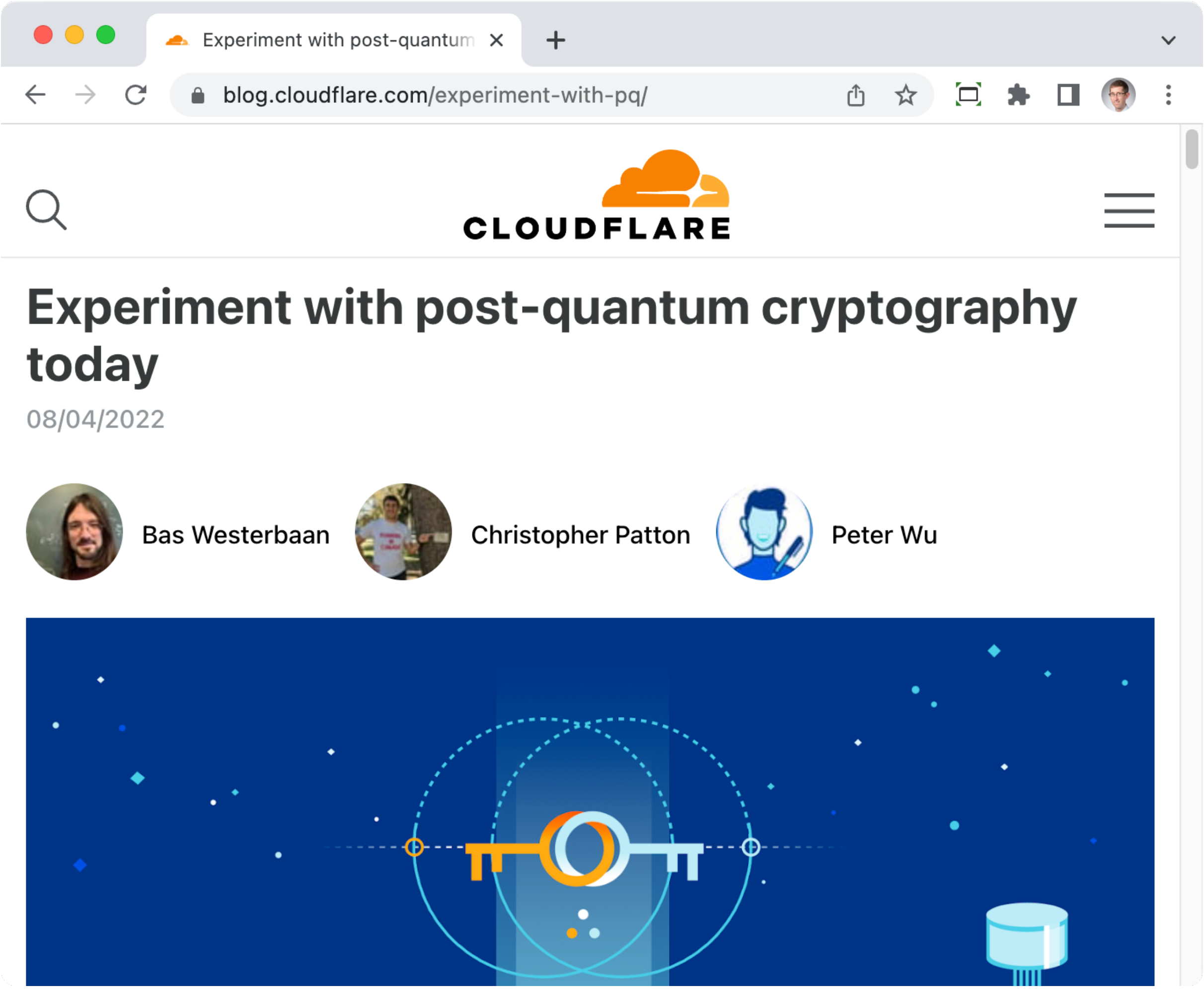
What is “post-quantum TLS”?

Pre-shared key (PSK) mode	Post-quantum key exchange	Classical+PQ key exchange	Post-quantum signatures	Classical+PQ signatures	Alternative protocol designs
• Already supported! • Still has the key distribution problem • No PQ forward secrecy	• Easiest to implement • Easy backwards compatibility • Needed soonest: harvest now & decrypt later with quantum computer	• “Hybrid” • Easy to implement • Possibly in demand during pre-FIPS-certification period	• On the web: requires coordination with certificate authorities • Less urgently needed: can’t retroactively break channel authentication	• “Hybrid” or “Composite” • May not make sense in the context of a negotiated protocol like TLS	• Harder to implement; may require state machine or architecture changes • Lots of interesting research to do!

