Post-Quantum Ratcheting for Signal

Benedikt Auerbach
Yevgeniy Dodis
Shuichi Katsumata
Daniel Jost
Thomas Prest
Rolfe Schmidt

NIST PQC 2025

Plan: Solving a Series of Problems

1. We need post-quantum PCS
   a. Solution: use a KEM public ratchet
2. It uses too much bandwidth
   a. Partial solution: use optimized “Ratcheting KEMs”
3. It still uses too much bandwidth
   a. Solution: break messages into chunks with erasure codes
4. The keys come out too slowly
   a. Solution: Opportunistic sending
5. There are a bunch of opportunistic protocols, how do we compare?
   a. Solution: compare number of messages leaked
6. The best protocols don’t use standardized ML-KEM
   a. Solution: “incremental interface” for ML-KEM gets us close

Then we’ll talk about the present and future of our deployed protocol.

Non-plan

We won’t talk about

- How we hybridize post-quantum crypto with the classic Double Ratchet
- Changes needed for symmetric ratchets paired with our post-quantum ratchet

Both are interesting!

Our experience with the standardized primitives is more important.

The Signal Protocol: A “Refresher”

- **PQXDH** - initial handshake provides
    - Authentication (ECDH)
    - Forward secrecy (ECDH)
    - Deniability*
    - HNDL secrecy (MLWE)
- **Double Ratchet** - continuous key agreement provides
    - Post Compromise Security (ECDH)
- **Triple Ratchet** - *this talk*
    - Post Compromise Security (MLWE)
    - Rolling into production now

Problem 1/6: we need post-quantum PCS

We need Post Compromise security (PCS): heal after leaking state by adding fresh entropy.

