Post Quantum Cryptography

Sushmita Ruj

Recap

- Crypto for cloud
- Bilinear Pairings
- BLS signatures
- Shamir Secret sharing
- Data Audits
- Attribute based access control

This Lecture

- Why PQC
- What is PQC
- PQC Techniques
- Hash-based signatures
- PQC Transition

Quantum Computing

- Practical Quantum Computing existence/timeline is still debatable1
- QC research funding is increasing
- BM has multiple small-scale prototypes
- Google’s quantum supremacy claim

1Dyakonov, Mikhail. "When will useful quantum computers be constructed? Not in the foreseeable future, this physicist argues. Here's why: The case against: Quantum computing." IEEE Spectrum56.3 (2019): 24-29

Timeline

1982 Idea by Richard Feynman proposed

1998 First 2-qubit quantum computer realized

2000 7-qubit quantum computer realized

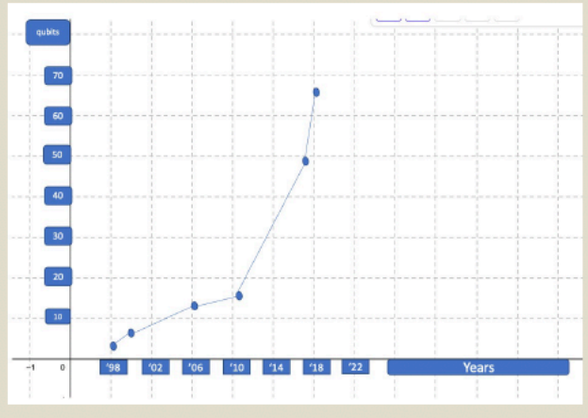
2006 12-qubit quantum computer realized

2011 14-qubit quantum computer realized

2017 49/50-qubit quantum computer realized

2018 72-qubit quantum computer realized

2020 IBM realized a 65-qubit quantum computer



IBM future goal: >1000 qubits by 2023

- A 1000-qubit computer is an important milestone
- However still 1000 times too small to fulfill quantum computing's full potential (e.g. breaking public-key crypto)

2021 127 qubits realized

2022 433 qubits

2023 1127 qubits

Classical Cryptography

Classical public-key cryptography relies on the following hard mathematical problems:

Integer Factorization:

Given an integer n, factor into prime numbers p, q, i.e. $n = p \cdot q$.
Classical algorithm with best asymptotic approximation is General Number Field Sieve with sub-exponential time

$$O(exp(c \log n)^{1/3}(n \log n)^{2/3}))$$

Discrete Logarithm Problem:

Given$b = a^e \mod n$, compute exponent e.

- Some of the best-known algorithms to solve this problem are: Brute Force, Pollard's rho, Number Field Sieve.
- Their runtime is exponential or sub exponential, therefore inefficient!

These problems can be easily solved by quantum computers

Quantum Threat

- Shor’s algorithm (1994): Factors Large numbers faster than classical algorithms

- Threat to public key cryptography

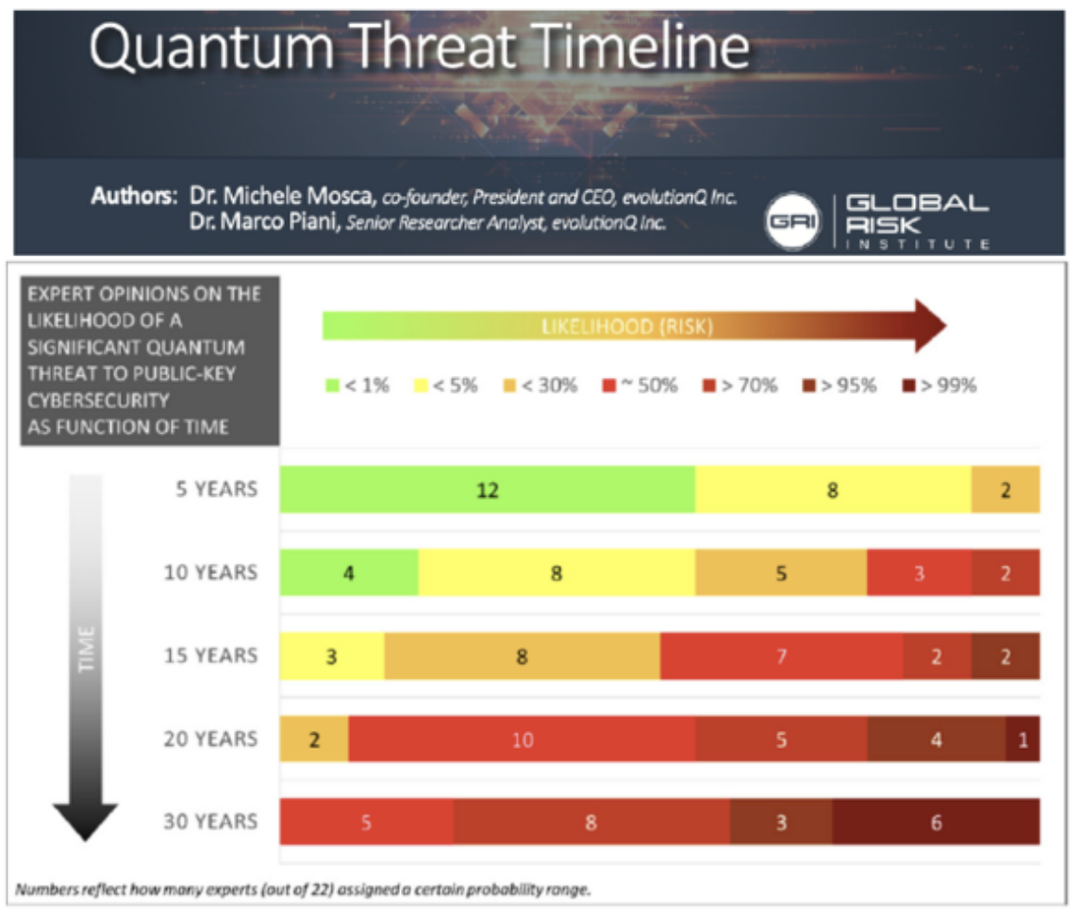
- Oded Regev (2023): Faster factoring algorithm (improvement over Shor’s algorithm)

