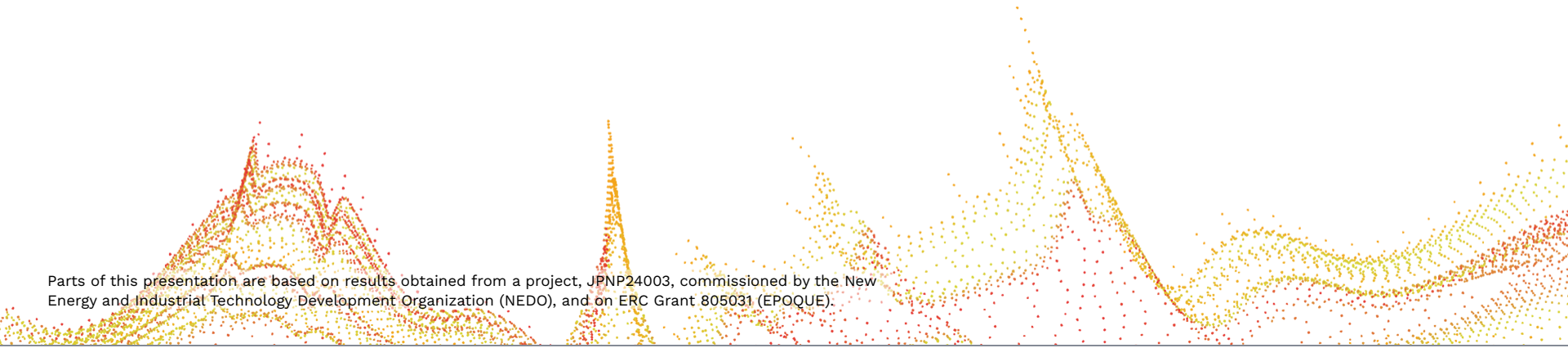
Migrating protocols to PQ: the good, the bad, and the ugly

Thom Wiggers



Parts of this presentation are based on results obtained from a project, JPNP24003, commissioned by the New Energy and Industrial Technology Development Organization (NEDO), and on ERC Grant 805031 (EPOQUE).

Aysajan Abidin

Invitation to speak at workshop co-located with Eurocrypt 2026

Aan: Thom Wiggers

Hi Thom,

I am reaching out to regarding a workshop COSIC KU Leuven is organizing on 9 May, 2026.
The workshop will focus on PQ protocols and will be in Rome, co-located with Eurocrypt.
I want to invite you in advance for a talk on PQ TLS or any related topics.

“Sure, I can give a talk on some work I did on

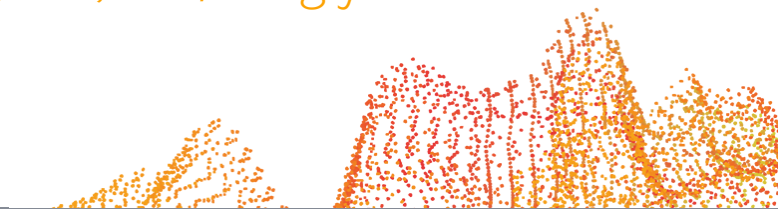
PQ TLS **or** Signal **or** WireGuard

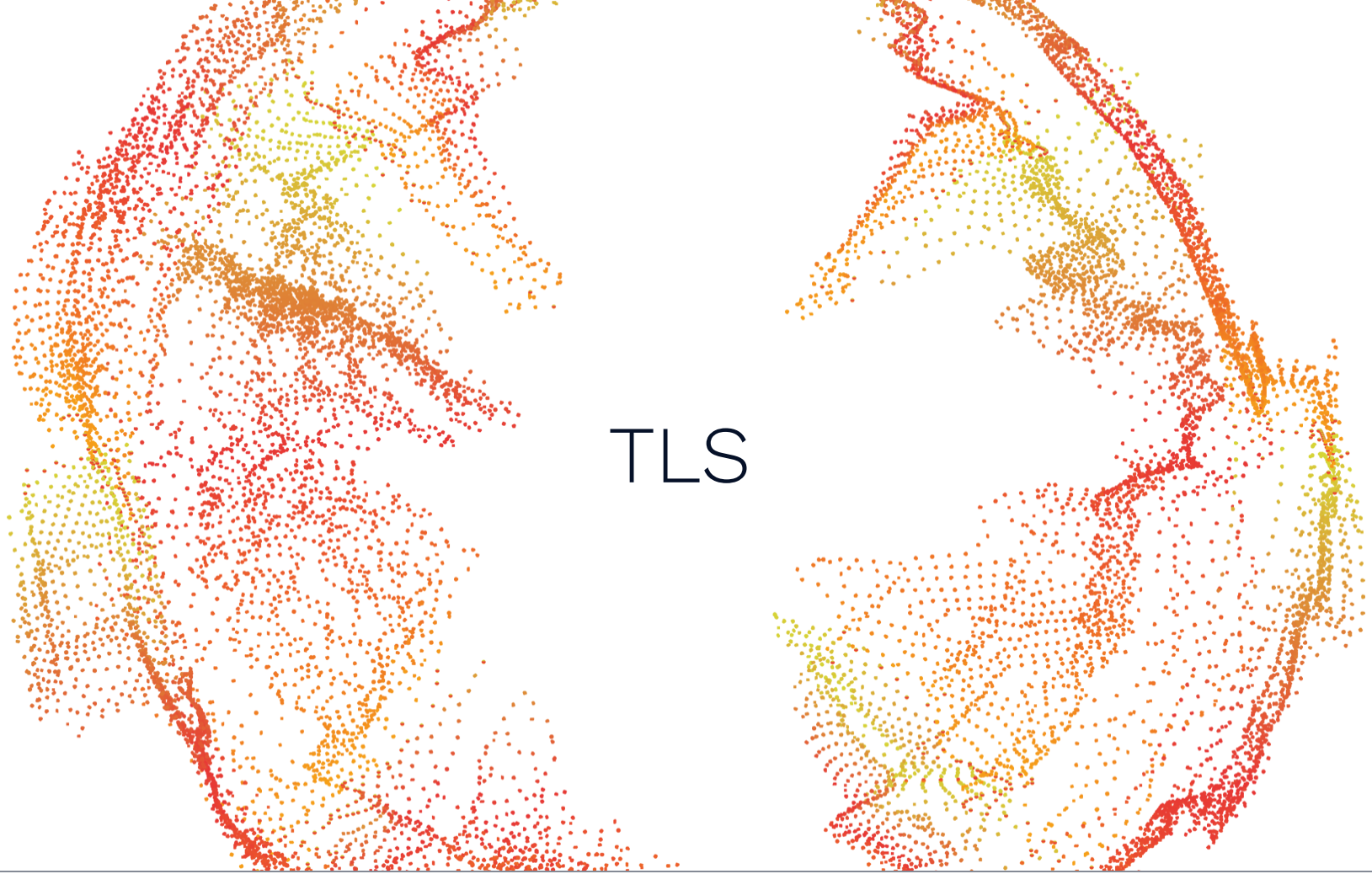
“Yes”