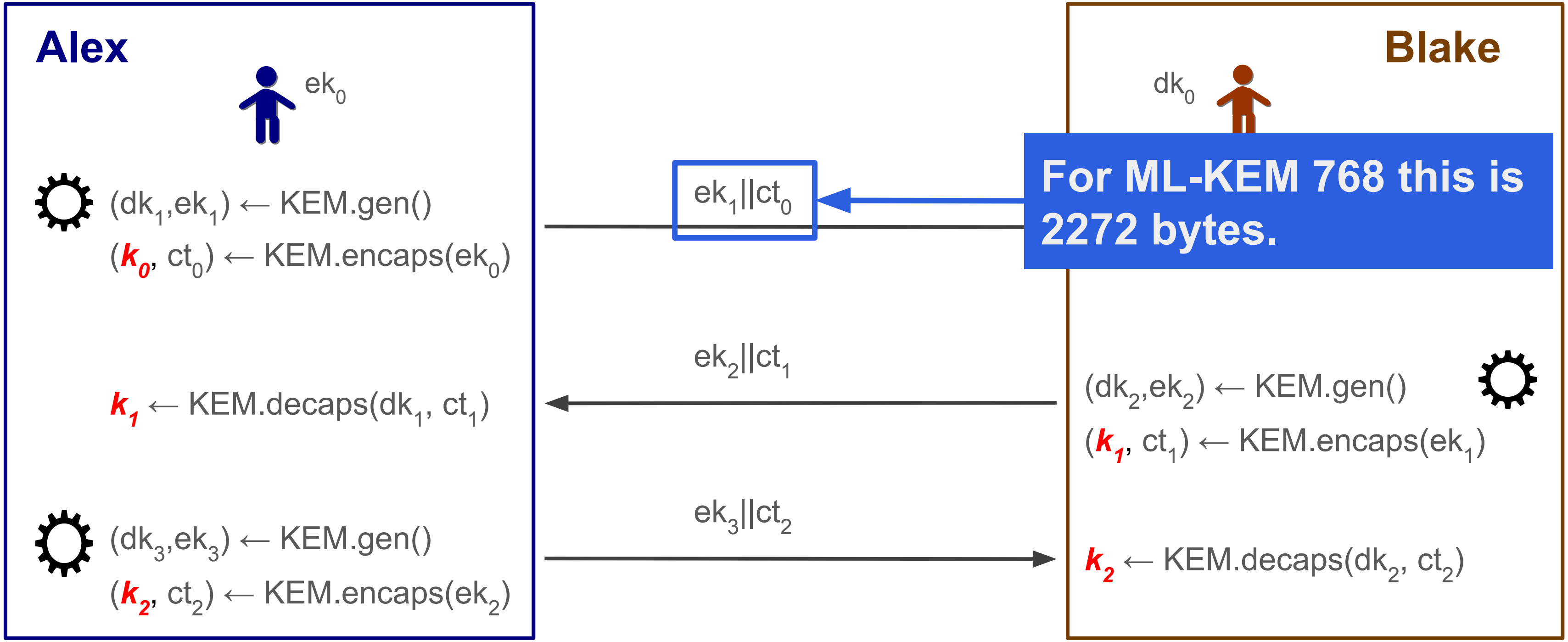
A Post Quantum Ratchet from **CKA**

We can build **Continuous Key Agreement (CKA)** with a PQ KEM
[EC:ACD19]

Alex		**Blake**
ek_0		dk_0
$(dk_1, ek_1) \leftarrow \text{KEM.gen}()$ $(\boldsymbol{k_0}, ct_0) \leftarrow \text{KEM.encaps}(ek_0)$	$ek_1 \| ct_0$ →	$\boldsymbol{k_0} \leftarrow \text{KEM.decaps}(dk_0, ct_0)$
$\boldsymbol{k_1} \leftarrow \text{KEM.decaps}(dk_1, ct_1)$	← $ek_2 \| ct_1$	$(dk_2, ek_2) \leftarrow \text{KEM.gen}()$ $(\boldsymbol{k_1}, ct_1) \leftarrow \text{KEM.encaps}(ek_1)$
$(dk_3, ek_3) \leftarrow \text{KEM.gen}()$ $(\boldsymbol{k_2}, ct_2) \leftarrow \text{KEM.encaps}(ek_2)$	$ek_3 \| ct_2$ →	$\boldsymbol{k_2} \leftarrow \text{KEM.decaps}(dk_2, ct_2)$

A Post Quantum Ratchet from **CKA**

We can build **Continuous Key Agreement (CKA)** with a PQ KEM
[EC:ACD19]



35x

Using ML-KEM 768 like this would increase the size of a typical small message by a factor of 35.

This costs us and our users.

This affects usability for users with poor connections.

Problem 2/6: this uses too much bandwidth

Idea: Reuse part of a KEM ciphertext as an encapsulation key for the next round.

A Common Lattice Based KEM

Encapsulation Key

- **s**: small, secret
- **x**: small, secret
- **ek** = **As** + **x**

Ciphertext to encrypt a bit K

- **s’**: small, secret
- **z’**: small, secret
- z”: small secret ring elt
- **ct1** = $\mathbf{A}^T\mathbf{s'}$ + **z’**
- **ct2** = $\mathbf{ek}^T\mathbf{s'}$ + z” + K⌊q/2⌉
- **ct** = (**ct1**, **ct2**)

ct1 is large and is just (the transpose of) an **ek**!

ct2 depends on ek but is small!

Let’s reuse **ct1** as an **ek** for the next round.

We don’t have to send **ek**!

This is an example of what we call a **Ratcheting KEM (RKEM)**.

⚠️⚠️⚠️

The new computations leak info about the secrets.

⚠️⚠️⚠️

Must assume
Hint-MLWE and select
appropriate
parameters.

⚔️ Katana: A Concrete RKEM [EC:DJK+25]

	Target Security	|EK|	|CT|	|EK|+|CT|
ML-KEM	128	800	**768**	**1568**
⚔️ Katana		832	**48**	**880**
ML-KEM	192	1184	**1088**	**2272**
⚔️ Katana		1344	**72**	**1416**

https://ia.cr/2025/078

Problem 3/6: this *still* uses too much bandwidth

Solution: Chunk long messages with **erasure codes** to keep overhead at ~40B per message.