- Grover’s algorithm(1996): Speeds up structured search quadratically

-Demands larger key sizes for symmetric key algorithms

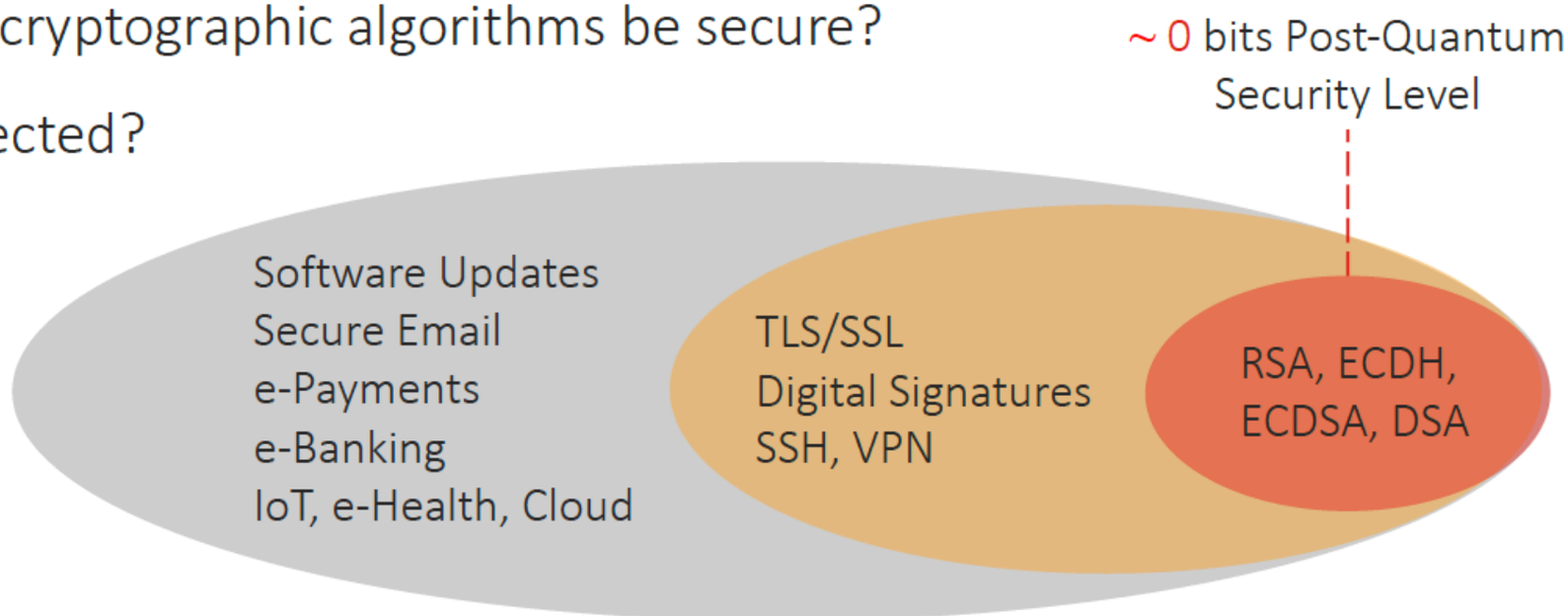
Need 100000 Qubits (approx) to break 2048-bit RSA

Quantum Safe Technologies



Quantum Computing – Practical impact?

- A large scale QC will be able to solve Integer Factorization and Discrete Logarithm Problems[1]
- Will our current cryptographic algorithms be secure?
- What will be affected?



[1]Shor, Peter W. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." *SIAM review* 41.2 (1999): 303-332

Techniques

- Lattice-based
- Code-based
- Symmetric key based/Hash based
- Multivariate
- Isogenies

NIST Standardization Efforts

- Encryption: Crystals Kyber

- Signatures: Crystals Dilithium, Falcon, Sphincs+

Motivation

- Authenticated key exchange
- Two parties establish a shared secret over a public communication channel
- Many security definitions capturing various adversarial powers:
- Different types of authentication credentials: public key, shared secret key, password, identity-based, …
- Additional security goals: weak/strong forward secrecy (Forward secrecy protects past sessions against future compromises of keys or passwords.), key compromise, impersonation resistance, post-compromise security, …
- Group key exchange
- Real-world protocols: TLS, SSH, Signal, IKE, ISO, EMV, …

Hash-Based Signatures

Lamport Signatures

Key generation:
Secret key: (pseudo-)random
$((x_{0,0}, x_{0,1}), (x_{1,0}, x_{1,1}), (x_{2,0}, x_{2,1}), \cdots, (x_{256,0}, x_{256,1})$
each $s_{i,j} \in \{0,2^{256-1}\}$

Public key:
$(H(x_{0,0}), H(x_{0,1})), (H(x_{1,0}), H(x_{1,1})), (H(x_{2,0}), H(x_{2,1})), \cdots, (H(x_{256,0}), H(x_{256,1}))$

- Signing:

Sign messages (hashes) of 256 bits $(m_0, m_1 . \cdots, m_{255})$

Signature is $(x_{0,m_0}, x_{1,m_1}, x_{2,m_2}, \cdots, x_{255,m_{255}})$

- Verification:

Compare hashes of signature components to elements of the public key

Lamport Signatures

Secret key $x_{0,0}$ $x_{0,1}$ $x_{1,0}$ $x_{1,1}$ $x_{256,0}$ $x_{256,1}$

Public key $H(x_{0,0})$ $H(x_{0,1})$ $H(x_{1,0})$ $H(x_{1,1})$ $H(x_{256,0})$ $H(x_{256,1})$

Message m_0 $m_{1,0}$ $m_{256,1}$

Signature $x_{0,0}$ $x_{1,0}$ $x_{256,1}$

Verify $\forall i, H(\sigma_i) = pk_i$

This is a one-time signature

Lamport Signatures

- One-time signatures
- Large key sizes
- Shrinking secret keys: Generate secret keys using PRG. Size = size of seed of PRG, increases cost of signing
- Shrinking public keys: Homework. (Exervcise 14.9, Boneh-Shoup's book)
- PQ Security: Using exhaustive attacks a quantum adversary can win the game in time $2^{v/2}$, v is the size of the hash function.