Outline

- Post-Quantum TLS
- PQ Signal
- PQ WireGuard

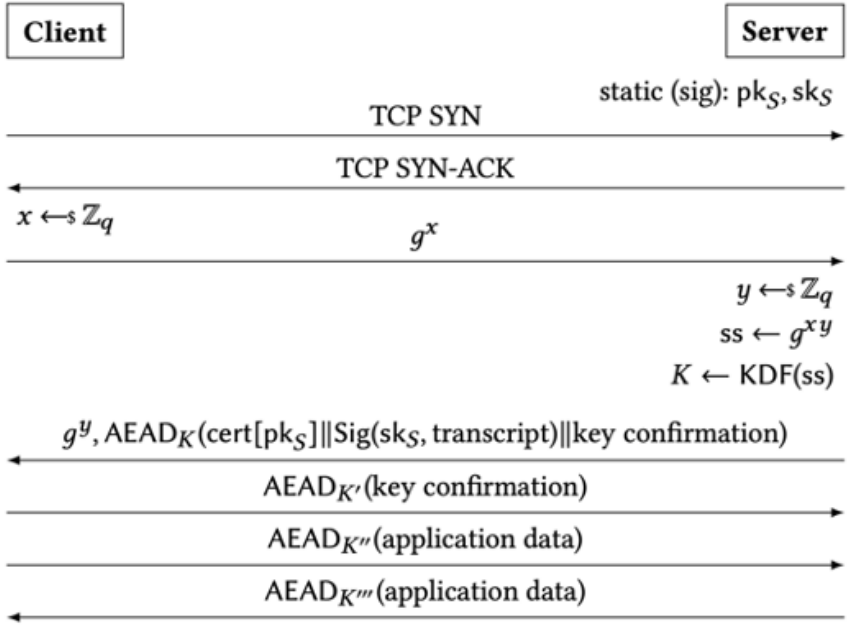
You decide which of these is good, bad, and/or ugly





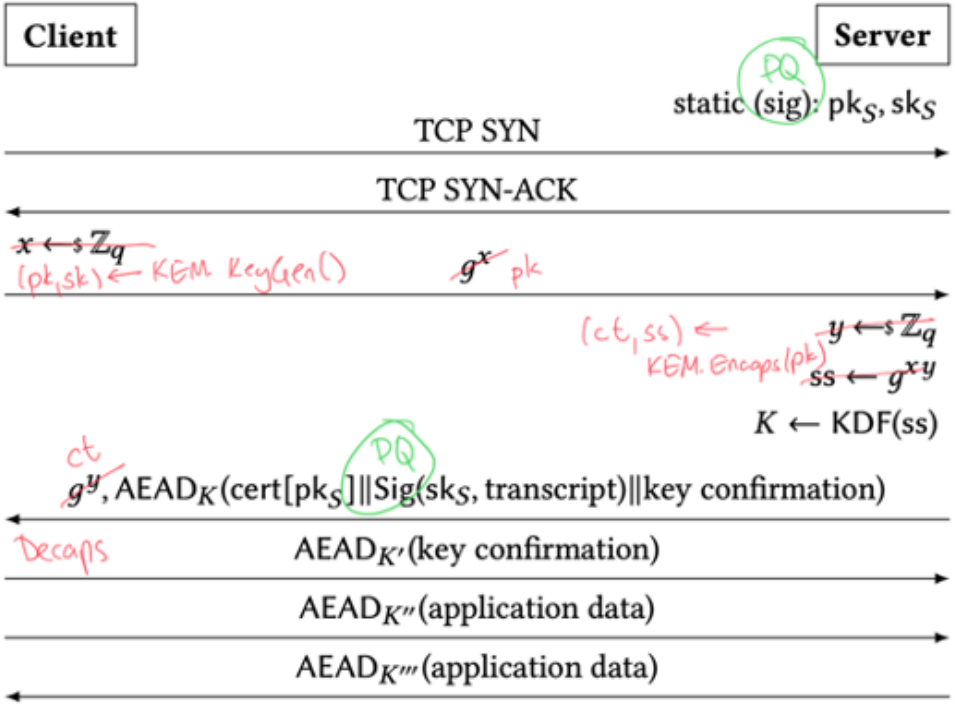
TLS 1.3 full handshake

- Key exchange via **ECDH**
  - only ephemeral key exchange
- Server Authentication: **Signature**
- Handshake authentication: HMAC-SHA256
  - “key confirmation”
- AEAD: only AES-GCM or ChaCha20-Poly1305



TLS 1.3 overview
K, K’, K”, K”’: bunch of purpose-specific keys

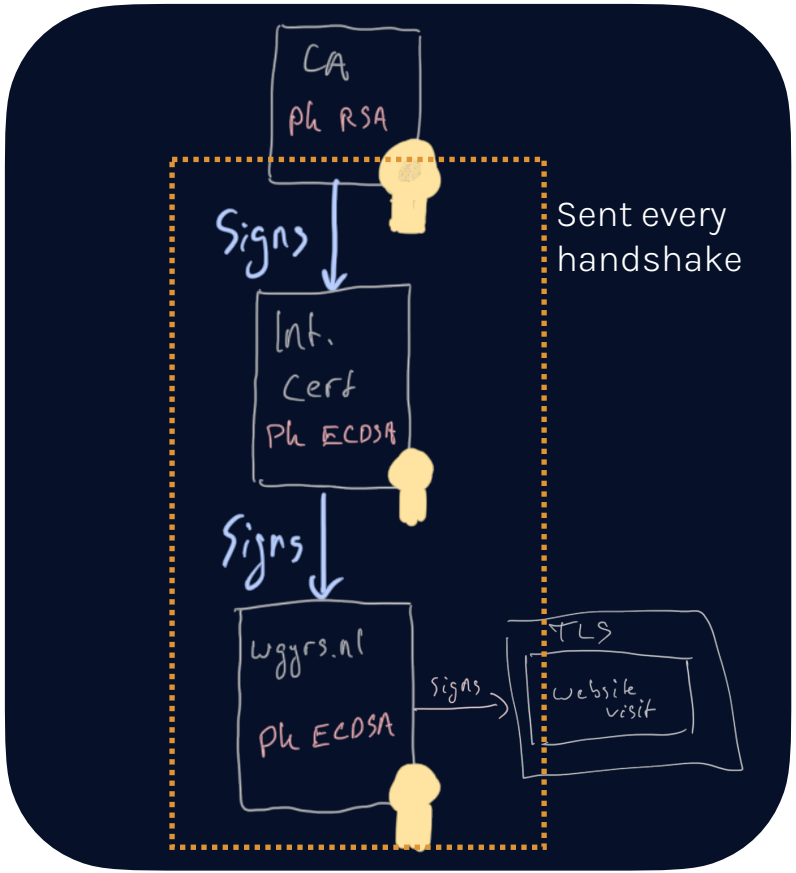
Post-Quantum TLS 1.3



Authentication in TLS

- There are **many** signatures in TLS
- Server certificate:
  - 1x public key, 1x signature
  - Bonus: 3x **Certificate Transparency** SCTs
  - 1x Signature on handshake
- Intermediate Certificate:
  - 1x public key, 1x signature

Replacing all of this by ML-DSA adds **18–36 kB!**



Certificate Transparency

- Certificate transparency is a **public log** of all issued certificates
  - In particular, a Merkle Tree
- **Aim:** detect “DigiNotar” incidents
  - Hacked CA issued certificates for gmail.com
- Chrome, Firefox, Safari require Certificate Transparency