Likely first to be adopted

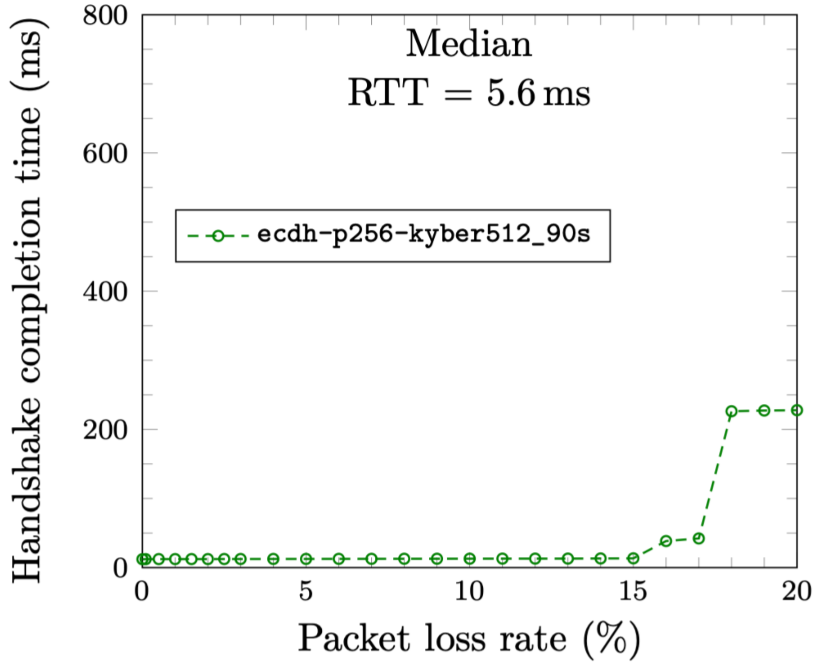
Preliminary PQ TLS experiments





TLS performance

Higher latency & packet loss



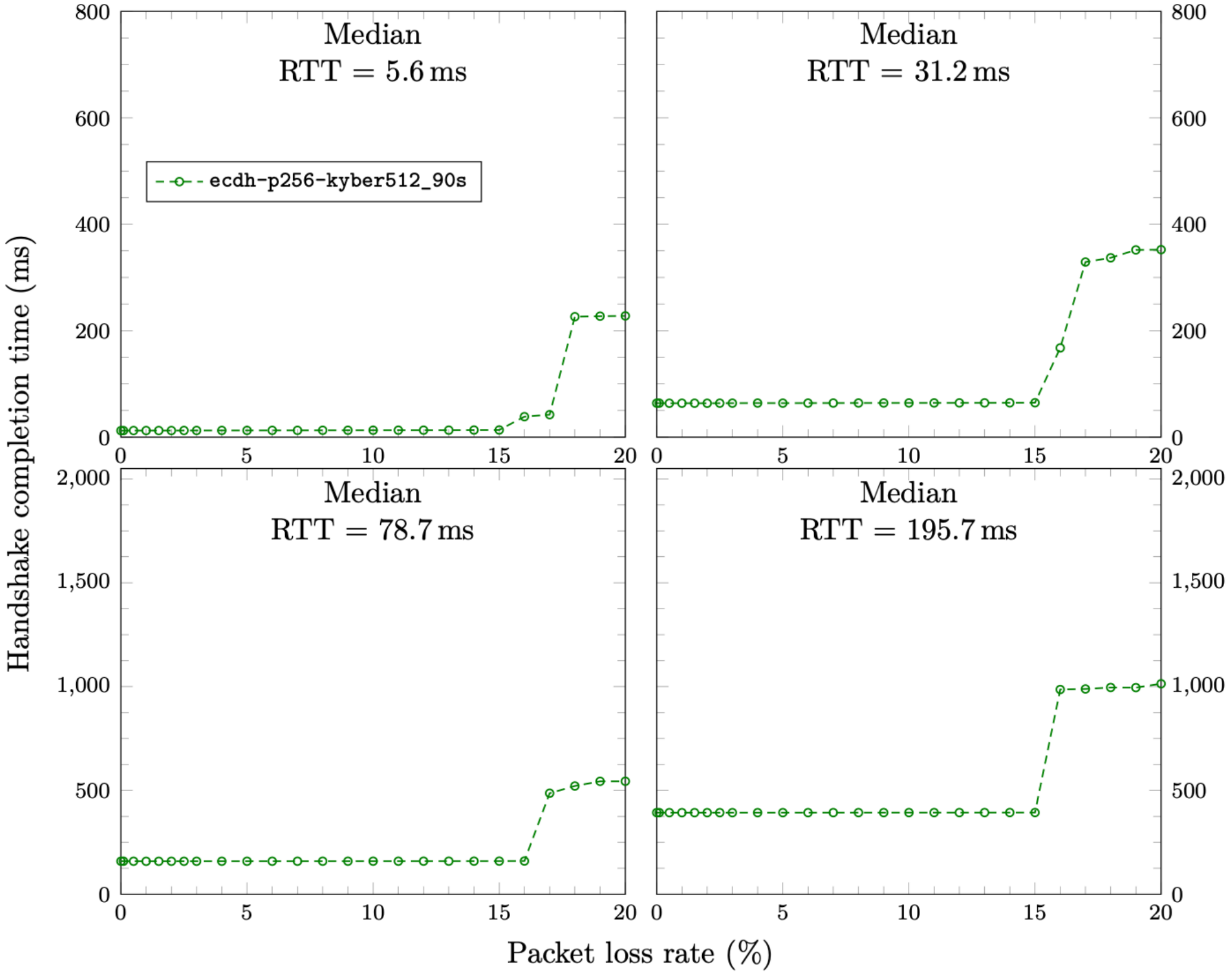
50th percentile

OQS-OpenSSL 1.1.1, x86_64, AVX2 enabled – https://eprint.iacr.org/2019/1447

TLS performance

Higher latency & packet loss

50th percentile

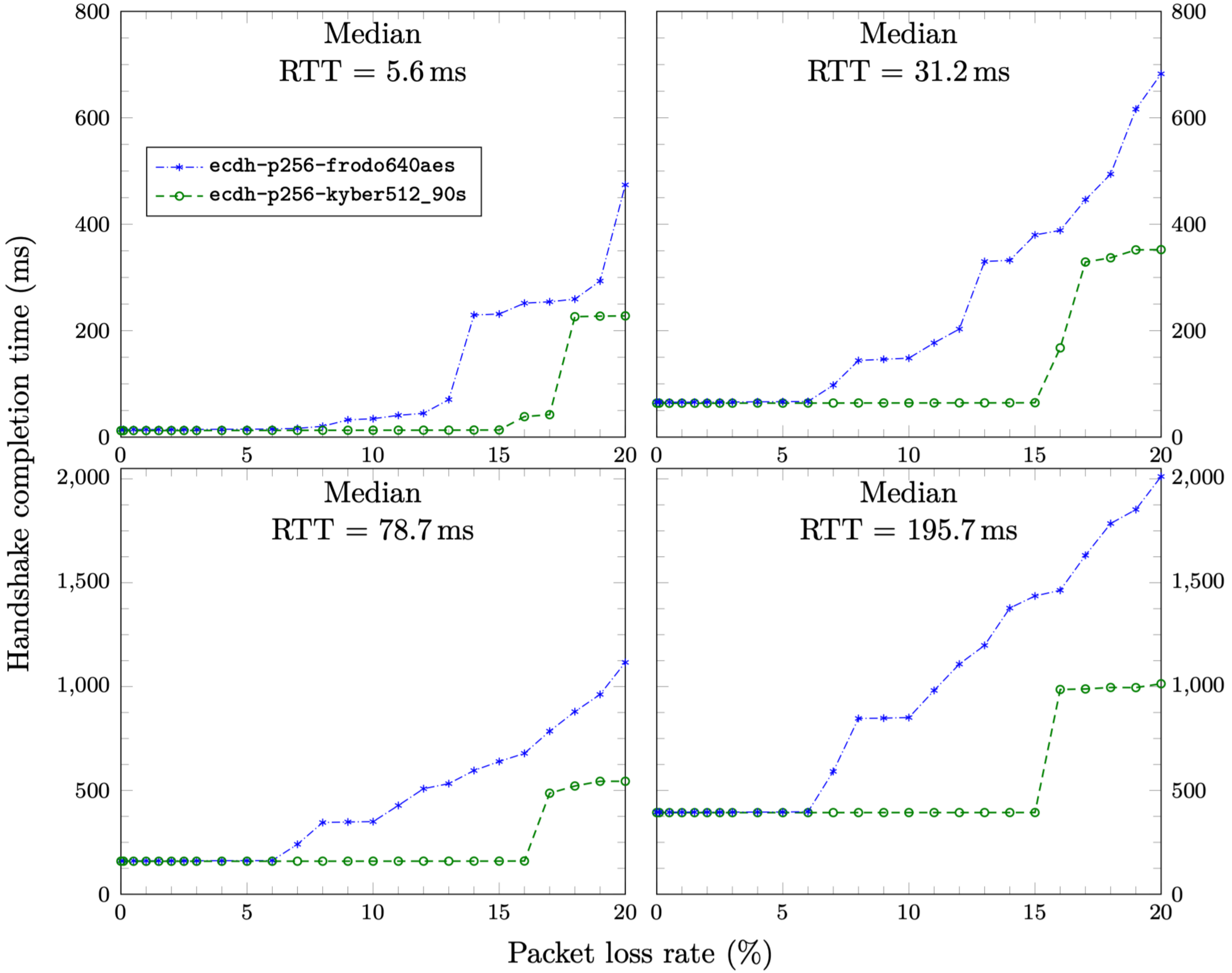


OQS-OpenSSL 1.1.1, x86_64, AVX2 enabled – https://eprint.iacr.org/2019/1447

TLS performance

Higher latency & packet loss

50th percentile

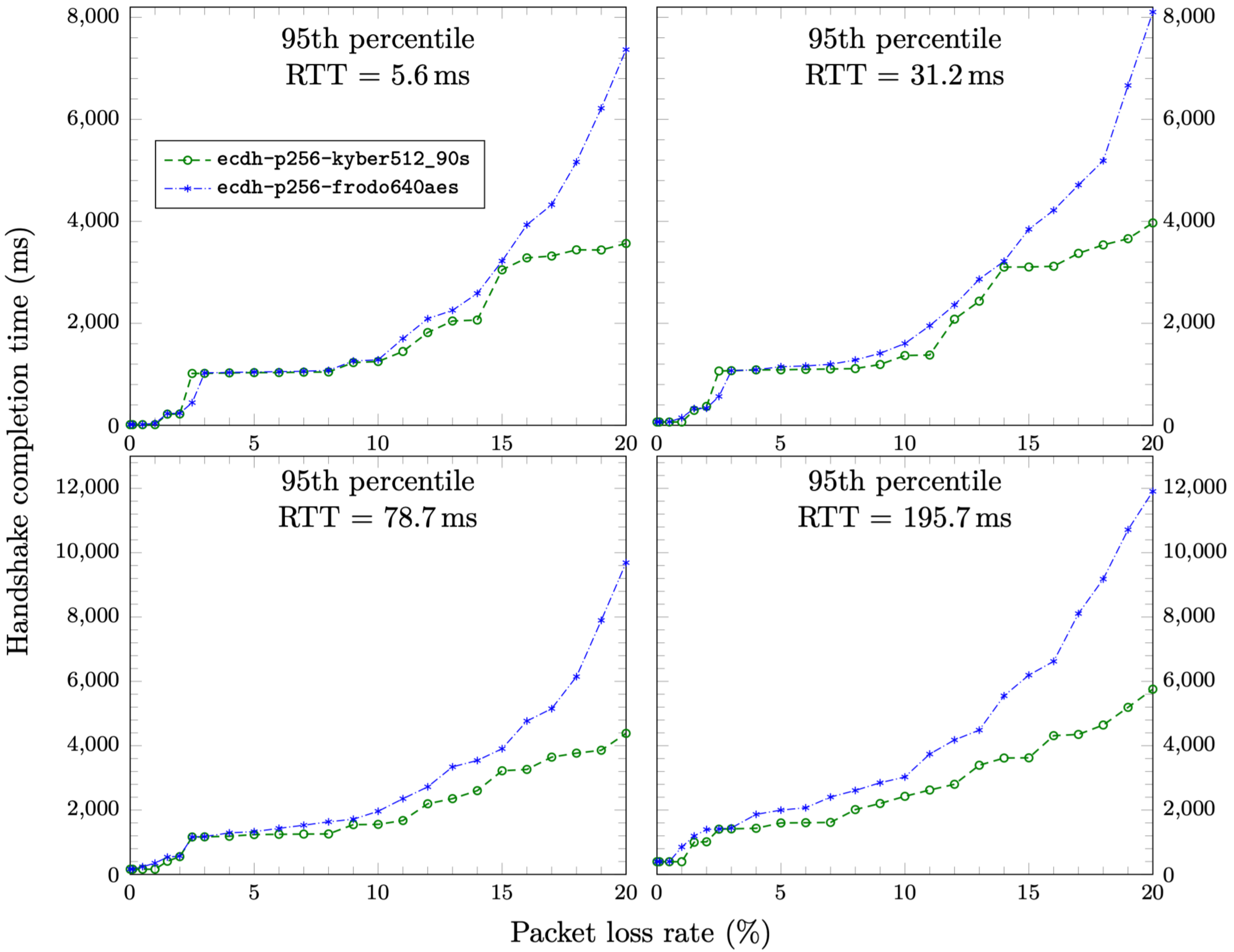


OQS-OpenSSL 1.1.1, x86_64, AVX2 enabled – https://eprint.iacr.org/2019/1447

TLS performance

Higher latency & packet loss

95th percentile



OQS-OpenSSL 1.1.1, x86_64, AVX2 enabled – https://eprint.iacr.org/2019/1447

TLS performance

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

Reducing communication size

Big communications is bad in constrained environments

- Consumes battery power
- Consumes costly mobile data
- May exceed available memory on small devices
- Long communication times on low bandwidth connections

Big communications is bad in unconstrained environments, too

Internet protocols running over **UDP** (unreliable datagrams)

- Need to fit into single packet ~1.4 KB

Internet protocols running over **TCP** (reliable connections)

- Greater chance of delays due to retransmission of lost packets
- Latency increases in early parts of communication due to small TCP window sizes

Reducing communication size

Strategy #1:

Change cryptographic protocols to use PQ algorithms more cleverly/efficiently

Strategy #2:

Change network protocols to be more communication efficient