Lamport-like signatures

- One-time to many time signatures

Key Generation : A secret key is n random values $x_1, x_2, \cdots, x_n \in \mathcal{X}$ for some n. The public-key consists of the n hashes $y_1 = f(X_i)$ for $i = 1,2,\cdots,n$.

Signing: To sign a message $m \in \mathcal{M}$, use a special function P that maps m got a subset $s \leftarrow P(m) \subseteq \{1,2,\cdots,n\}$

We will see examples of such P in just a minute. Signature is $\sigma := \{x_i\}_{i \in s}$

• Verification: To verify a signature on a message m the verifier checks that σ contains the pre-images of all public-key values $\{y_i\}_{i \in P(m)}$

Optimisations

Algorithm $G()$:

$k \xleftarrow{R} \mathcal{K}$

for $i = 1, \ldots, n$:

$x_i \leftarrow F(k, i) \in \mathcal{X}$

$y_i \leftarrow f(x_i) \in \mathcal{Y}$

output:

$pk = (y_1, \ldots, y_n)$

$sk = (k)$

Algorithm $S(sk, m)$:

$s \leftarrow P(m) \subseteq \{1, \ldots, n\}$

let $s := \{s_1, \ldots, s_\ell\}$

for $j = 1, \ldots, \ell$:

$\sigma_j \leftarrow F(k, s_j)$

output:

$\sigma \leftarrow (\sigma_1, \ldots, \sigma_\ell)$

Algorithm $V(pk, m, \sigma)$:

let $P(m) = \{s_1, \ldots, s_\ell\}$

let $\sigma := (\sigma_1, \ldots, \sigma_u)$

if $\ell = u$ and $f(\sigma_i) = y_{s_i}$

for all $i = 1, \ldots, \ell$

then output accept

otherwise output reject

Choice of P: If adversary wants to forge a message m' on m, and $P(m') \subseteq P(m)$, then signature on m also gives signature on m'

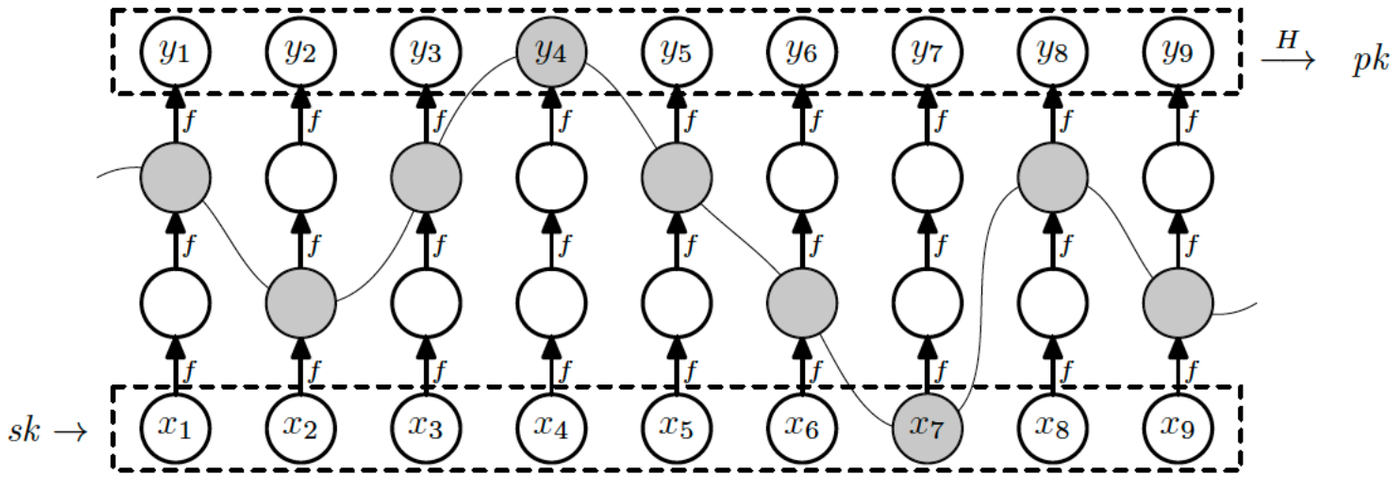
Security: Show that it is difficult for an adversary to find messages m and m' such that

$$P(m') \subseteq P(m)$$

How to build containment free functions?

Winternitz one-time signatures

- Hash-chains:
  $f^{(0)}(x) := x; f^{(1)}(x) := f(x); f^{(2)}(x) := f(f(x)); f^{(3)}(x) := f(f(f(x)))\ldots$ For $x \in \mathcal{X}$ the sequence $f^{(0)}(x), f^{(1)}(x), \ldots f^{(d)}(x)$is called a hash chain of length d+1. The value x is called the base of the chain and $y := f^{(d)}(x)$ is called



The secret key sk is used to derive $x_1, \ldots, x_9 \in \mathcal{X}$. The public key pk is the hash of $y_1, \ldots, y_9 \in \mathcal{X}$. The shaded circles represent the signature on a message m where $P(m) = (2, 1, 2, 3, 2, 1, 0, 2, 1)$. This $P(m)$ describes a cut through the rectangle illustrated by the thin line.