✅ Tunable overhead

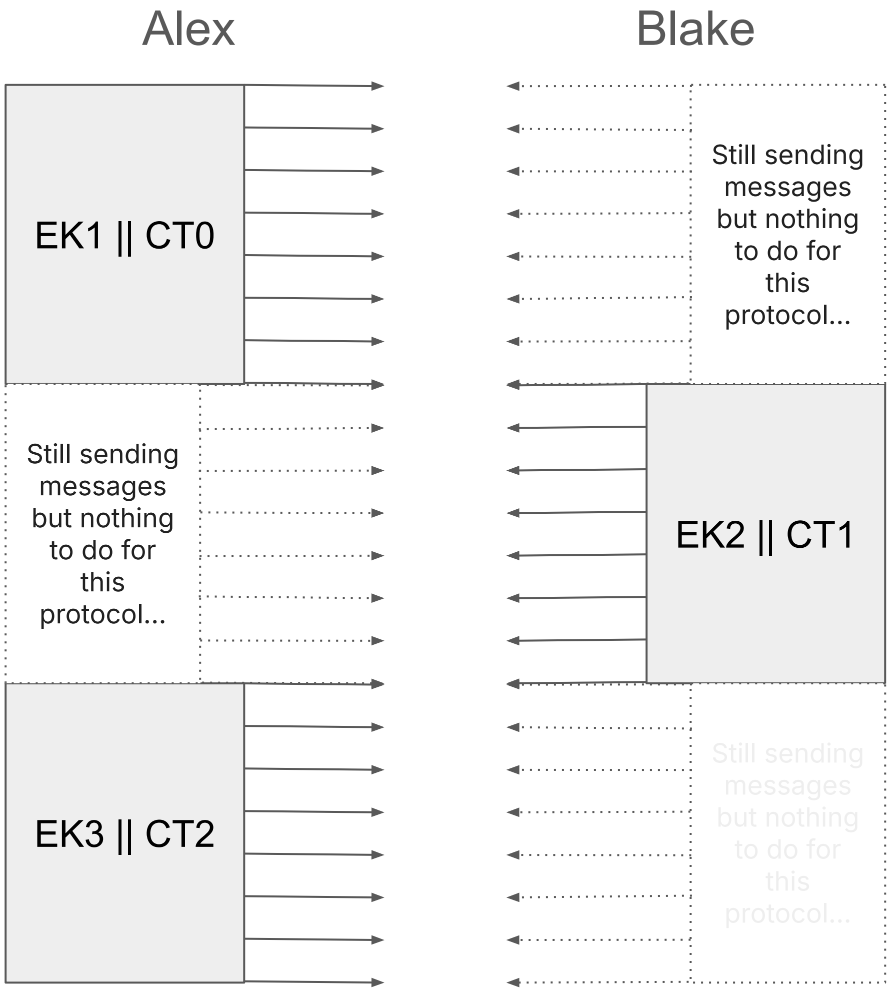
✅ Robust to adversarially dropped messages.

Secure Messaging with Sparse CKA

Now we can take any CKA and turn it into a “chunked” protocol.

Note: It isn’t a CKA anymore syntactically because it doesn’t emit a new key every time it sends or receives a message.

So we define a “Sparse CKA” (SCKA) and show how to construct secure Messaging from a Sparse CKA.