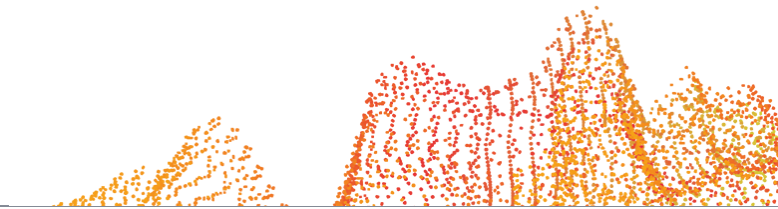
Certificate Transparency is fragile

- Running a log is hard
  - Many failure modes lead to **log disqualification**
- Running a log is **expensive**
  - ~20 TB disk per year
  - ~10 TB per day bandwidth
  - Before PQC

Let's Encrypt presentation in PLANTS meeting at IETF 124

Certificate Transparency uses Trees

- CT logs only accept submissions from trusted CAs
  - The list only contains validated certificates
- Merkle Trees can produce a proof of inclusion in the tree
- So a **proof of inclusion could prove validity** of the certificate!
  - Inclusion proof is just a bunch of hashes



MTC: Merkle Tree Certificates

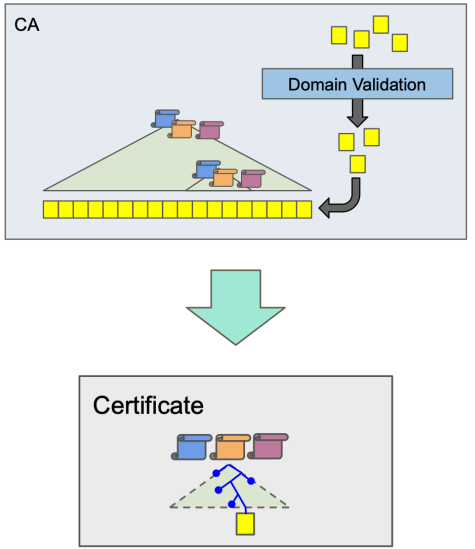
- A traditional CA first **signs**, then **logs** the result and **collects SCTs**
  - CT was layered on top of existing PKI
  - Complications stem from the gap between issuance and logging
- A Merkle Tree CA first **logs**, then **signs** a checkpoint and collects **co-signatures**
  - Each CA runs a separate issuance log
  - Independent mirrors and witnesses co-sign log state

https://datatracker.ietf.org/meeting/124/materials/slides-124-plants-solution-space-and-dispatched-work-00

MTC: Issuance and Verification

- CA creates certificates and builds Merkle Tree
- Tree state gets checked and tree head is signed by co-signers
- Certificate is now:
  - Public key
  - Path in Merkle Tree
  - The (co)signatures on the tree head

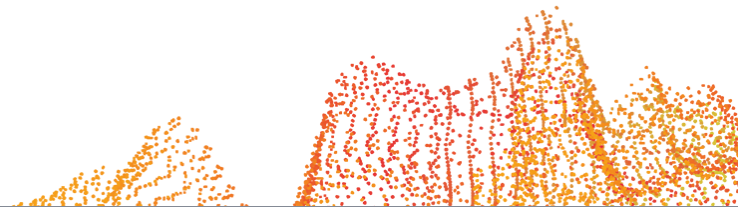
- Note that these signed tree heads are **shared** with many log entries, and can be **distributed out-of-band**!



https://datatracker.ietf.org/meeting/124/materials/slides-124-plants-solution-space-and-dispatched-work-00

MTC: Transparency sustainability

- Instead of many distinct logs with all certificates, CA now maintains its one list
- Cosigners ensure transparency properties
- Compared to CT **today**, MTC log entries are 12x smaller
  - 97x smaller than CT+ML-DSA-44
- Mirrors now provably identical
  - monitors only need to download one time
- More feasible to run mirrors
  - Load spread out more
- Log errors/downtime only means issuance failure, no log disqualification



MTC: Optimised certificates

- 3 ML-DSA signatures: 7,260 bytes (1 CA sig, 2 SCTs)
- 1 inclusion proof: **736** bytes
  - Tree size depends on issuance rate
  - (estimated from Web PKI / Let's Encrypt volumes)
- No longer a need for intermediate certificates

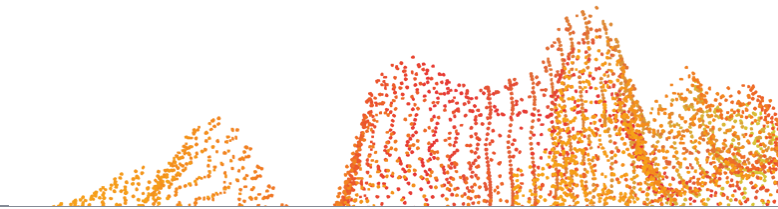
https://datatracker.ietf.org/meeting/124/materials/slides-124-plants-solution-space-and-dispatched-work-00

MTC: How to deploy this?

- TLS **Clients** (browsers) must be updated with support
  - Ability to **fetch tree heads** out-of-band if you want maximum performance
- TLS **Servers** must be updated with support
  - Clients will indicate which tree-heads they have
  - Server selects "**full**" or "**signature-less**" MTC certificate
    - "Full" will include the signatures on the tree head
    - "Signature-less" omits signatures if client is up-to-date with tree heads
  - Server must update its inclusion proof periodically
    - Likely via **ACME** (aka Let's Encrypt's "certbot")

What about “old-fashioned” certificates

- MTC will (likely) only work in WebPKI and similar settings
- “Old-fashioned” certificates with chains of signatures will probably continue to exist
  - Fall-back in the WebPKI
  - Private PKIs
  - Non-Web use cases



How real is this?

- Proposal is being pushed by Google Chrome and Cloudflare
- Standards are being developed in the **PLANTS** (PKI, Logs, And Tree Signatures) working group in IETF
  - Disclaimer: I am co-chair of this working group
  - Please review and comment
  - Join the mailing list plants@ietf.org

- Cloudflare and Google announced they will start experimenting on the web in 2026

What is MTC really telling us?

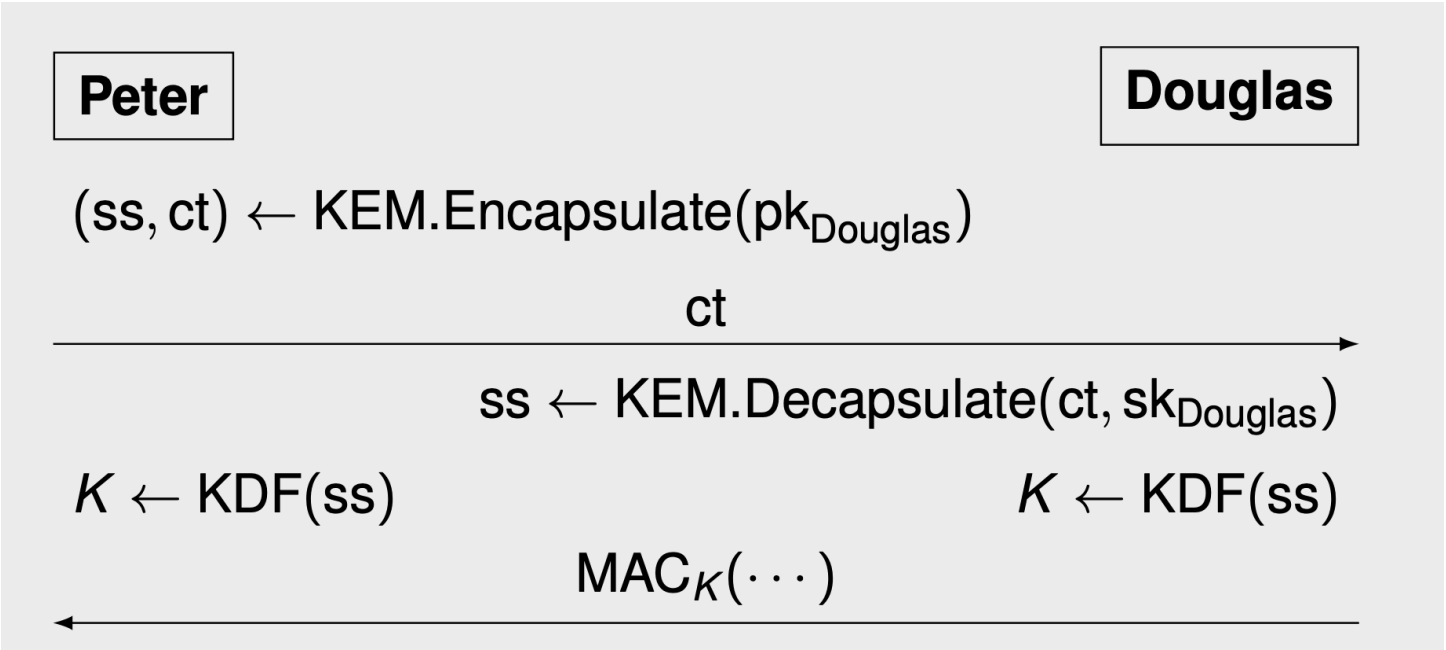
- Things have been built based on certain assumptions that may no longer be true
- Things have been built over time