Winternitz Signature

- Keygen: Fix parameters n and d.
  - SK: $x_1, x_2, \cdots, x_n \xleftarrow{R} \mathcal{X}$,
  - PK: $y_i := f^{(d)}(x_i) \quad for \quad i = 1,2,\cdots,n$
- Sign: Let $I_d^n := (\{0,1,2,\cdots,d\})^n$.
  - Choose function $P : \mathcal{M} \rightarrow I_d^n$,
  - To sign m computer $s \leftarrow P(m)$, where $s = (s_1, s_2, \cdots, s_n) \in I_d^n$
  - $\sigma = f^{s_1}(x_1), \cdots f^{s_n}(x_n)$
- To verify compute $P : \mathcal{M} \rightarrow I_d^n$.
  - $\hat{y} = (f^{d-s_1}(\sigma_1), \cdots f^{d-s_n}(\sigma_n)) \in \mathcal{X}^n$
  - Security?

Types of PQC Signatures

S**tateless signatures.** The signer does not maintain any internal state between invocations of the signing algorithm. (SPHNIX+)

Stateful signatures. The signer has to maintain internal state between invocations of the signing algorithm. (XMSS)

One-time Signatures: Can be used only once (Lamport signature)

Many time Signatures: Can be used a finite number if times.

You can change a one-time signature to many-time signatures. (Boneh-Shoup Section 14.6.2)

Authenticated Key Exchange Mechanisms

Motivation

- Authenticated key exchange
- Two parties establish a shared secret over a public communication channel
- Many security definitions capturing various adversarial powers:
- Different types of authentication credentials: public key, shared secret key, password, identity-based, …
- Additional security goals: weak/strong forward secrecy (Forward secrecy protects past sessions against future compromises of keys or passwords.), key compromise, impersonation resistance, post-compromise security, …
- 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

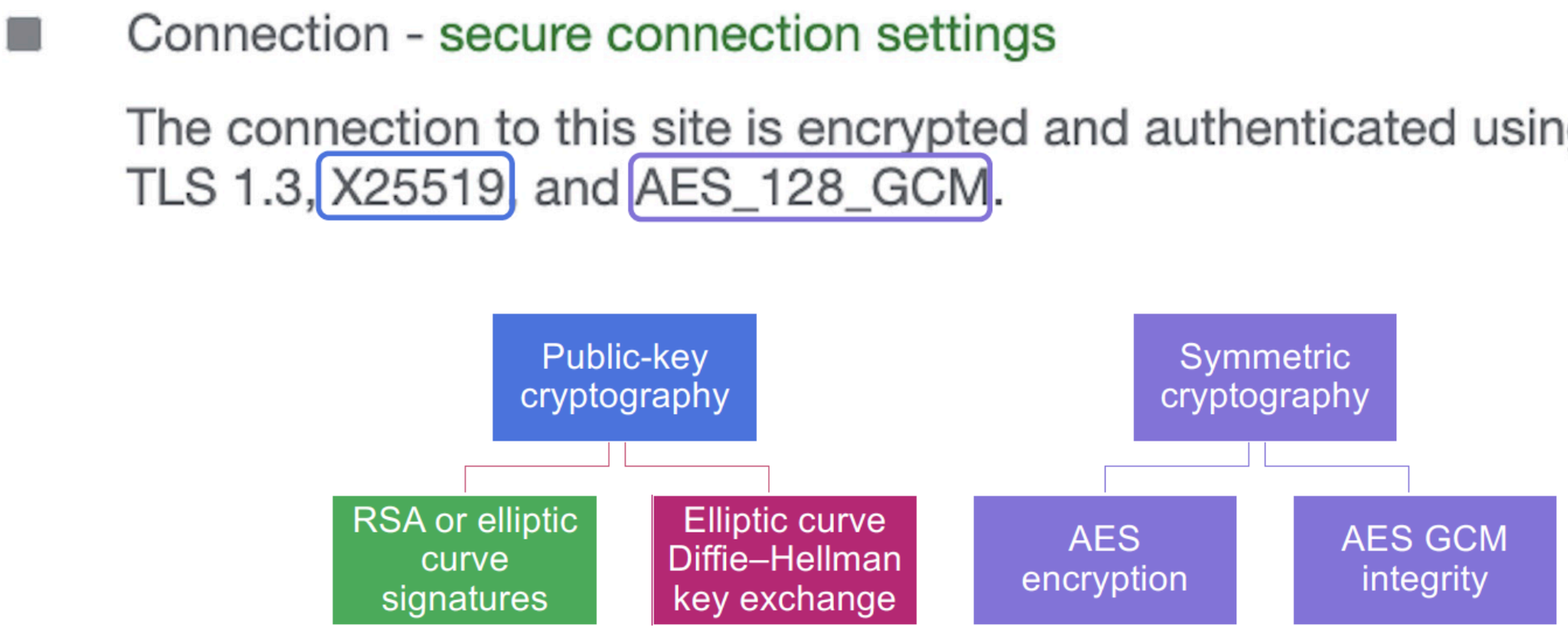
Explicitly authenticated key exchange: Signed Diffie–Hellman