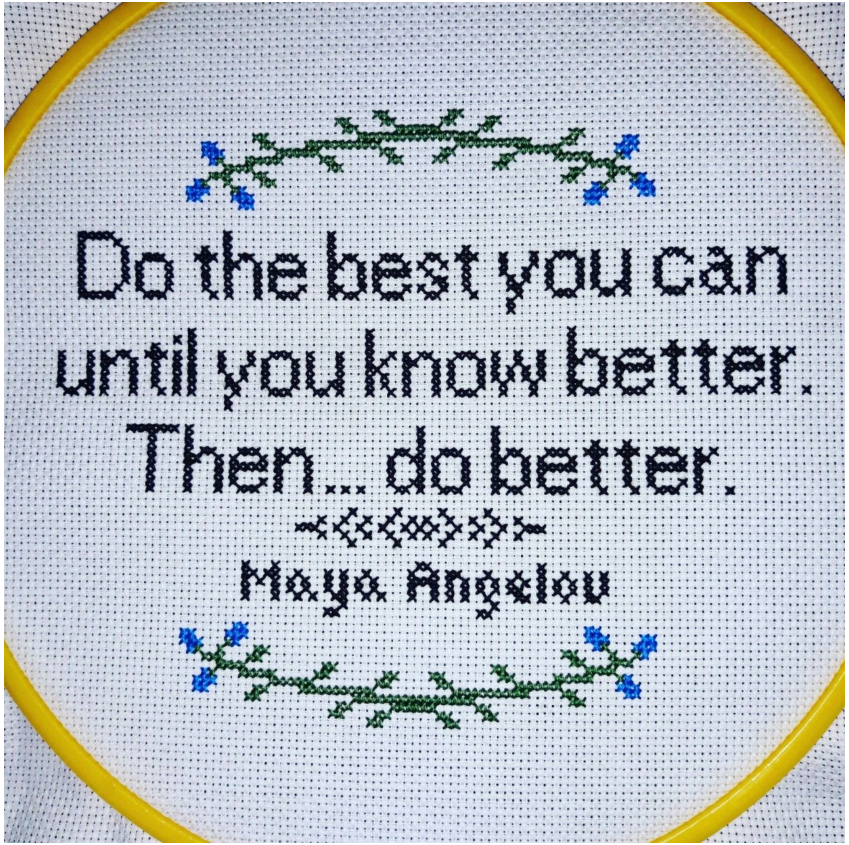


We could stop here.

- Use Katana to intantiate a CKA.
- Use our “chunking compiler” to turn it into an SCKA.
- Drop this into our SCKA-based Secure Messaging protocol to get messaging with MLWE-based security.

We could stop here.

- Use Katana to intantiate a CKA.
- Use our “chunking compiler” to turn it into an SCKA.
- Drop this into our SCKA-based Secure Messaging protocol to get messaging with MLWE-based security.
- **Or we could do better.**