The transition to post-quantum cryptography is perhaps a once-in-a-lifetime opportunity to re-examine the status quo and make big changes

Re-considering assumptions

- Traditionally, signatures similar to key exchange
  - Famously, RSA signing is “encryption in reverse”
- **Observation:** ML-KEM is quite a bit smaller than ML-DSA!
  - What if we use KEMs for authentication instead of signatures?

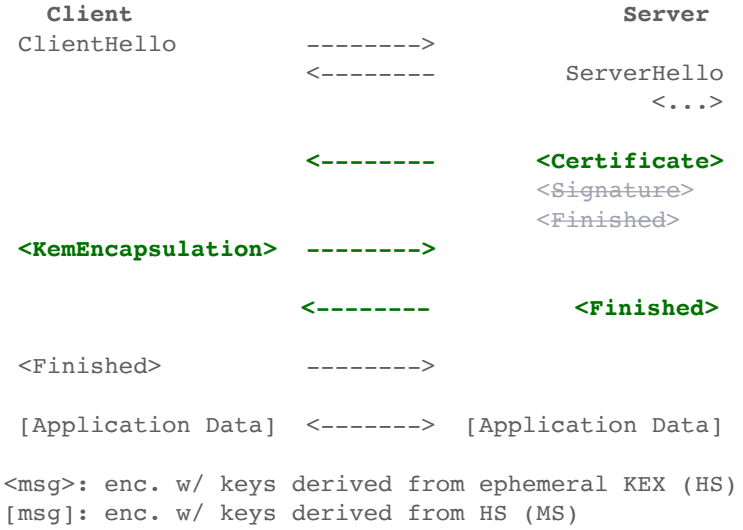
Authenticated Key Exchange via KEM



Note that this protocol assumes that we have already exchanged the public keys!

But, KEMs are not the same as signatures

- Signatures let you authenticate **non-interactively**
- KEMs are inherently **interactive**
- **Naively**, this means that KEM authentication would require an additional round-trip in TLS
- TLS 1.3 was specifically designed to reduce the number of round-trips to **just 1**!