- Technically about reducing latency due to communication size, not reducing communication size itself

Reducing communication size

Implicit authentication: KEMTLS

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

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

Sofía Celi, Jonathan Hoyland, Douglas Stebila, Thom Wiggers
ESORICS 2022. https://eprint.iacr.org/2022/1111

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

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

Observation:

PQ signatures

are bigger than

PQ public key encryption / KEMs

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 (stateful)**	**32**	**979**
SPHINCS+	**Hash-based (stateless)**	**32**	**7,856**

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 (stateful)**	**32**	**979**
SPHINCS+	**Hash-based (stateless)**	**32**	**7,856**

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**
BIKE	**Code-based**	**1,541**	**1,573**
Classic McEliece	**Code-based**	**261,120**	**128**
HQC	**Code-based**	**2,249**	**4,481**
CSIDH	**Isogeny-based**	**64**	**64**

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
$x \leftarrow_\$ \{0, \ldots, q-1\}$		$y \leftarrow_\$ \{0, \ldots, q-1\}$
$X \leftarrow g^x$	X $\longrightarrow$	$Y \leftarrow g^y$
	$\longleftarrow$ 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	

Static DH

Ephemeral DH

Key encapsulation mechanisms (KEMs)

An abstraction of Diffie–Hellman key exchange

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

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

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

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

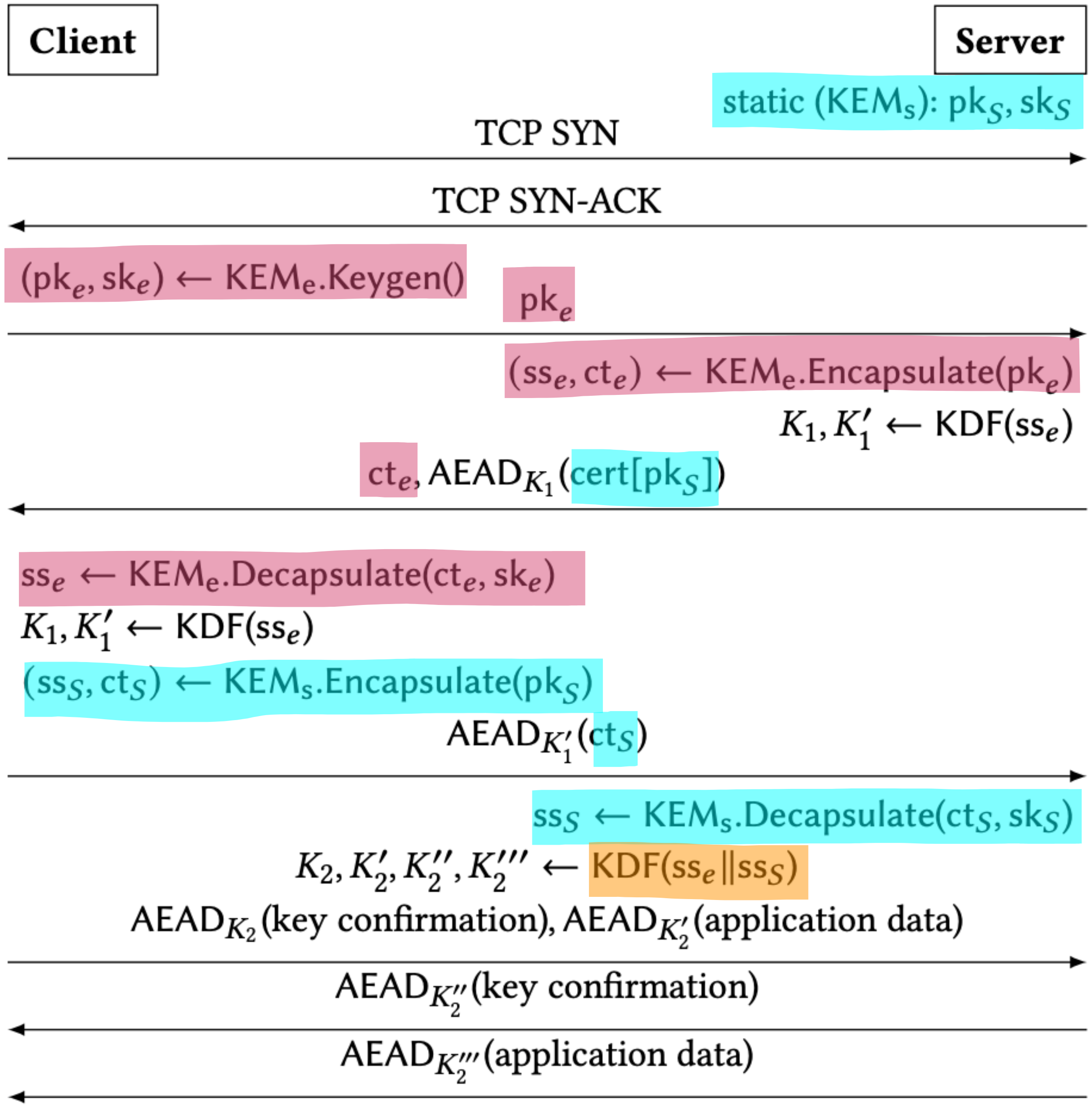
$k \leftarrow \mathsf{KEM.Decaps}(sk, ct)$