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

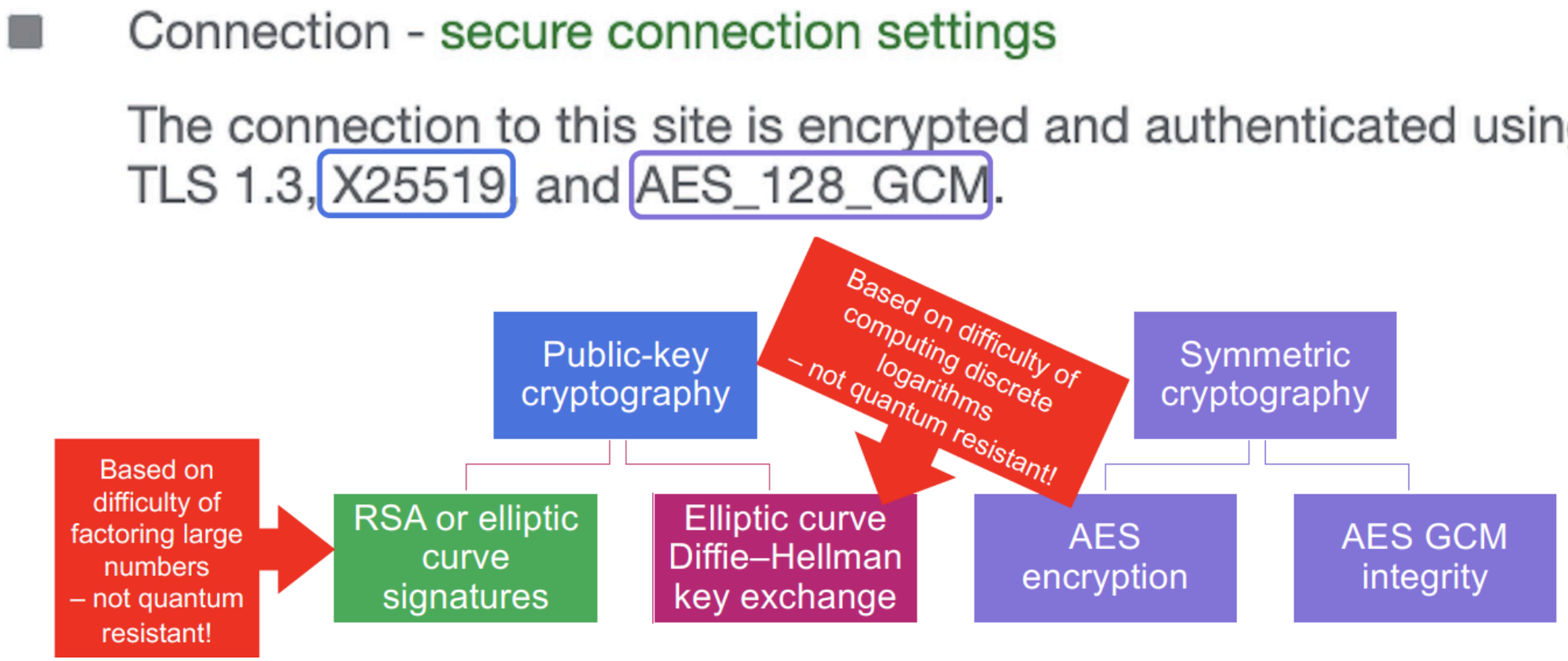
Implicitly authenticated key exchange: Double-DH

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

Cryptographic building blocks



Cryptographic building blocks



Hybrid-Approach (Classical + PQ)

- PQ security for early adopters without sacrificing current security
- "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 [GHP18], [BBFGS19]

[GHP18] Giacon, Heuer, Poettering. PKC 2018. https://eprint.iacr.org/2018/024
[BBFGS19] Bindel, Brendel, Fischlin, Goncalves, Stebila. PQCrypto 2019. https://eprint.iacr.org/2018/903

Functionality goals for hybridisation

- Backwards compatibility
  - Hybrid-aware client, hybrid-aware server
  - Hybrid-aware client, non-hybrid-aware server
  - Non-hybrid-aware client, hybrid-aware server
- Low computational overhead
- Low latency
- No extra round trips
- No duplicate information

Design Options

1. How to negotiate algorithms
2. How to convey cryptographic data (public keys / ciphertexts)
3. How to combine keying material

- How combine keying material
  - 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

[SFG19] Stebila, Fluhrer, Gueron. https://tools.ietf.org/html/draft-stebila-tls-hybrid-design-03

Draft Standards

- **NIST SP 800-56C**
  - "Recommendation for Key-Derivation Methods in Key Establishment Schemes" – includes various combiners
- **Hybrid key exchange in TLS** [SFG20]
- **Hybrid key exchange in SSH** [KSFHS20]
- ETSI

[NIST] https://csrc.nist.gov/publications/detail/sp/800-56c/rev-2/final
[SFG20] Stebila, Fluhrer, Gueron. https://tools.ietf.org/html/draft-ietf-tls-hybrid-design-01
[KSFHS20] Kampanakis, Stebila, Friedl, Hansen, Sikeridis. https://tools.ietf.org/html/draft-kampanakis-curdle-pq-ssh-00

Protocol Constraints

- TLS 1.2
  - Message size limit: 2^{24} bytes
  - Fragment size limit: 2^{14} bytes
  - OpenSSL key exchange message buffer: 20,480 bytes
    - FrodoKEM level 5: 21,600 bytes public key / ciphertext
    - Classic McEliece level 1: 261,120 bytes public key