Problem 4/6: healing is slow

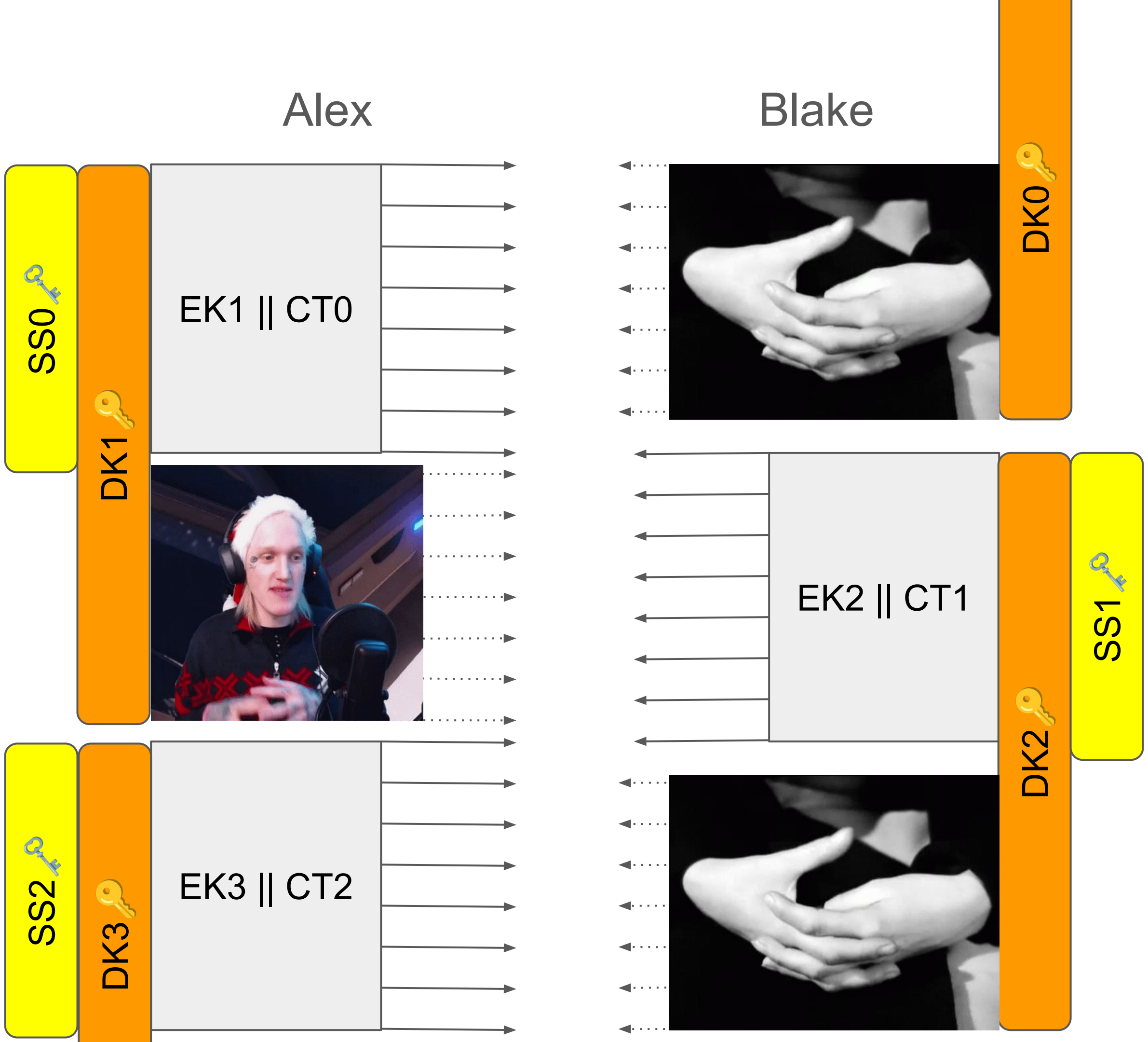
The Problems

When we “chunk” the Standard KEM CKA protocol, there is always someone sitting quiet.

And look how long Alex and Blake have to hold onto their secrets. 😬.

Big attack surface, slow key emission.

Can’t they do *something*?



KEM Shared Secret 🗝️

Decapsulation Key 🔑

Send "Opportunistically":
If you are blocked, sample something and send!

Bad Idea: Whenever you are blocked, sample a new keypair and start sending EK.

What happens if Alex sends more messages than Blake?

They'll accumulate an unbounded number of secrets, make healing SLOW!

So it takes some care.

6 Good Protocols [USENIX:ADJ+25]

RKEM Use Ratcheting KEM Send EK||CT back and forth in ping-pong fashion	**UniKEM** Use IND-CPA KEM A sends EK B sends CT repeat	**BiKEM** Use IND-CPA KEM A sends EK B sends CT B sends EK A sends CT repeat
Opp-RKEM Sample and send new EK while receiving EK || CT	**Opp-UniKEM** Sample and send “CT1” part of ciphertext while receiving EK	**Opp-BiKEM** Send independent EK’||CT’ while receiving EK||CT

https://www.usenix.org/system/files/usenixsecurity25-auerbach.pdf

Which one?

They all look reasonable.

They all have the same "Δ_{PCS}".

Are we *sure* the opportunistic ones are better? Remember oversampling can hurt PCS!

Problem 5/6: we don’t know how to compare protocols

Signal’s goal is to protect users’ messages.