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

KEM for authenticated key exchange

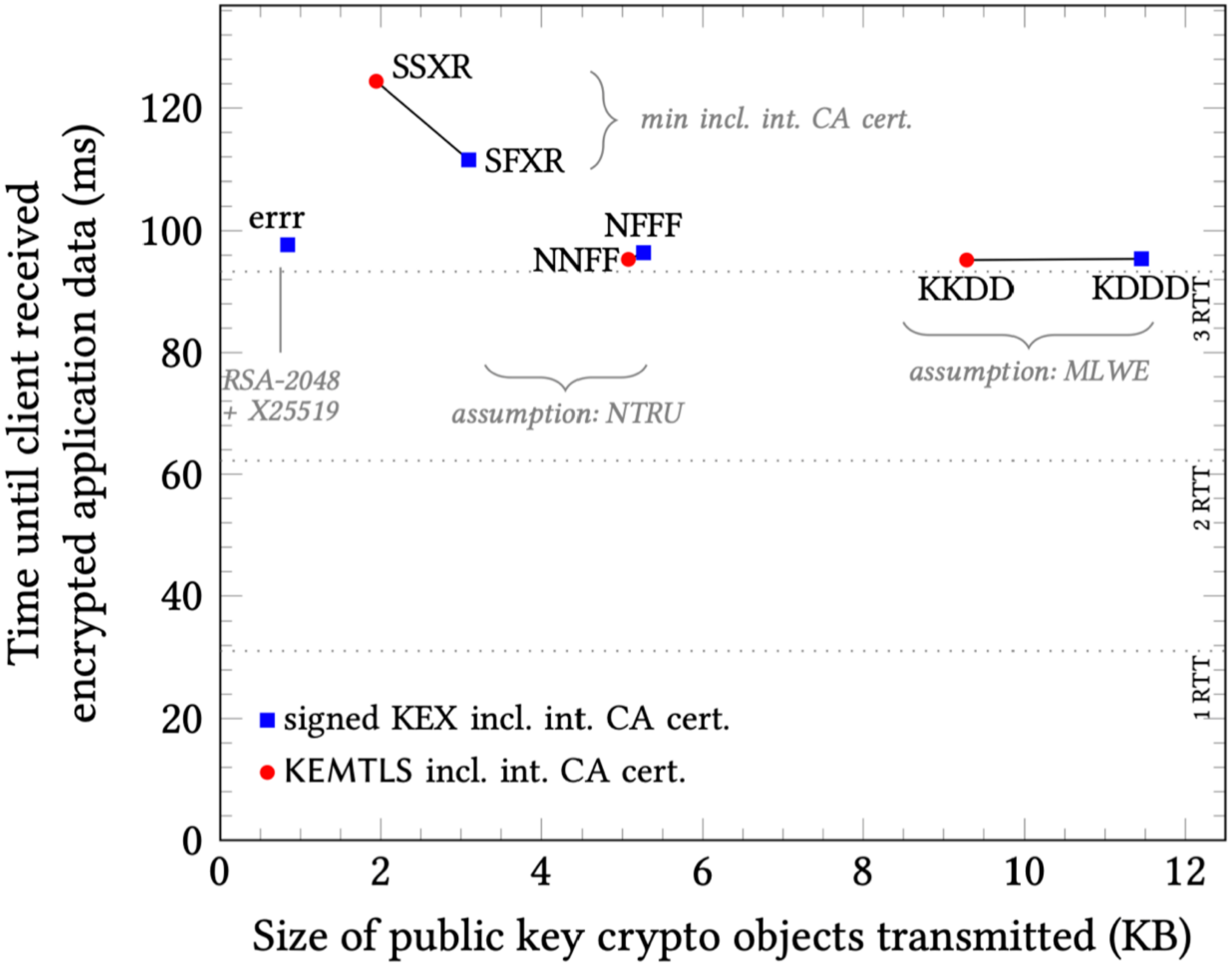
Signature scheme for intermediate CA

Signature scheme for root CA

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'

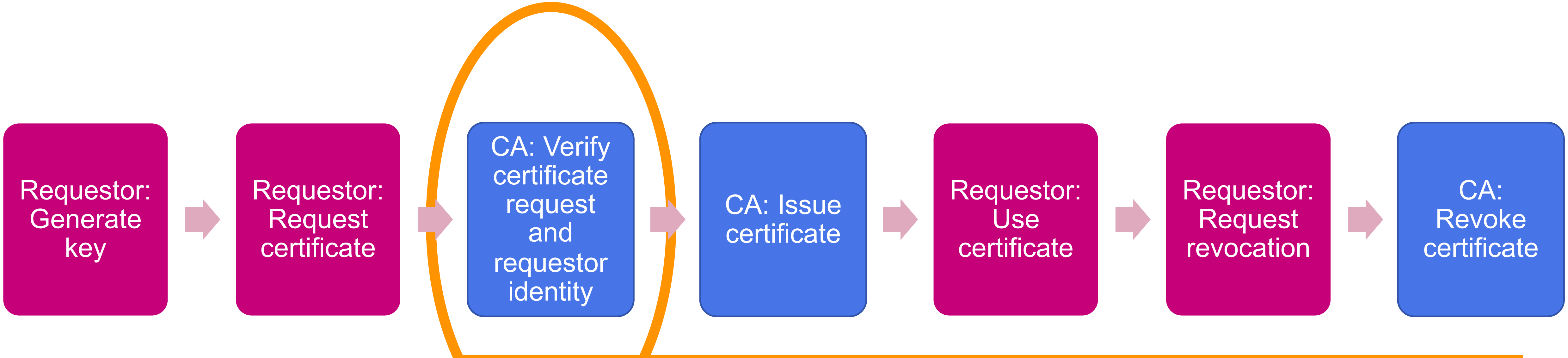


Rustls client/server with some AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

Certificate lifecycle for KEM public keys

Tim Güneysu, Philip Hodges, Georg Land, Mike Ounsworth, Douglas Stebila, Greg Zaverucha
ACM CCS 2022. https://eprint.iacr.org/2022/703

Certificate lifecycle



Proof of possession:
CAs want to verify that the requester has the corresponding secret key

Certificate requests in the X.509 PKI

How does requester prove possession of corresponding secret keys?

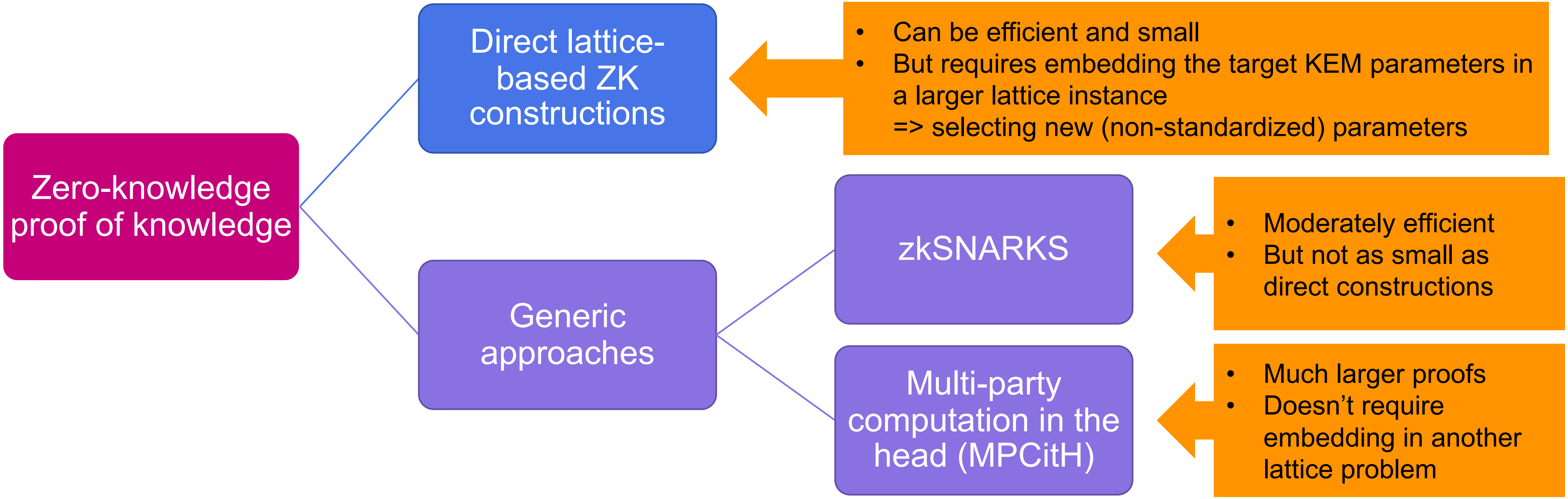
1. Interactive challenge-response protocol [RFC 4210 Sect. 5.2.8.3]
2. Send certificate back encrypted under subject public key [RFC 4210 Sect. 5.2.8.2]
   - Weird confidentiality requirement on certificate.
   - Maybe broken by Certificate Transparency or other logging mechanisms?
3. Non-interactive certificate signing requests [RFC 2986]
   - CSRs okay for signature schemes, but not for public key encryption or key encapsulation mechanisms

Goal:
Design non-interactive proof of possession for lattice-based KEM public keys