Implementation patch to fix

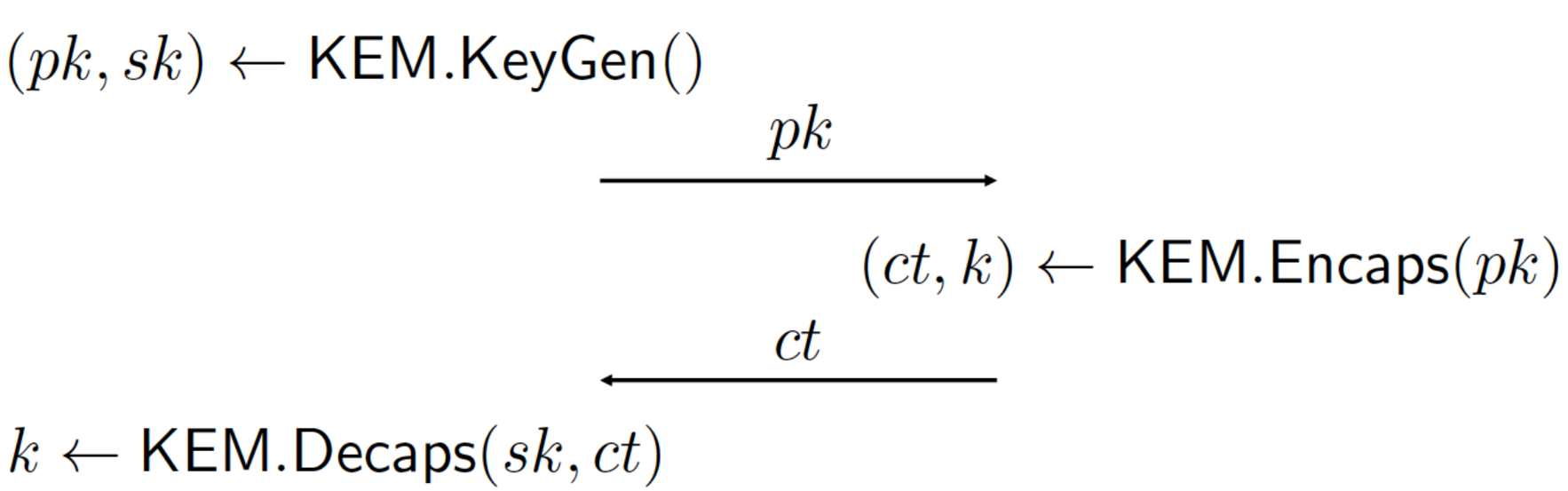
Need protocol changes to fix

- TLS 1.3
  - Key exchange message size limit: 2^{16} bytes (OpenSSL: 20,000 bytes)
  - Certificate size limit: 2^{24} bytes (OpenSSL $2^{16.6}$ bytes)
  - Signature size limit: 2^{16} bytes (OpenSSL 2^{14} bytes)
    - Picnic1 level 1: 34,000 bytes signature (but Picnic 3 is small enough)
    - Rainbow: 58KB-1.7MB public keys

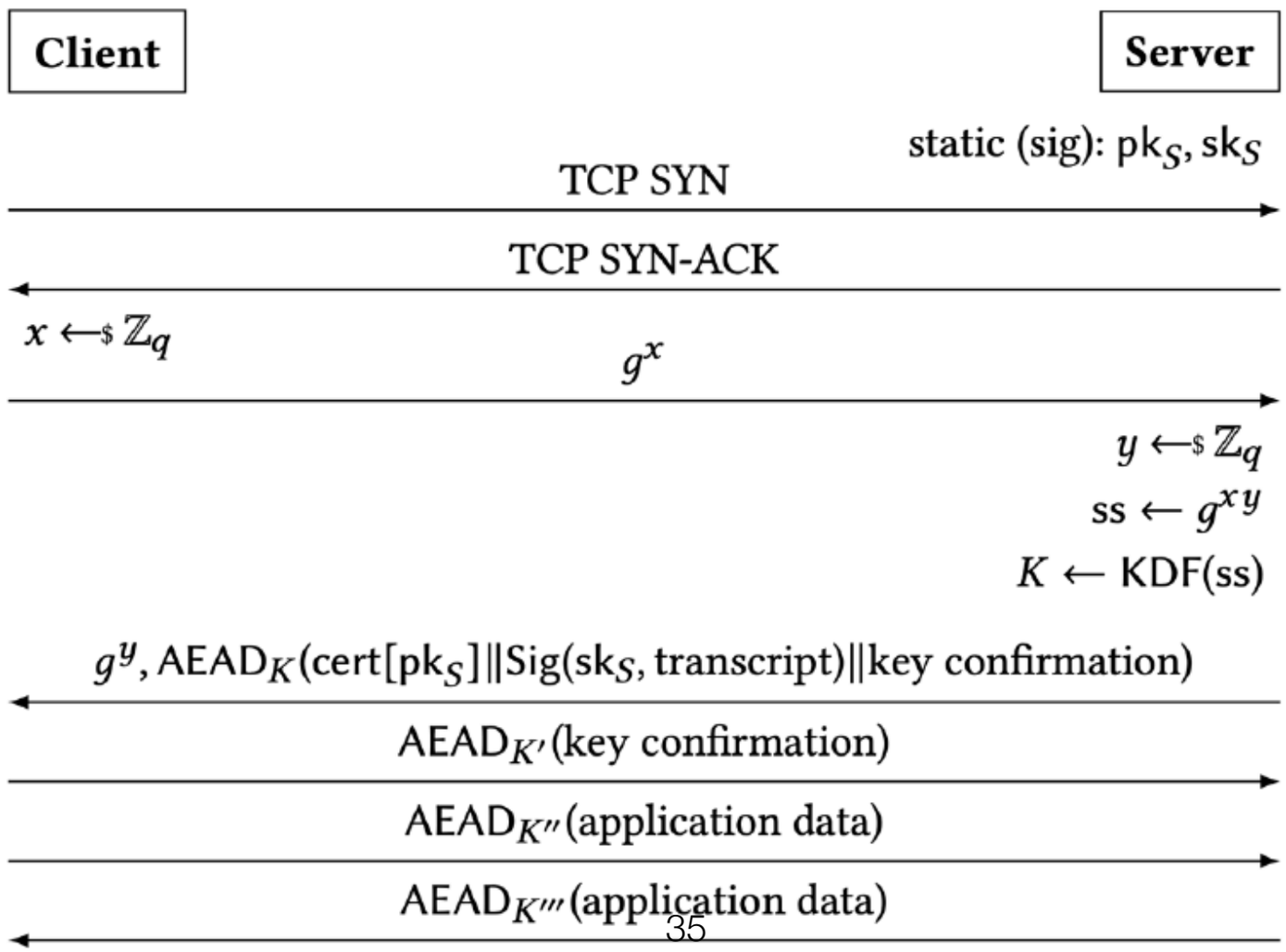
[CPS19] Crockett, Paquin, Stebila. NIST 2nd PQC Standardization Conference 2019. https://eprint.iacr.org/2019/858

Key Encapsulation Mechanisms (KEMs)