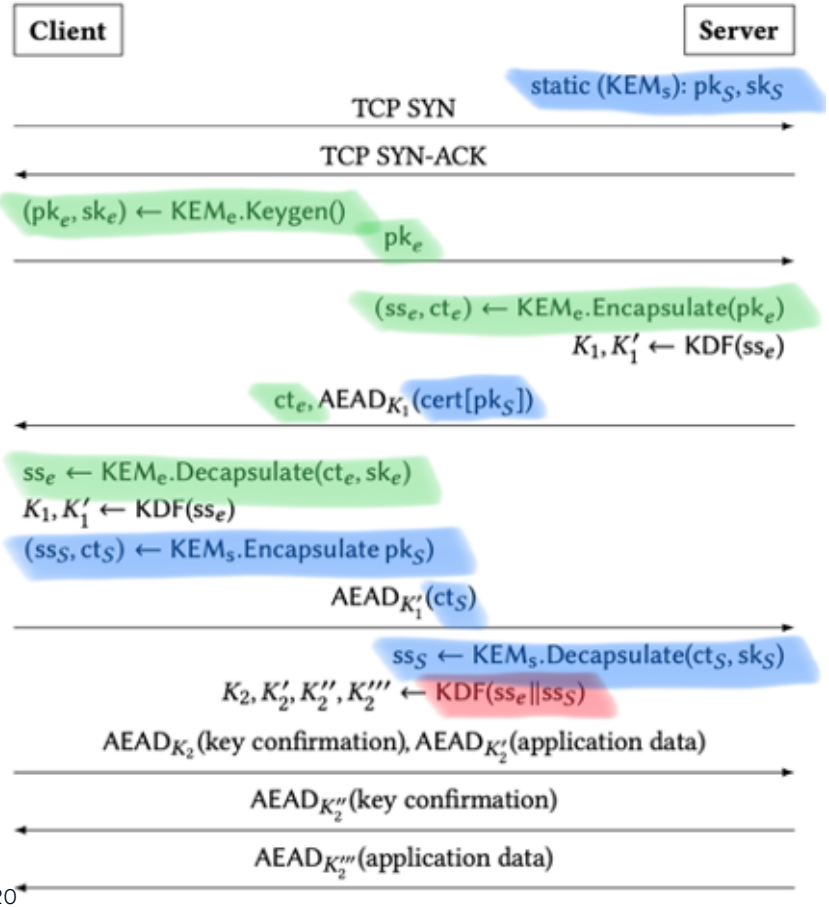


KEMTLS

KEM for
ephemeral key exchange

KEM for
server-to-client
authenticated key exchange

Combine shared secrets



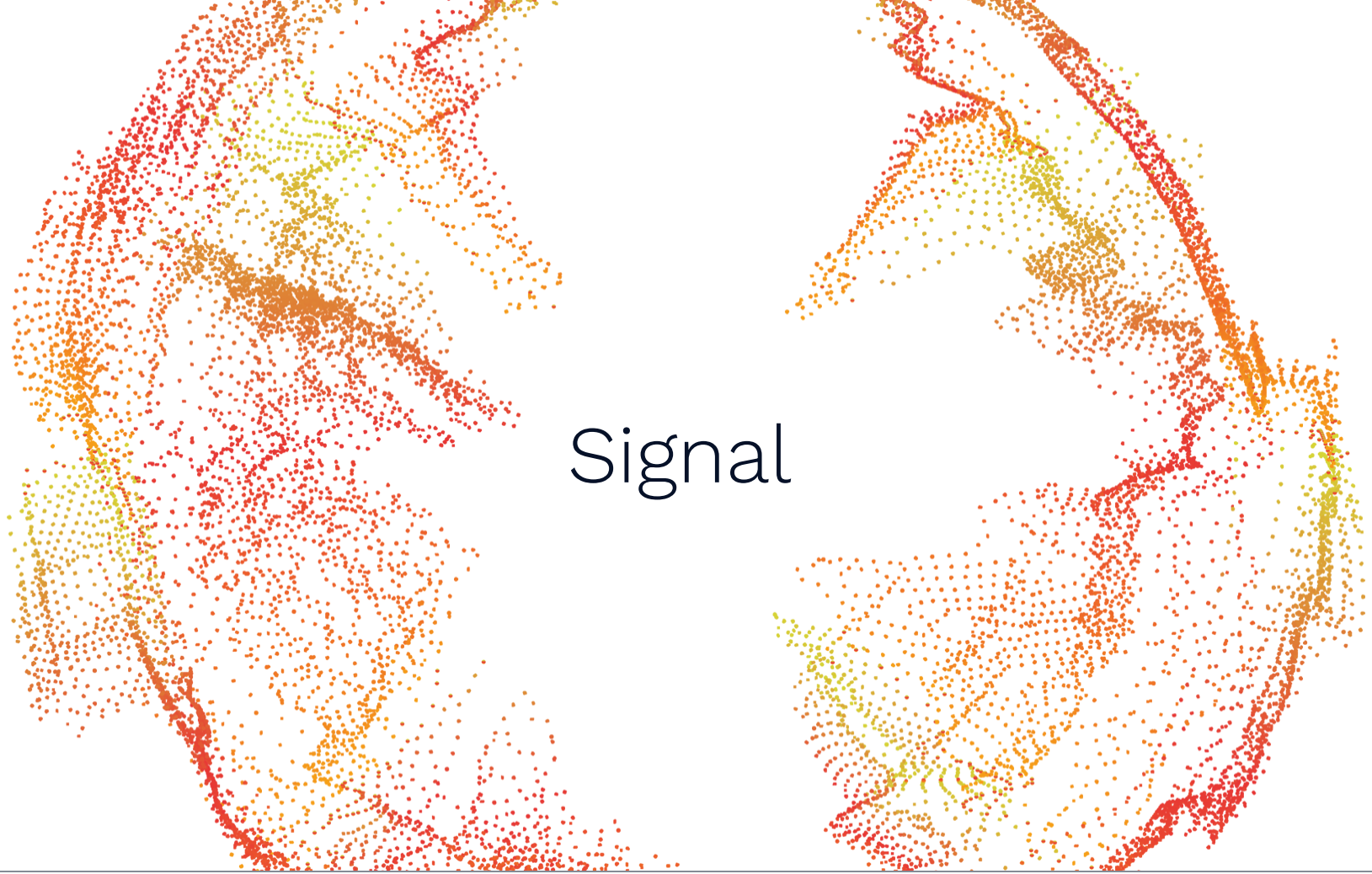
Schwabe, Stebila, **Wiggers**. Post-Quantum TLS without handshake signatures. ACM CCS 2020

Re-considering the status quo

- **Merkle Tree Certificates**
  - Re-designing the WebPKI from transparency first, instead of bolting it on
  - Leaning into the strengths of the Merkle Trees already used in transparency

- **KEMTLS**
  - Why are we even using signatures?
  - Use *implicit authentication* to avoid the round-trip cost of KEMs

- **Further reading**
  - **MTC:** https://datatracker.ietf.org/wg/plants/about/
  - **Post-Quantum TLS & KEMTLS:** https://thomwiggers.nl/publication/thesis
  - **AuthKEM (KEMTLS) IETF draft:** https://datatracker.ietf.org/doc/draft-celi-wiggers-tls-authkem/

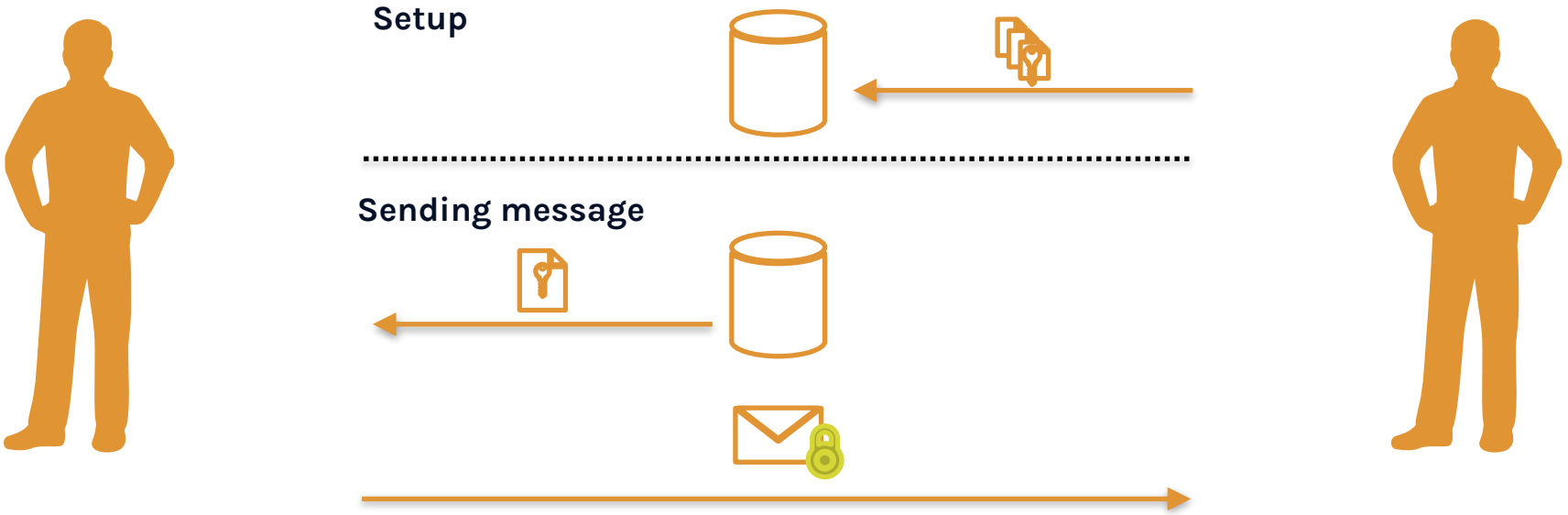


Messaging

Messaging

How do I still send messages when the recipient is unavailable?

Signal key exchange



X3DH

- Signal's well-known key exchange protocol
- Heavily relies on **non-interactive DH key agreement** for authentication