(so that we can have the same certificate lifecycle
for KEM certificates to enable KEMTLS)

lattice-based = FrodoKEM (plain LWE), Kyber (module LWE)

Possible approaches for non-interactive proof of possession for (lattice-based) KEM public keys



Our approach

Proof of possession via
verifiable generation

Generate the key and a
proof at the same time

FrodoKEM key generation

1. Generate $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ from a seed

2. Sample $\mathbf{S} \leftarrow_{\$} \chi^{n \times \overline{n}}$

3. Sample $\mathbf{E} \leftarrow_{\$} \chi^{n \times \overline{n}}$

4. Compute $\mathbf{B} \leftarrow_{\$} \mathbf{AS} + \mathbf{E}$

5. Public key: $(\mathsf{seed}_{\mathbf{A}}, \mathbf{B})$

6. Secret key: $\mathbf{S}$

<u>Frodo-640</u>

$q = 2^{15}$

$n = 640, \overline{n} = 8$

$\chi \in [-12, \ldots, 12]$

https://frodokem.org/

Verifiable generation for FrodoKEM

1. Generate and commit to many allegedly small values for S and E
2. Reveal some of them to prove they're small
3. Use the rest for the actual key generation

This doesn't prove that all the unrevealed values are small, only most of them with high probability

How do we prove we actually used them in the rest of the key generation?

- MPC-in-the-head à la Picnic
- Fiat–Shamir to get a signature scheme

5-round interactive protocol for verifiable generation

1. Prover: Generate sufficiently many small values.
   Generate an additive secret sharing among N parties.
   Commit to the shares.
   Send commitments.
2. Verifier: Pick some fraction of the bundles to audit.
3. Prover: Open commitments for challenged bundles.
   Use unaudited bundles to construct secret key (S, E) and public key B=AS+E. Commit to shares of B.
   Send commitments and public key (A, B).
4. Verifier: Select N-1 parties to audit.
5. Prover: Reveal state of N-1 parties.
6. Verifier: Check state of revealed parties.

Making it non-interactive

- Interactive protocol has soundness $^1/_N$, which isn't cryptographically small.
- Repeat τ times to get soundness $^1/_{N^\tau}$.
- (Use the same bundles from step 1 in all repetitions.)
- Apply the Fiat–Shamir transform to make it non-interactive:
  - Generate challenges in step 2 and 4 by hashing all previous commitments with a random oracle.

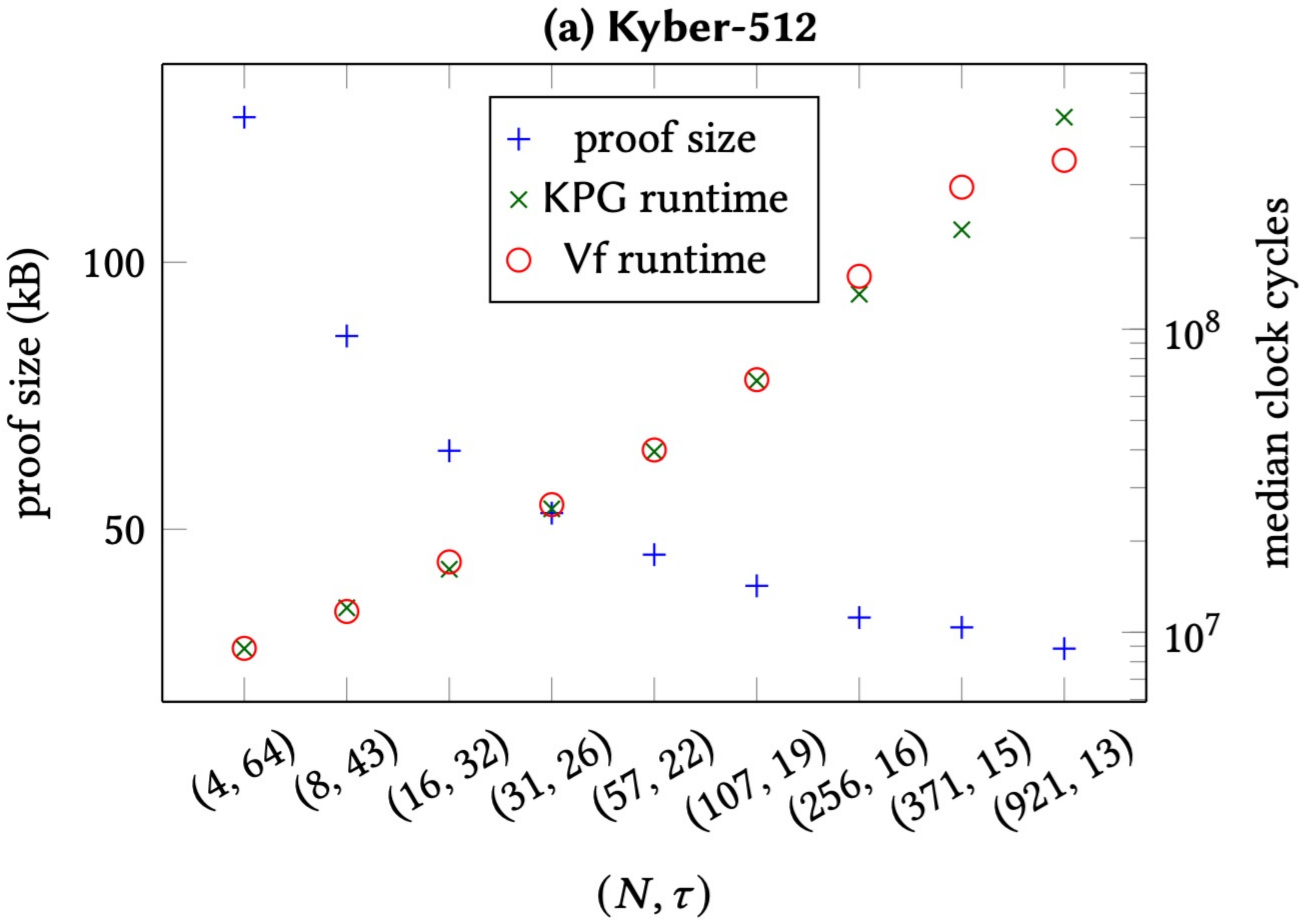
Performance trade-offs

52.9 kB / 0.01s

33.4 kB / 0.03s

25.6 kB / 0.1s

17.8 kB / 3.8s



Intel Core i7-8565U CPU @ 4.6 GHz, gcc 11.2.0

Summary of verifiable generation

Verifiable generation with MPC-in-the-head yields reasonable proof sizes and runtimes for both FrodoKEM and Kyber at all security levels

- Smallest sizes can be competitive with direct lattice-based ZK constructions without needing to embed in a larger LWE instance with different parameters
- Order of magnitude smaller than previous MPC-in-the-head approaches

