Post-quantum Security at Signal

Rolfe Schmidt
Signal Messenger
Includes work with K. Bhargavan, C. Jacomme, and F. Kiefer

- Signal and its PQ needs
- Boring PQ cryptography is interesting (and we are making practical progress)
- Fancy PQ cryptography is interesting (and we have a lot of questions)
- Conclude

- **Signal and its PQ needs**
- Boring PQ cryptography is interesting (and we are making practical progress)
- Fancy PQ cryptography is interesting (and we have a lot of questions)
- Conclude

Signal is a global scale private communications app.

Signal Messenger

Our core features are E2EE

- 1:1 messaging
- Group messaging
- Calling

But we need many supporting features:

- Contact discovery and management
- Identity (profiles, usernames, ...)
- Account backup
- Spam prevention
- Stickers, reactions, group links, badges, and much more…

Signal's threat model

Signal protects

- Data (messages, attachments, call contents, …)
- Metadata (group membership, social graph, …)

From everyone who shouldn't have it:

- Third parties
- Cloud providers
- Our own administrators

This means we have interesting challenges and use a range of cryptographic tools.

Let's use the "boring/fancy"* dichotomy to look at them.

* More on this in Bas' talk!

Some tools we use

"Boring"

- Secure Messaging (of course!)
- Noise Protocol
- Sealed sender
- HPKE
- Signatures/AEAD/MACs/…

"Fancy"

- Anonymous credentials
- Verifiable encryption
- PPSS/OPRFs*
- VRFs(?)
- Partially blind signatures

* OSDI'24 https://eprint.iacr.org/2024/887

We have a lot of work to do to move to post quantum crypto.

Signal's post-quantum priorities

- **Immediate** concern about **Harvest Now Decrypt Later (HNDL)** attacks on **data** and **metadata** throughout our systems.
- Want to **understand** **fully post-quantum** secure alternatives for all parts of our clients and infrastructure.
- Will move to **hybrid** secure systems early if the costs are reasonable.
- Want to keep **DH-based security** as long as we feel it has value (and we expect that to be a while).

- Signal and its PQ needs
- **Boring PQ cryptography is interesting (and we are making practical progress)**
- Fancy cryptography is interesting (and we have a lot of questions)
- Conclude

Boring PQ Cryptography is Interesting

And we're making progress

- PQXDH: our first step
- Formal verification of PQXDH
- KEM security properties: what we learned

Data security is rooted in 1:1 messaging. This is where we need to start.

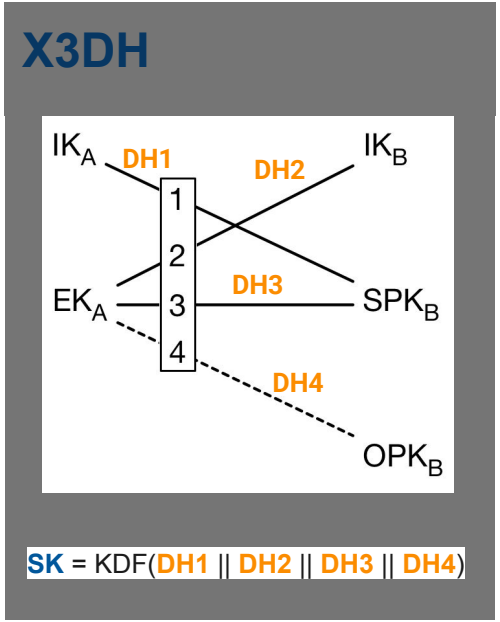
The Signal Protocol (Marlinspike & Perrin)