So the metric that matters is the **number of user messages leaked by a compromise**.

Measuring the Vulnerable Message Set

We want to minimize the size of the **Vulnerable Message Set**.

This is **not** intrinsic to the protocol!

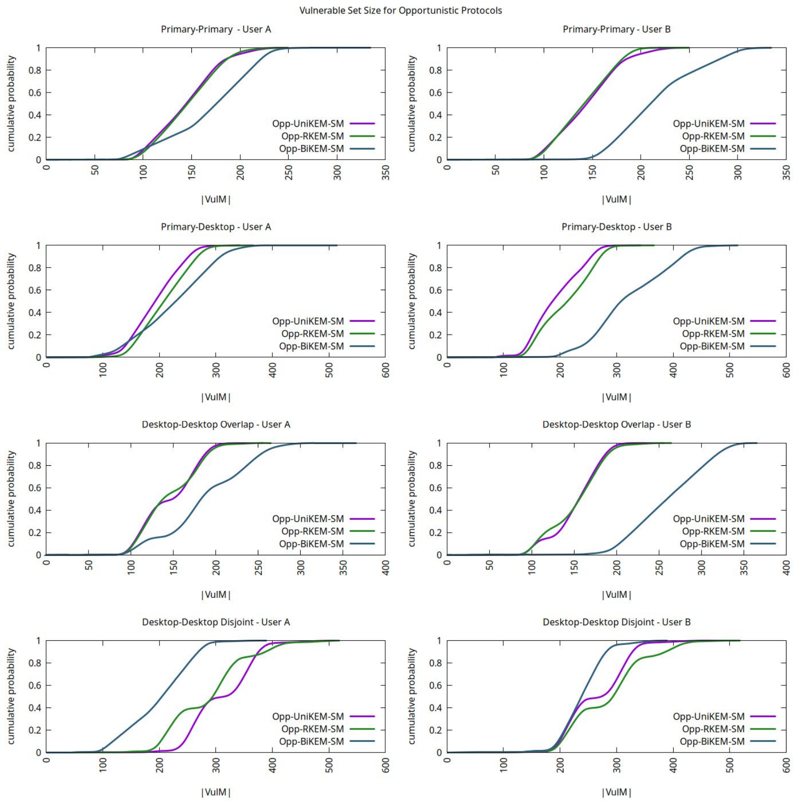
- If I send you messages all day and you don't turn your laptop on, we will never do a key exchange and heal.

So we need probabilistic models of messaging behavior:

- Phones that are usually online
- Laptops that are offline regularly
- Time zones
- Communication balance

Opp-RKEM and Opp-UniKEM stand out

- For two online devices Opp-RKEM slightly edges Opp-UniKEM and both clearly beat Opp-BiKEM
- Opp-RKEM and Opp-UniKEM swap places when a phone is talking to a desktop
- Two desktops in opposite time zones? Now Opp-BiKEM is the winner



But we can construct message sending patterns where any “good protocol” wins.

Problem 6/6: the best protocols don’t use standard ML-KEM

We want to use standardized ML-KEM:

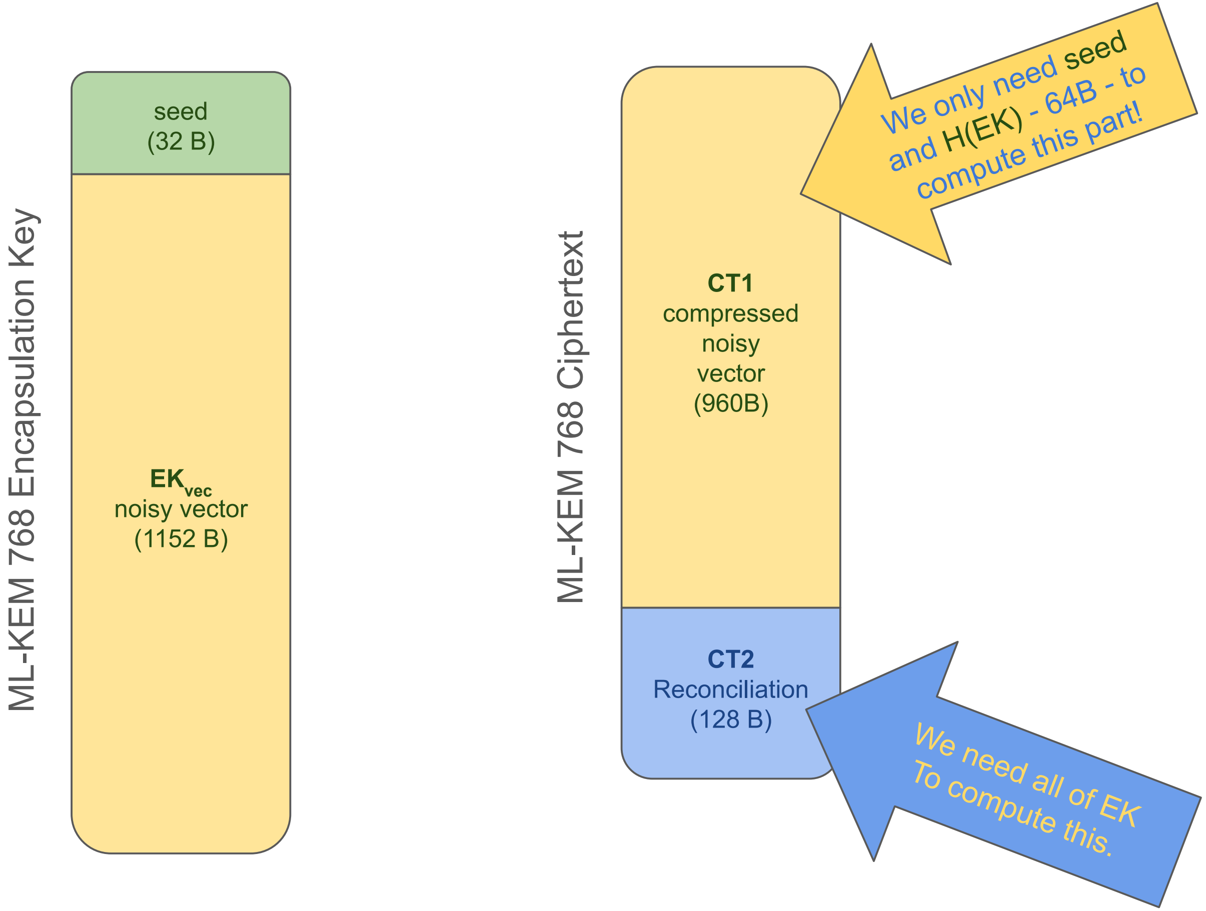
- Extensive scrutiny
- Library support

Opp-UniKEM used an **IND-CPA** KEM.

Opp-RKEM used an **RKEM** (🤯).

The ML-KEM FO transform gets in the way, but we work around it.