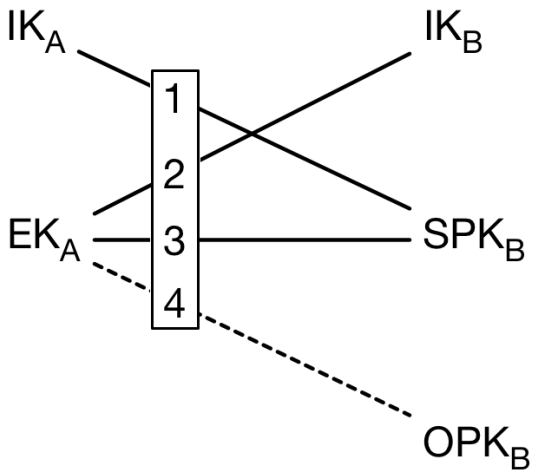


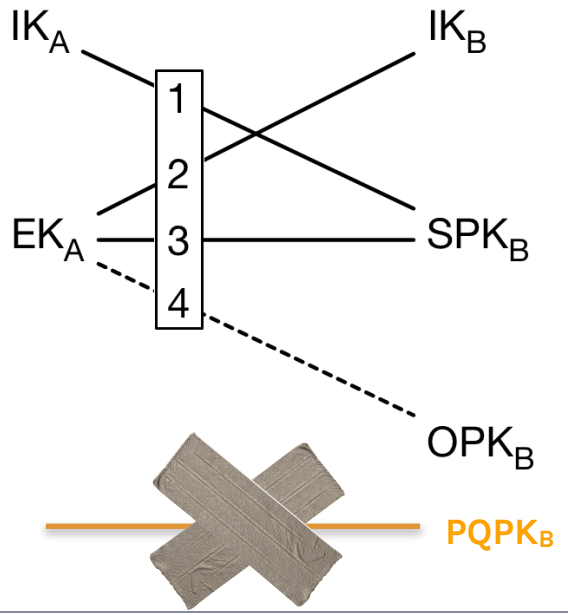
Image from Signal's X3DH spec

~~X3DH~~ PQXDH

- Signal's well-known key exchange protocol
- Heavily relies on **non-interactive DH key agreement** for authentication

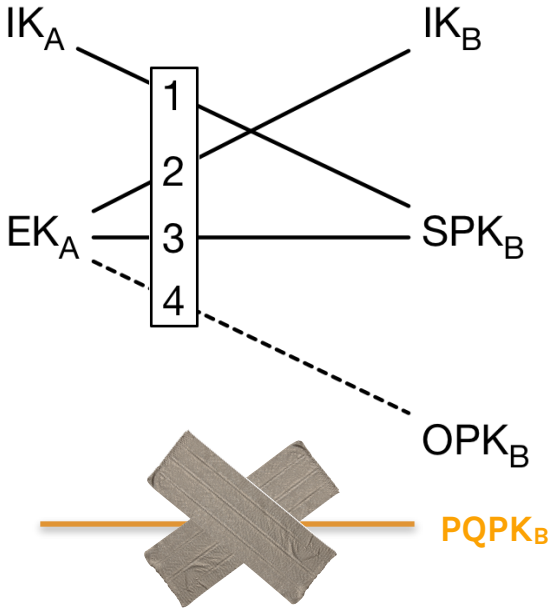
- **PQ:** tape a PQ KEM on top

- Only offers **Harvest-Now Decrypt-Later** security
- A fully post-quantum solution is still open



Towards fully PQ Signal key exchange

- In (PQ)X3DH, computing DH(IK_A, SPK_B) authenticates Alex to Blake
- This is possible because DH is **Non-Interactive**
- **KEMs are Interactive**
- Signatures are non-interactive but also **non-repudable**
- Signal requires **deniability**



Ring Signatures

- Users U_i each have (vk_i, sk_i)
- Signing: $Sig \leftarrow RingSign(sk_i, msg, \{vk_i, vk_j, vk_k, vk_m, \ldots\})$
- Verification: $1/0 \leftarrow RingVerify(\{vk_i, vk_j, vk_k, vk_m, \ldots\}, msg, sig)$
- Should be hard to tell which user of $\{vk_i, vk_j, vk_k, vk_m, \ldots\}$ was the true signer!

Deniable Authentication from Ring Signatures

- Alice generates and publishes Ring signature keypair (vk_A, sk_A) and throw away sk_A
- Bob who wants to authenticate to Alice signs msg with Bob's sk_B and ring $\{vk_A, vk_B\}$
- Now: even if you get `sig` unclear who produced the signature, **Alice or Bob**!
- But, **Alice knows** Bob must have produced the signature

RingXKEM: Deniable KEX for Signal

- Use **KEMs** for key establishment and implicit, deniable authentication of Bob
- Use deniable PQ Ring signatures for deniable authentication for Signal
  - Building on Brendel et al. (PKC22), Hashimoto et al. (PKC21, JC22)
  - New notion of **deniable Ring signatures** (instead of *anonymous RS*) allows for efficient, provable constructions and proposing **RingFalcon and RingMAYO** (building on Gandalf, [C:GaJaKi22])
- Use **Merkle Trees** to optimize delivery of **prekey bundles** by amortising the cost of signatures
- Security **proof** in a **new model** for **Bundled Authenticated Key Exchange** protocols that allows **direct comparison** of BAKE protocols (X3DH, PQXDH, RingXKEM)
  - Hashimoto, Katsumata, Wiggers. Bundled Authenticated Key Exchange: A Concrete Treatment of (Post-Quantum) Signal's Handshake Protocol. USENIX 2025
- **Deniability analysis** in a framework that allows **directly comparing** different BAKE protocols
  - Katsumata, Niot, Tucker, Wiggers. Comprehensive Deniability Analysis of Signal Handshake Protocols: X3DH, PQXDH to Fully Post-Quantum with Deniable Ring Signatures. USENIX 2025

Table 1: Signal key exchange protocols and their deniability and security properties

Signal handshake protocol deniability properties

	Protocol:	X3DH	PQXDH	PQXDH	RingXKEM	SignXKEM
		Classic $\mathcal{A}$/D	Classic $\mathcal{A}$/D	Classic $\mathcal{A}$ Quantum D	Classic or Quantum $\mathcal{A}$/D	Classic or Quantum $\mathcal{A}$/D
	Deniability Level	**Leakage** leak disc	**Leakage** leak disc	**Leakage** leak disc **QROM**	**Leakage** leak disc **QROM**	**Leakage** leak disc **QROM**
	local	● ●	● ●	● ● ✔	● ● ✔	◕ ◔ ✔
	global	◕ ●	◕ ●	◕ ● ✔	◓ ● ✔	◕ ◔ ✔
strong-	local	●† ●	●† ●	$●^{SO}$ ● ?	$●^{SO}$ ● ?	$◕^{SO}$ ● ?
strong-	global	◕† ●	◕† ●	$◕^{SO}$ ● ?	$◓^{SO}$ ● ?	$◕^{SO}$ ● ?
	Security [HKW25]	Classical	Harvest-Now Decrypt-Later		Fully post-quantum	

Legend

Icon	Last-resort prekey: No leak/disc	Last-resort prekey: Yes leak/disc
●	high	high
◕	high	med
◓	med	med
◔	med	low
○	low	low
?	Open problem	
$○^{SO}$	Accusers $\mathcal{A}$ restricted to being senders, no deniability otherwise.	
†	Proof using GGM.	

Example: RingXKEM is local deniable with leakage leak = high and disclosure disc = high even if a last-resort prekey bundle was used. SignXKEM is local deniable with leak = high and disc = med, but restricted to leak = med and disc = low using a last resort prekey bundle.

Remark: For strong deniability, we always set disc = high, since we have no control over the information a malicious accuser may reveal.

Deniability is very subtle!