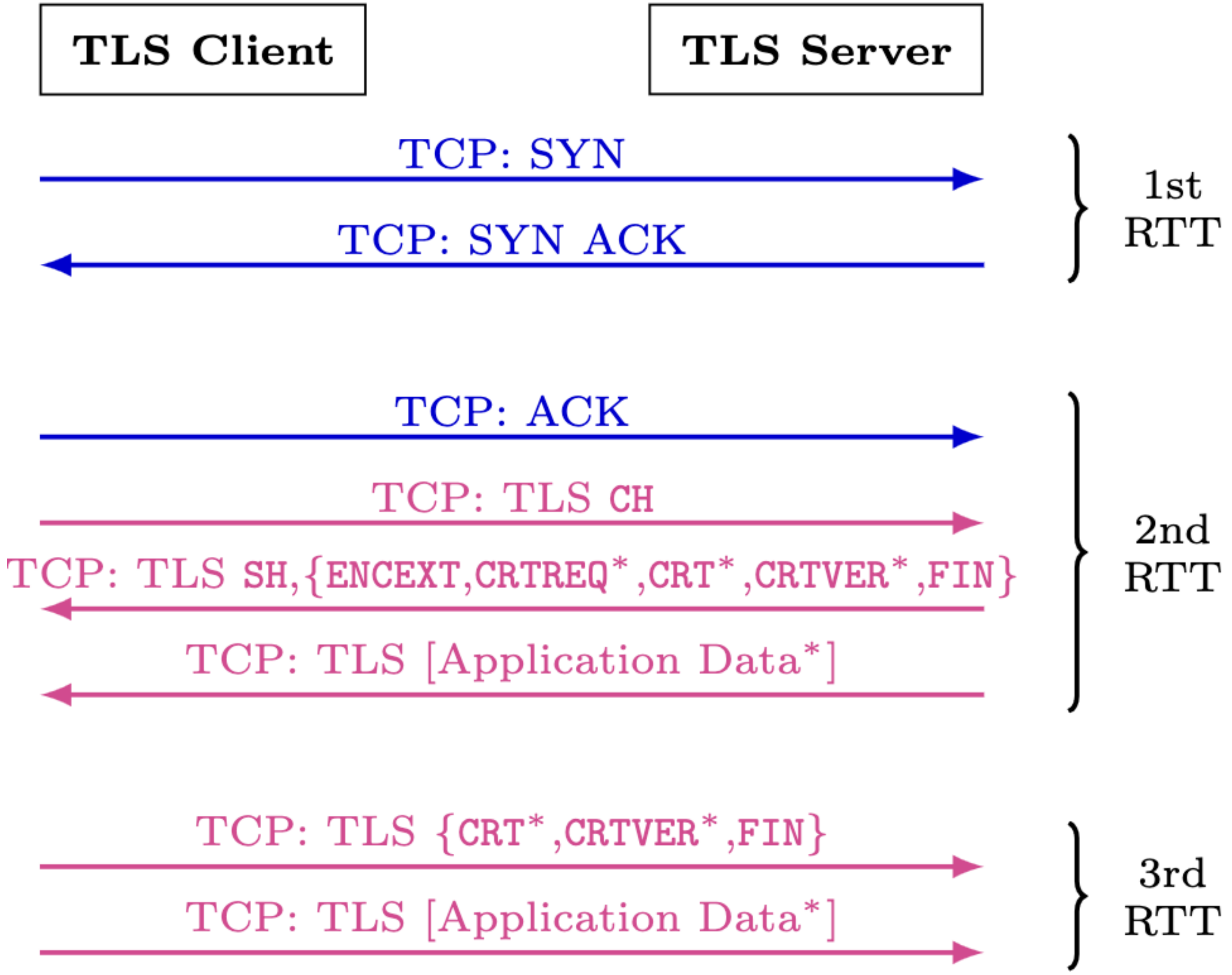
Reducing communication latency

UDP request-based fragmentation in DNSSEC and TLS 1.3

Jason Goertzen and Douglas Stebila
https://arxiv.org/abs/2211.14196

Carlos Aguilar-Melchor, Thomas Bailleux, Jason Goertzen, David Joseph, Douglas Stebila
https://arxiv.org/abs/2302.05311

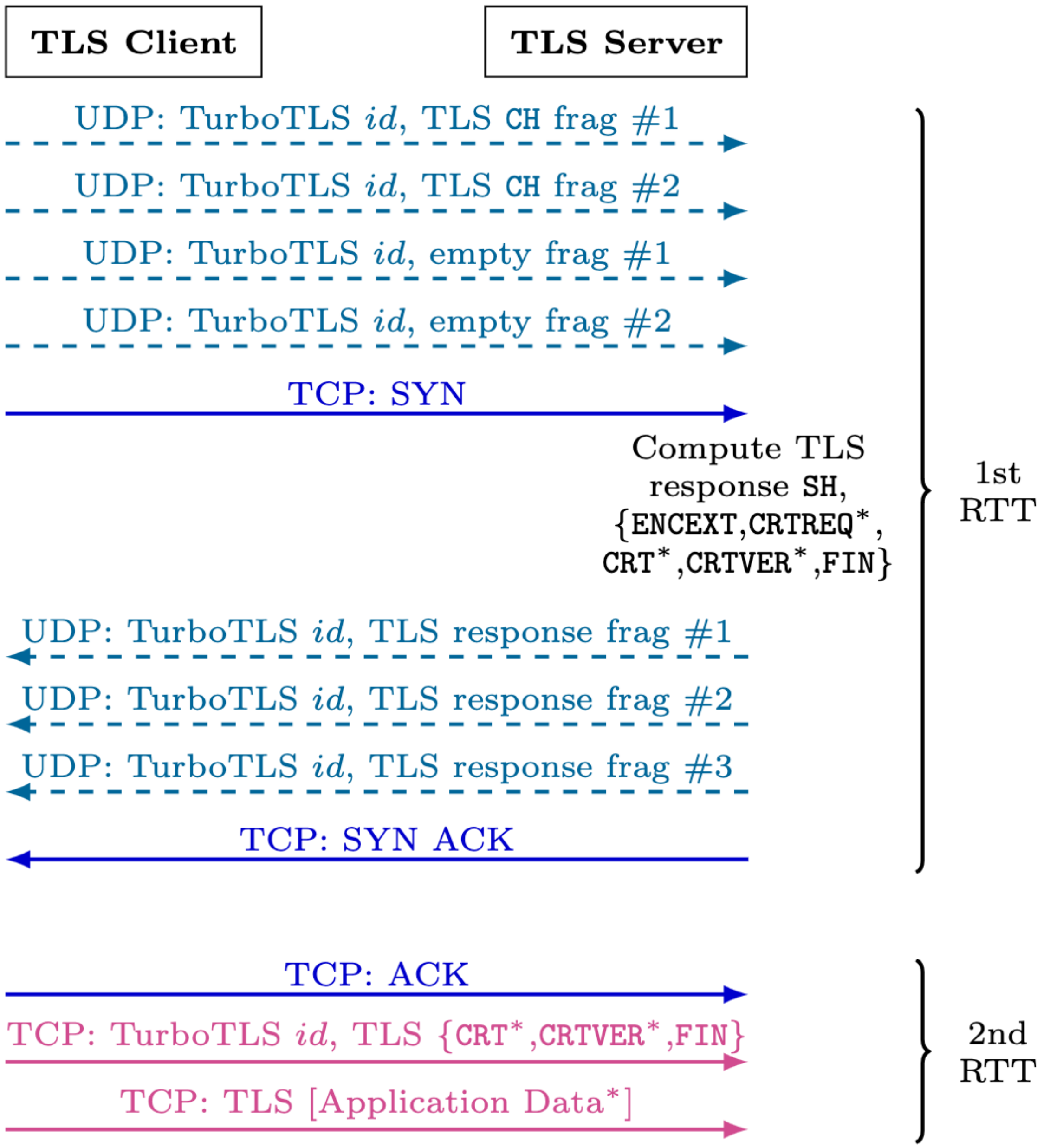
TLS 1.3 connection establishment



2 round trips before client starts sending application data

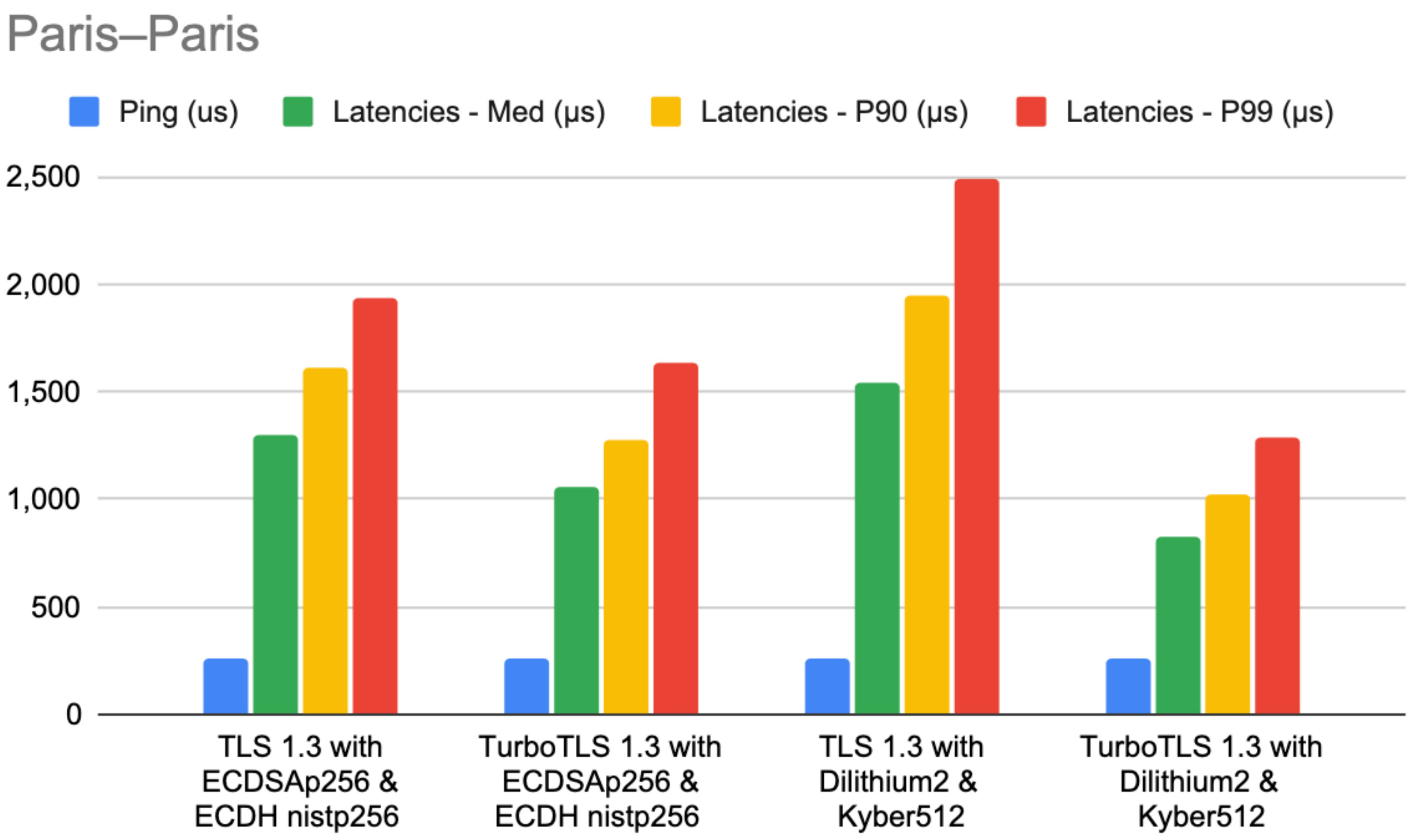
TurboTLS: connection establishment with 1 less round trip