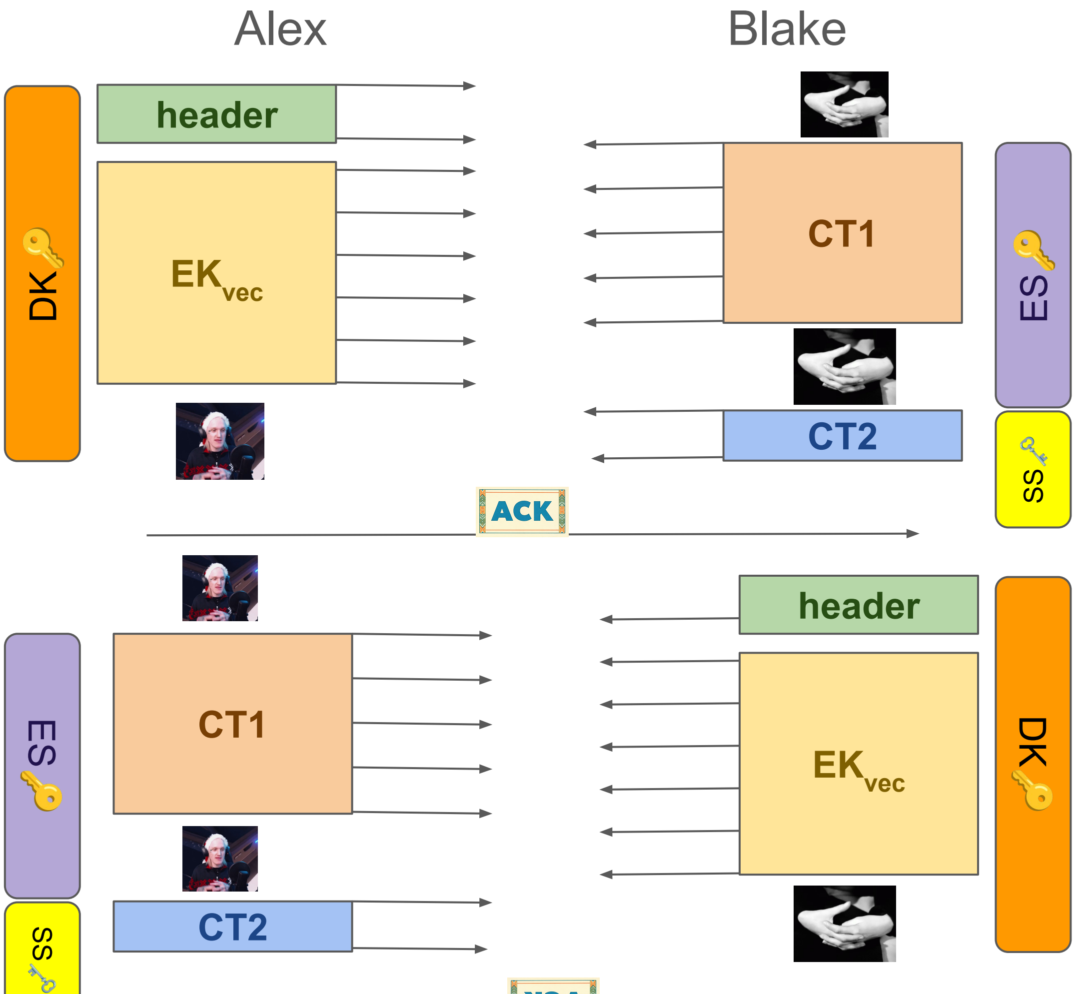
Open the KEM Black Box: Incremental KEM



Main Idea: Send “header” with **seed** and **H(EK)** first so Blake can sample **CT1** early.

We can do most of the work in parallel!

- Alex sends the **seed** first
- When Blake gets **seed**, they sample **CT1** and start sending it.
- When Alex gets a chunk of **CT1** they can stop sending seed and start sending **EK$_{vec}$**.
- Once Blake has all of **EK$_{vec}$** (and knows Alex has CT1!) they can start sending **CT2**.
- When Alex gets **CT2** they can start using the shared secret and ACK Blake.
- When Blake gets the ACK, they start using the shared secret and swap roles.



The Deployed Protocol: Present and Future

This is being deployed now.

Implementation and more details at https://github.com/signalapp/SparsePostQuantumRatchet

Next Steps: Ratcheting

Use IND-CPA ML-KEM:

- Need library support and IP clarity

Implement and use Katana:

- Expect to implement this ourselves
- Want more public scrutiny

Next steps: A Fully Post-Quantum Signal Protocol

We still need a fully post-quantum handshake.

There has been some great research in this direction and we are working on the detailed design.

It seems clear we will want Falcon with:

- Message Recovery
- Key Recovery
- Ring signatures

Stay tuned!

Thank you!

rolfe.1729

Scan this QR code with your phone to chat with me on Signal.

rolfe@signal.org