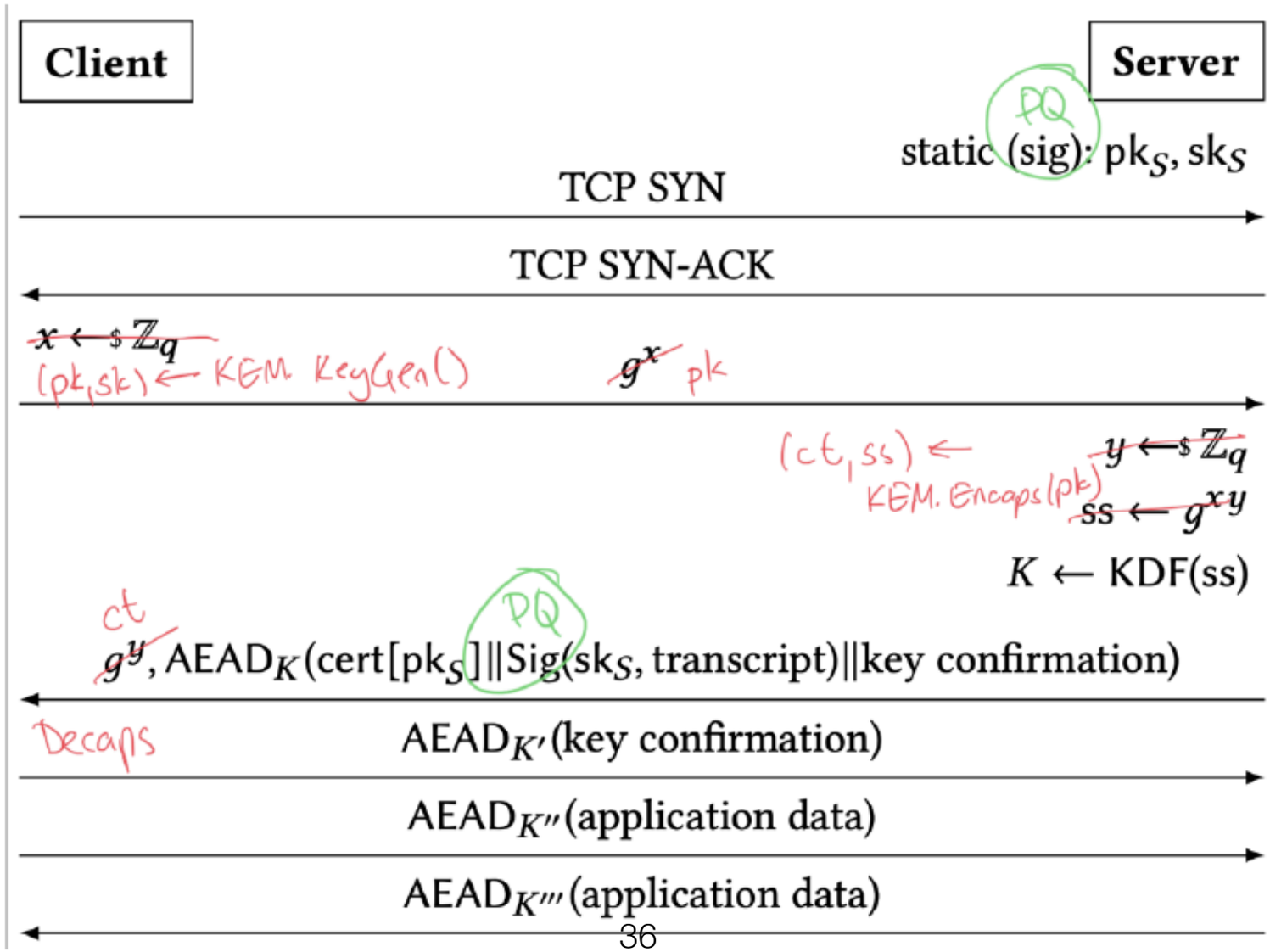
An abstraction of Diffie–Hellman key exchange



TLS 1.3 Handshake (Signed Diffie-Hellman)



TLS 1.3 Handshake (Post-Quantum)