Idea: do first TLS handshake flow over UDP while doing TCP handshake in parallel

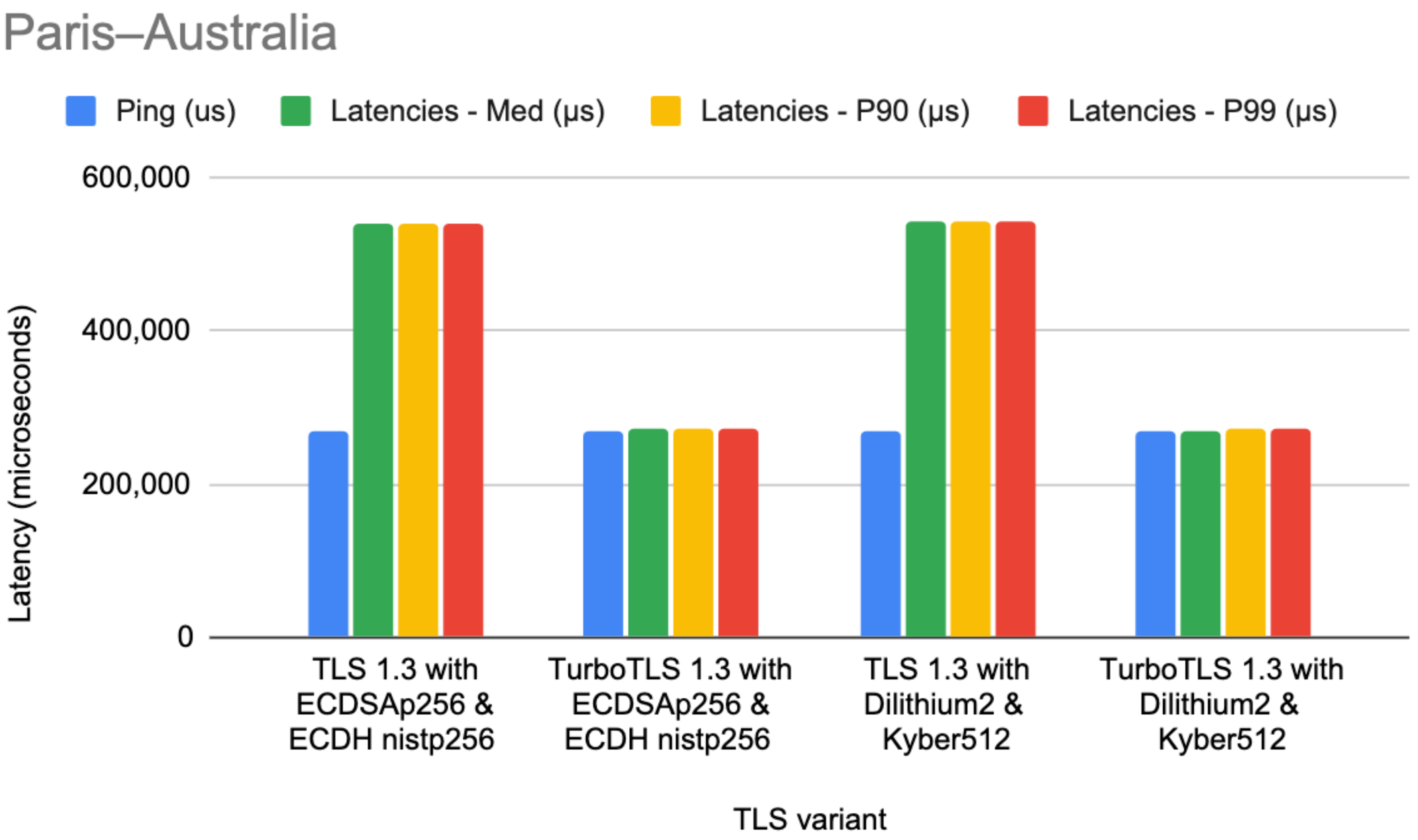


TurboTLS performance

On short distance connections, starts to make a difference…



TurboTLS performance



On long distance connections, halves latency of connection establishment

Wrapping up

Protocol	Key exchange / PKE	Authentication	Alternatives
TLS 1.3 (secure channel)	Hybrid: • Draft available (this talk) • Academic and industry experiments, early deployment PQ only: no activity	Hybrid: • Debate over merits PQ only: • Academic experiments	KEMTLS design for implicit authentication (this talk) TurboTLS for lower latency (this talk)
Secure Shell (SSH) (secure channel)	Hybrid: • Draft available • Already deployed in OpenSSH by default PQ only: no activity	Hybrid: • Debate over merits PQ only: • Already deployed in OpenSSH	
IPsec (secure channel)	Hybrid: • Draft available	No activity	
Certificates (X.509) (public key infrastructure)	Hybrid: no activity PQ only: • Drafts for Kyber	Hybrid: • Debate over merits PQ only: • Drafts for Dilithium	
Secure E-Mail (S/MIME and CMS) (encryption and/or authentication)	Hybrid: • Draft available PQ only: • Drafts for Kyber	Hybrid: • Debate over merits PQ only: • Drafts for Dilithium, SPHINCS+	
Domain Name Security (DNSSEC) (authentication)	Not applicable	Hybrid: no activity PQ only: • Academic research on Falcon, aggregated hash trees	Request-based fragmentation for handling large DNSSEC packets

Rethinking Internet protocols for post-quantum cryptography

Douglas Stebila
University of Waterloo

Public key cryptography designed to resist attacks by quantum computers

- Five families of mathematical assumptions
- Standardization of core algorithms under way by US National Institute of Standards and Technology
- Starting the process of standardizing post-quantum cryptography in Internet protocols