Deployment by Signal

- Signal already deployed **Triple Rachet / “SPQR”** which gives post-quantum **Post-Compromise Security** (see one of many nice talks e.g. at RWC 2025)

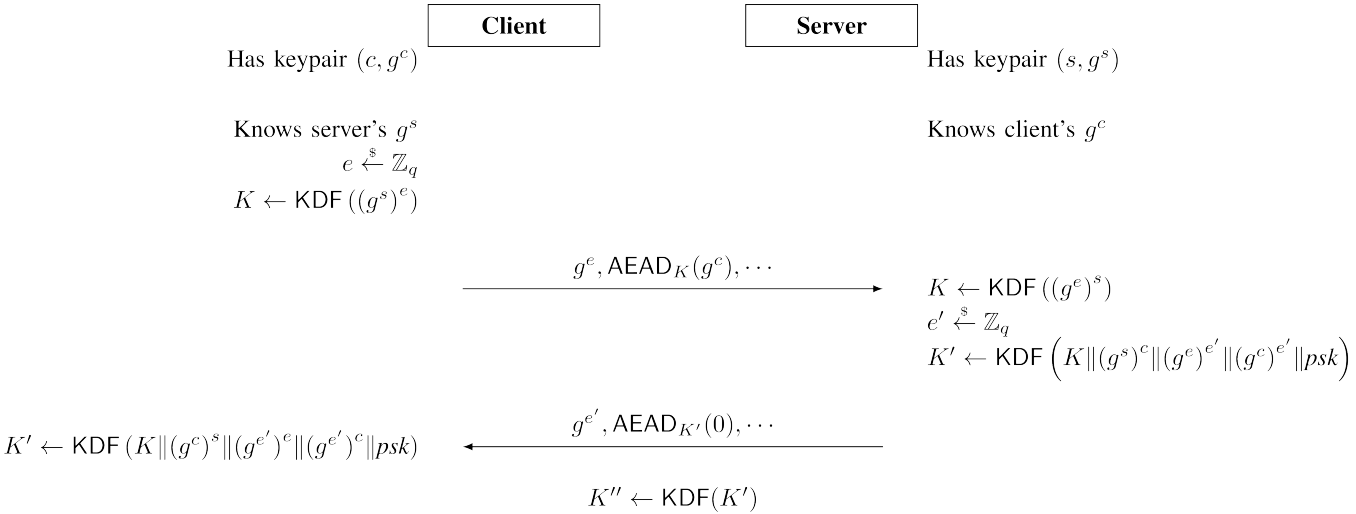
- Signal intends to deploy RingXKEM **hybridised** with **XHMQV** (Fiedler, Günther, Pan, Zeng, Crypto 2025)
  - XHMQV: essentially, X3DH x HMQV -> a more optimized **classic** protocol
  - Hopefully some time in 2026

Questions about Signal?

WireGuard

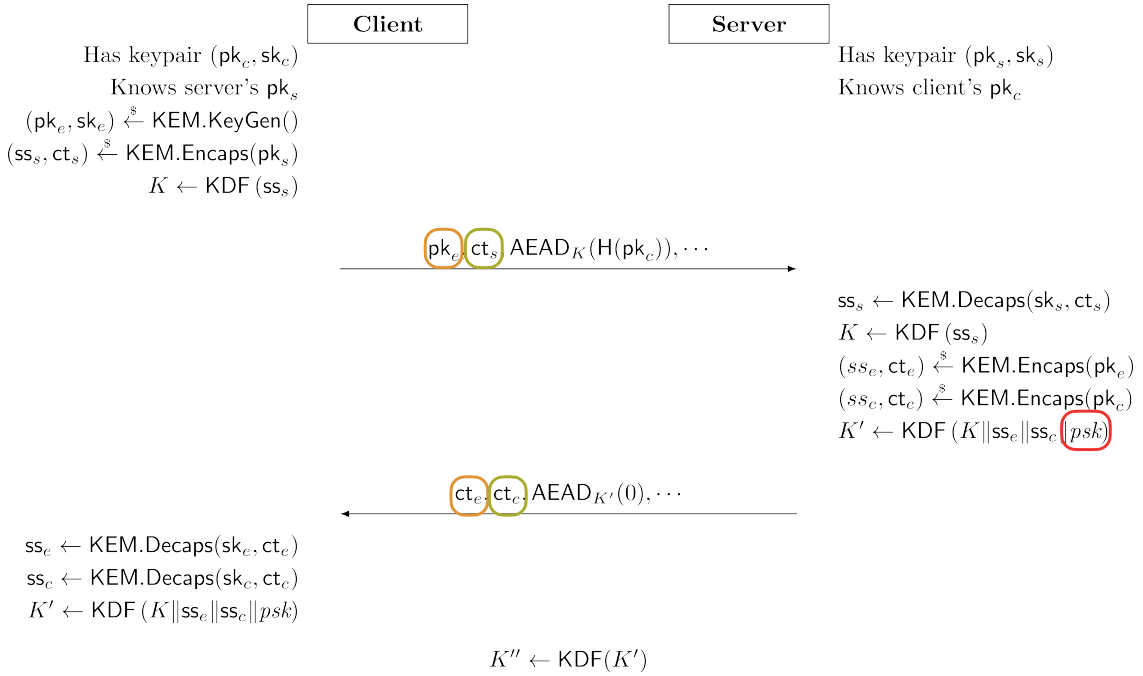
- Fairly recent but popular VPN protocol by Jason Donenfeld
- Looked at other VPN protocols and said “that’s way too complicated”
- **Opinionated** design allows many optimisations and a small implementation
  - Only X25519, ChaCha20-Poly1305
  - 32-byte public keys == user identities: no certificates, PKI, …
- Resilient to abuse
- Implemented in the Linux kernel

Pre-Quantum WireGuard



High-Level PQWG handshake

- KEMs instead of ephemeral kex
- 2 KEMs for mutual auth
- Send hash instead of pk for space
- PSK to authenticate first message
  - No DH NIKE



Hülsing, Ning, Schwabe, Weber, Zimmermann, Post-Quantum WireGuard. IEEE S&P 2021

Key exchange theory meets practice

- For technical reasons, WireGuard's handshake runs over **UDP**
- UDP does not support **packet fragmentation**
- This means that we support a **maximum payload of ~1200 bytes** assuming the minimum IPv6 MTU

		X25519	ML-KEM-512
Initial message	Ephemeral pk	32 B	800 B
	Client id pkC	32 B	800 B
	PQ-only: ciphertext to pkS	N/A	768 B
Response message	Ephemeral pk / ciphertext	32 B	768 B
	PQ-only: Client identity ciphertext	N/A	768 B

Careful selection of primitives

		X25519	ML-KEM	PQWG [SP:HNSWZ22]
Initial message	Ephemeral pk	32 B	800 B	800 B (MLKEM)
	Client id pkC	32 B	800 B	32B (H(pkC))
	PQ-only: ciphertext to pkS	N/A	768 B	96 B (McEliece)
Response message	Ephemeral pk / ciphertext	32 B	768 B	768 B (ML-KEM)
	PQ-only: Client identity ciphertext	N/A	768 B	96 B (McEliece)

Classic McEliece

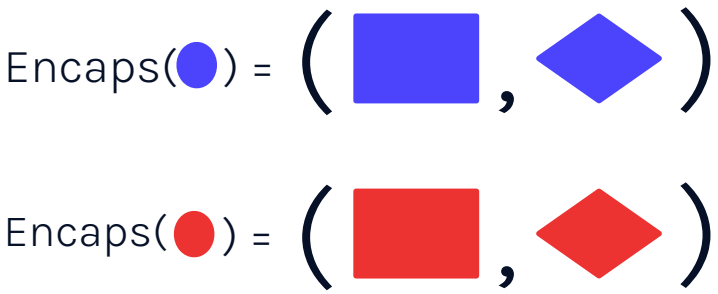
- WG Identity public keys are never transmitted over the wire
- So a public key of 260 kB isn't any problem, *right*?