Post Quantum Signatures are big

Signature Schemes

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

Use Post Quantum KEMS for Authentication

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

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

22

Implicitly authenticated Key Exchange

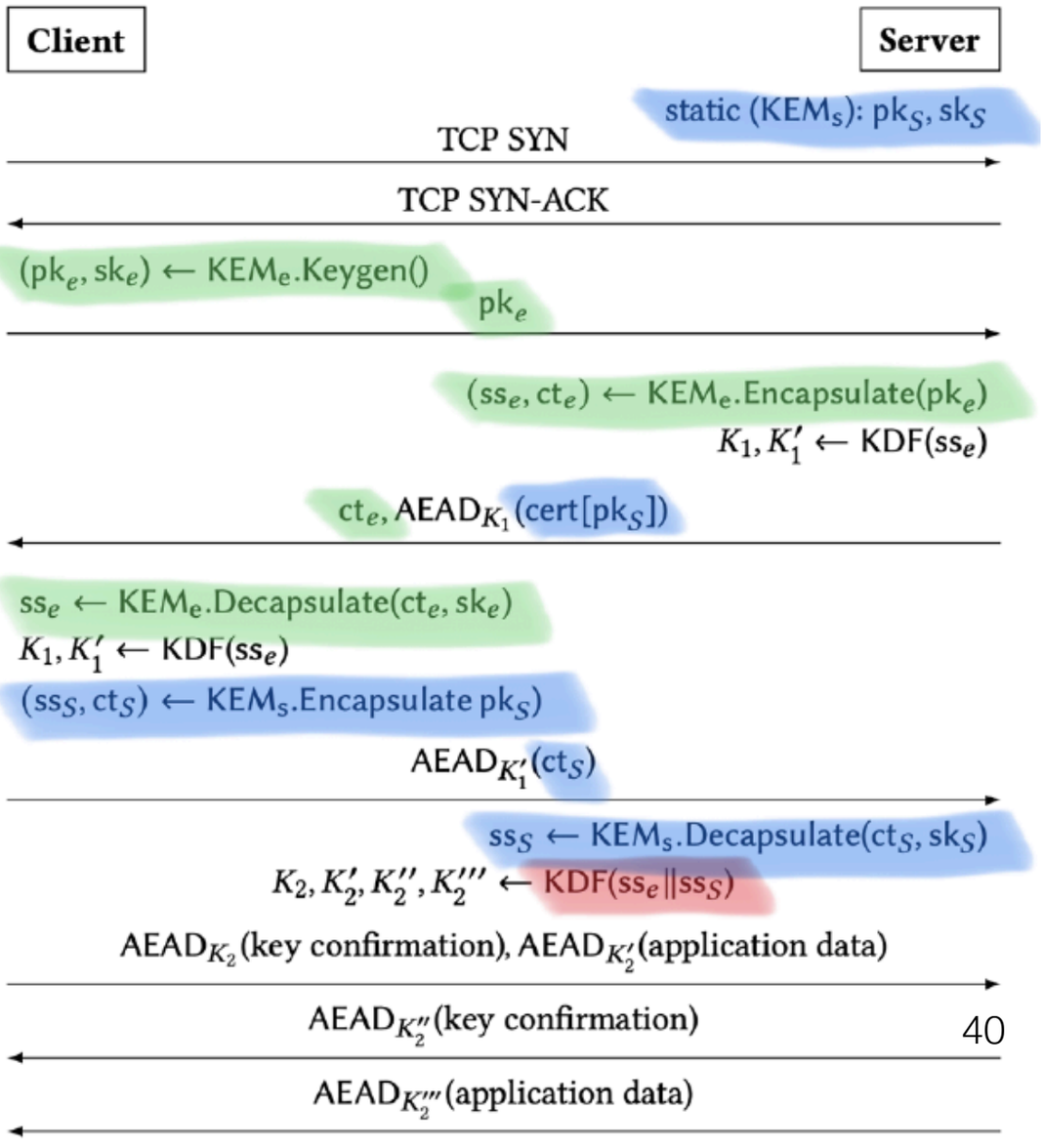
In theory

- DH-based:
- KEM-based:

In practice

- RSA key transport in TLS ≤ 1.2
  - Lacks forward secrecy
- Signal, Noise, Wireguard
  - DH-based
  - Different protocol flows

KEMTLS Handshake



KEM for
ephemeral key exchange

KEM for
server-to-client
authenticated key exchange

Combine shared secrets

Algorithm Choices

KEM for ephemeral key exchange

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

KEM for authenticated key exchange

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

Signature scheme for intermediate CA

- Want small public key + small signature

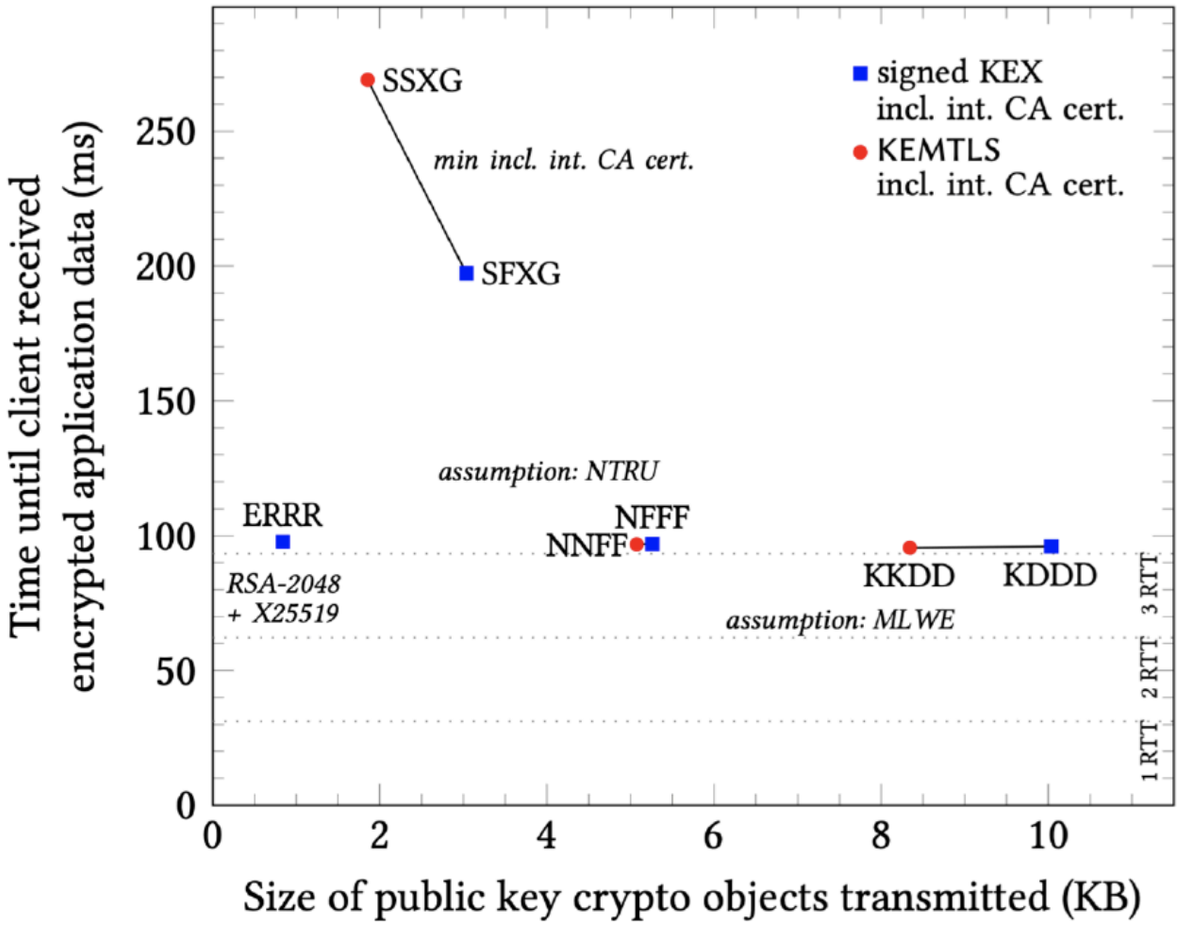
Signature scheme for root CA

- Want small signature

4 Scenarios

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

Signed Key Exchange Vs KEMTLS



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

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

Observations

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

Security Subtleties: Forward Secrecy

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

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

PQC Transition

- Create Inventory of cryptographic algorithms, and softwares
- Evaluate risk of each
- Plan mitigation strategy

Thank you!