Addressing challenges in using post-quantum cryptography

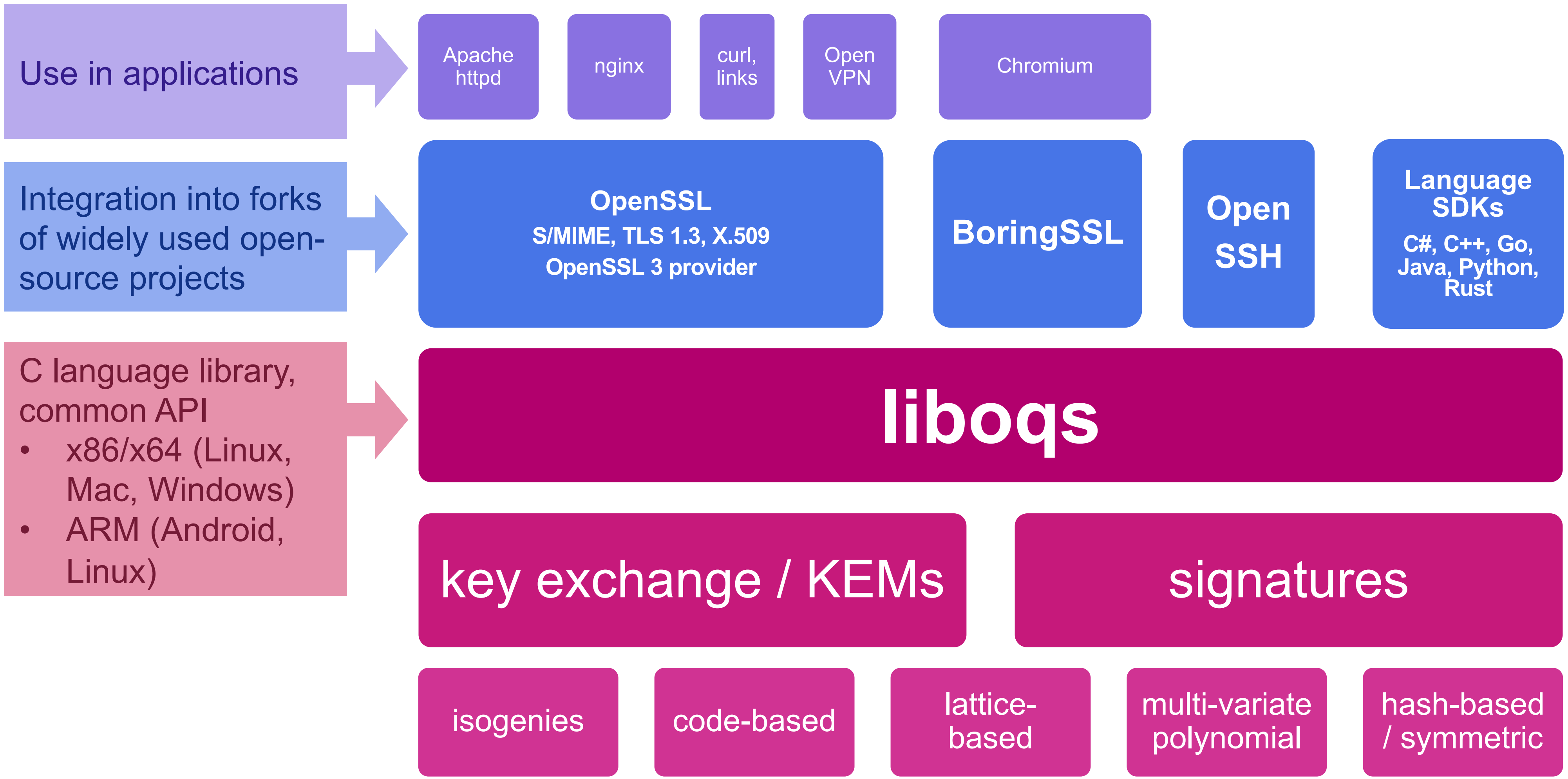
Lack of confidence in security	"Hybrid": Use multiple algorithms	this talk
Slow computation	Actually not too bad; research on algorithmic optimizations; general CPU improvements	
Large communication sizes	Change how security and network protocols use PQ crypto	this talk

https://www.douglas.stebila.ca/research • https://openquantumsafe.org/

Appendix

<table>
<tr><th colspan="2">Post-quantum</th><th>Traditional public key crypto</th></tr>
<tr><td colspan="2">Computational assumptions studied since</td><td rowspan="2">Computational assumptions studied since 1970s / 1980s</td></tr>
<tr><td>1970s</td><td>1990s/2000s/2010s</td></tr>
<tr><td colspan="2">Conjecturally resistant to quantum attacks</td><td>Vulnerable to quantum attacks</td></tr>
<tr><td colspan="2">Medium to large communication sizes
(700–30000+ bytes)</td><td>Small communication sizes
(32–384 bytes)</td></tr>
<tr><td colspan="2">Sub-millisecond computation times</td><td>Sub-millisecond computation times</td></tr>
<tr><td colspan="2">Less flexible for building fancy cryptography</td><td>Flexible for building fancy crypto</td></tr>
</table>

Open Quantum Safe Project



Led by University of Waterloo

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

Additional contributors:
- Senetas
- PQClean project
- Individuals

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

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

Four TLS 1.3 modes

- Signed Diffie–Hellman, server-only authentication
- Signed Diffie–Hellman, mutual authentication
- Pre-shared key (PSK) ← **Already PQ!**
- Pre-shared key with ephemeral Diffie–Hellman (PSK-ECDHE)

Defining security for proof of possession

Unforgeability:

- Hard to construct a valid proof of possession for an honest public key without the corresponding secret key

Zero knowledge:

- The proofs of possession leak no information about the secret key.

- Need to ensure the proof of possession composes nicely with the intended usage of the key
  - Zero knowledge shows the proof doesn't undermine the scheme
  - Need to extend unforgeability:
    - Use an "auxiliary secret key usage algorithm" in unforgeability experiment
    - Introduce a notion of KEM simulatability which FO-based KEMs have

Uniqueness of small FrodoKEM solutions

Recall high-level idea:

1. Generate and commit to many allegedly small values for S and E
2. Reveal some of them to prove they're small

This doesn't prove that all the unrevealed values are small, only most of them with high probability

- We prove a lemma upper-bounding the probability that a second FrodoKEM solution exists with mostly small solutions
- Choose number of bundles to audit to ensure no other mostly small secret key exists
- So proving possession of a mostly small solution implies proving possession of the true secret key
- Similar result for Kyber

Comparison with other approaches

Scheme	Technique	Regime 1	Regime 2		Kyber512		Frodo640	
		Size	Size	Time	Size	Time	Size	Time
Proof of knowledge of secret key (and proof of verifiable decryption, denoted ◇)								
Stern-like [49]	ZKP from SIS	2.3 MB†	4.3 MB†					
[7]	MPCitH		4.1 MB	2.4 s			$\geq$ 8.42 MB‡	
[15]	ZKP from RLWE & RSIS	384 kB†						
[11]	Σ-prot. for permuted-kernel	233 kB	444 kB					
Ligero [4]	zkSNARK from PCPs	157 kB†	200 kB†					
Aurora [9]	zkSNARK for R1CS	72 kB†	71 kB†					
[34]	ZKP from MLWE & MSIS	47 kB, 61 kB◇						
[54]	ZKP from MLWE & MSIS	47 kB*						
[55]	ZKP from MSIS & ext. MLWE	33 kB			43.6 kB◇			
[53]	ZKP from MLWE & MSIS	14 kB			19.0 kB◇			
Proof of verifiable generation								
Ours (31, 26)	MPCitH	251 kB	879 kB		52.9 kB	0.006 s	650 kB	0.12 s
Ours (256, 16)	MPCitH	155 kB	542 kB		33.4 kB	0.028 s	402 kB	0.63 s
Ours (1626, 12)	MPCitH	117 kB	407 kB		25.7 kB	0.109 s	302 kB	2.59 s
Ours (65536, 8)	MPCitH	79 kB	272 kB		17.8 kB	3.77 s	203 kB	85.6 s

TurboTLS comparison

	Runs over	UDP 1 req. ⇒ 1 resp.	Provides conn.	Kernel netw.	No state	TLS-based	Widely deployed	RTT to 1st byte
TLS 1.2	TCP	—	●	●	●	●	●	3
TLS 1.2 FalseStart	TCP	—	●	●	●	●	●	2
TLS 1.3	TCP	—	●	●	●	●	●	2
TLS 1.3 PSK	TCP	—	●	●	○	●	●	1
TLS 1.3 ECH	TCP	—	●	●	○	●	◓	1
OPTLS	TCP	—	●	●	○	●	○	1
TLS 1.3 + TCP Fast Open	TCP	—	●	●	○	●	◔	1
DTLS 1.3	UDP	●	○	●	●	●	●	2
QUIC	UDP	○	●	○	●	◕	◓	1
MinimaLT	UDP	●	●	○	●	○	○	1
MinimaLT with state	UDP	●	●	○	○	○	○	0
TurboTLS	UDP+TCP	●	●	◕	●	●	—	1
TurboTLS + PSK	UDP+TCP	●	●	◕	○	●	—	0
TurboTLS + ECH	UDP+TCP	●	●	◕	○	●	—	0