Implementation concerns again

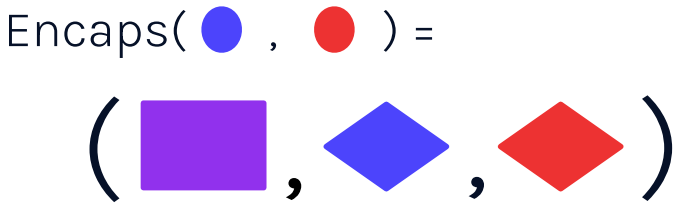
- Servers usually have many clients
- 260kB public keys x many clients = much memory used
- WireGuard implemented in the operating system **kernel** for performance reasons
  - Memory allocation is more restrictive
- No chance for embedded devices to deal with these public keys

Compressing KEMs

What are ML-KEM ciphertexts anyway



But can we also do this?



Related ideas: multi-recipient KEM (mKEM), Katana

Reinforced KEM

- Concept: 2-recipient-2-message KEM
- Encapsulate to two public keys
- Need **both** secret keys to recover the KEM shared secret

Operation	Input	Output
Keygen		pk sk
Reinforcing Keygen	pk	rpk rsk
REncaps	pk, rpk	ct K
RDecaps	sk, rsk, ct	K

We have code

TABLE 2. PERFORMANCE AND SIZES OF ML-KEM VERSUS RKEM. BENCHMARKS WERE OBTAINED ON AN INTEL CORE-I7 12700H CPU.

ML-KEM-512			RKEM		
Operation	Cycles	Size	Operation	Cycles	Size
KEM.KeyGen	60 364	800 B	KeyGen	239 557	1376 B
			RKeyGen	239 717	864 B
KEM.Encaps	68 330	768 B	REnc	658 294	1088 B
KEM.Decaps	87 900		RDec	708 649	

Size matters

		X25519	ML-KEM	PQWG [SP:HNSWZ22]	PQWG-RKEM
Initial message	Ephemeral pk	32 B	800 B	800 B (MLKEM)	864 B (RKEM rpk)
	Client id pkC	32 B	800 B	32B (H(pkC))	32 B (H(pkC))
	PQ-only: ciphertext to pkS	N/A	768 B	96 B (McEliece)	96 B (McEliece)
Response message	Ephemeral pk / ciphertext	32 B	768 B	768 B (ML-KEM)	1088 B (RKEM)
	PQ-only: Client identity ciphertext	N/A	768 B	96 B (McEliece)	N/A (RKEM)

Reduces server's client identity storage requirement by 190x

Applicability of RKEM

- Protocol authenticates **initiator** via KEM key exchange, and
- Responder knows initiator's static public key **early** in protocol, and
  - Not suitable for protocols that encrypt initiator public key with ephemeral KEX
- Protocol does ephemeral key exchange

WireGuard is a nice **demonstration** of this. Other examples:

- KEMTLS-PDK's client authentication
- Noise Protocols IK (=WG), IN, KK, KN, …

- Nice additional argument in favour of switching from signatures to KEMs for authentication (even more size reduction, KEMs were already smaller, KEMs were already faster, …)

Revisiting Post-Quantum WireGuard

Hashimoto, Katsumata, Niot, Wiggers. Revisiting PQ WireGuard: A Comprehensive Security Analysis With a New Design Using Reinforced KEMs. IEEE S&P 2026

- Improved design of PQ WireGuard (based in PQ Noise + KEM binding attack fixes)
- Proof of security
- Reinforced KEM allows avoiding overloading servers with excessively large client public keys, reducing public key memory by 190 to 390×
- Servers are still authenticated by Classic McEliece
- Still needs PSKs to authenticate the first message…

Conclusions

The lessons from

- **PQ TLS:** Migrating some protocols is a “drop-in” of primitives, but be aware of secondary effects such as bandwidth costs
- **PQ TLS:** Re-considering the status quo can lead to significant (even non-PQ) performance benefits
- **PQ Signal:** Diffie—Hellman was very nice, but non-interactive uses are hard to lift to PQ
- **PQ Signal:** achieving deniability requires non-traditional primitives
- **PQ WireGuard:** Underlying technical restrictions may make a post-quantum version difficult to prohibitive
- **Authentication is difficult, we need to consider it today**

Thanks for your attention

PQShield NIST Signatures Zoo

https://pqshield.github.io/nist-sigs-zoo/

Latest | Round 2 | Round 1 | History | Contribute ↗

Post-Quantum Signature Schemes

Comparing NIST on-ramp candidates and standardized schemes. Click column headers to sort. Use the filters to narrow down by category, security level, or size const…

Data reflects the latest known specifications for each scheme, last updated 2025-12-20. Consult the individual scheme websites for the most current information.

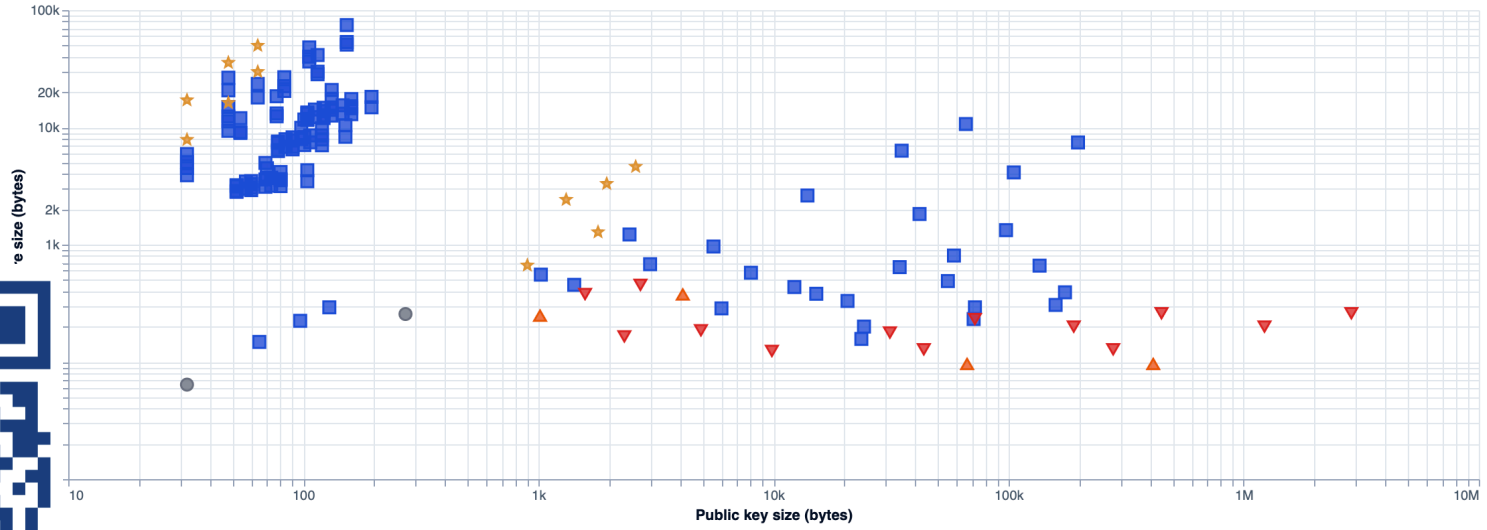
✨ new! ✨

Filters

SCHEMES All None

- Code-based
  - CROSS
  - LESS
- Isogenies
  - SQIsign
- Lattices
  - Falcon

pk size vs. sig size (log–log scale)



zoom · drag to pan

Reset zoom