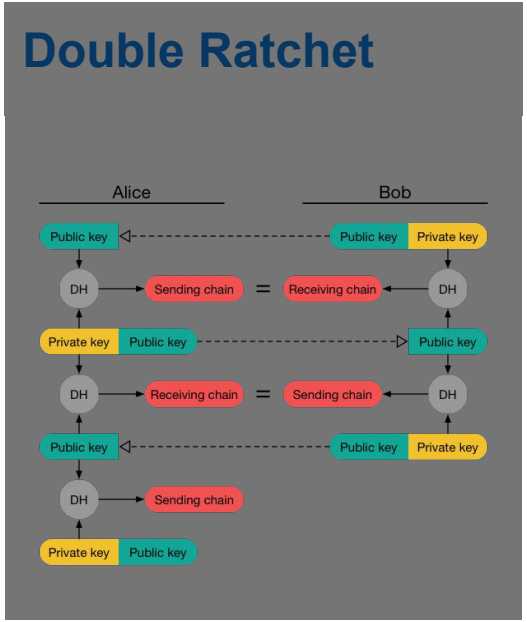
Two parts:

- X3DH handshake
- Double Ratchet for continuous key agreement

Important security guarantees:

- Mutual authentication
- Post-compromise security
- Forward secrecy
- Deniability





The Signal Protocol (Marlinspike & Perrin)

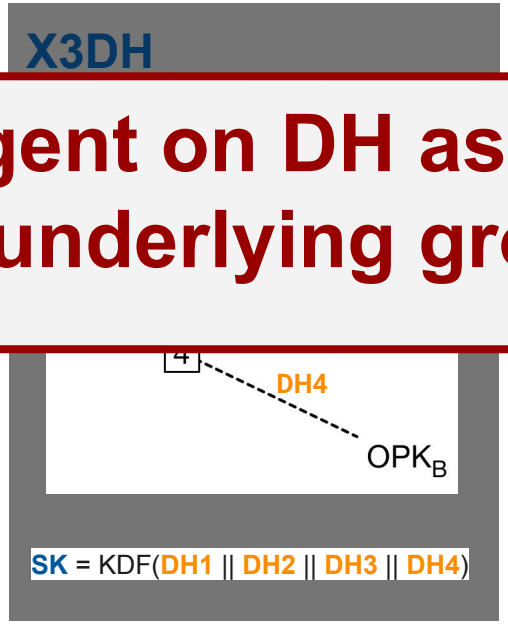
Two parts:

- X3DH handsh
- Double Ratch
  continuous k

Important security

- Mutual authentication
- Post-compromise security
- Forward secrecy
- Deniability

Contingent on DH assumptions for the underlying group!



PQXDH was a small update to X3DH to provide HNDL protection. It was deployed in 2023.

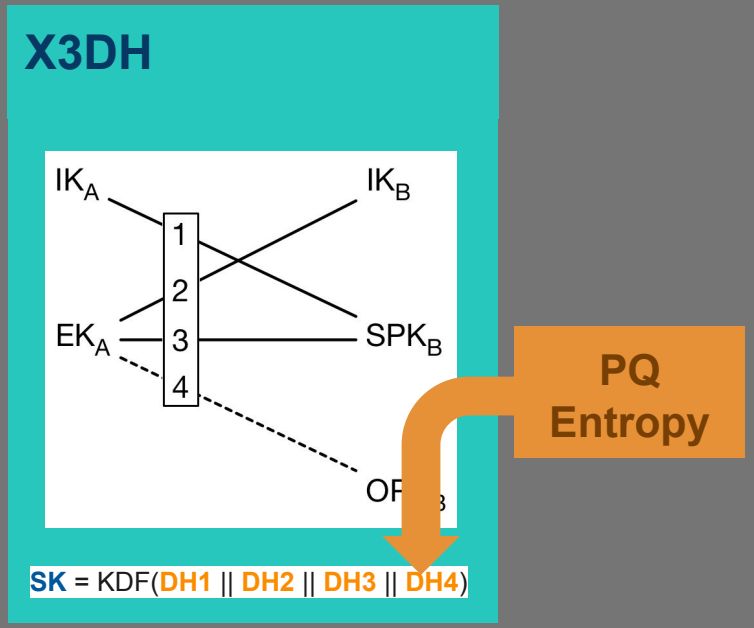
PQXDH Protocol Requirements

- Provide HNDL protection against future DL solvers
- No loss of current DH-based security guarantees

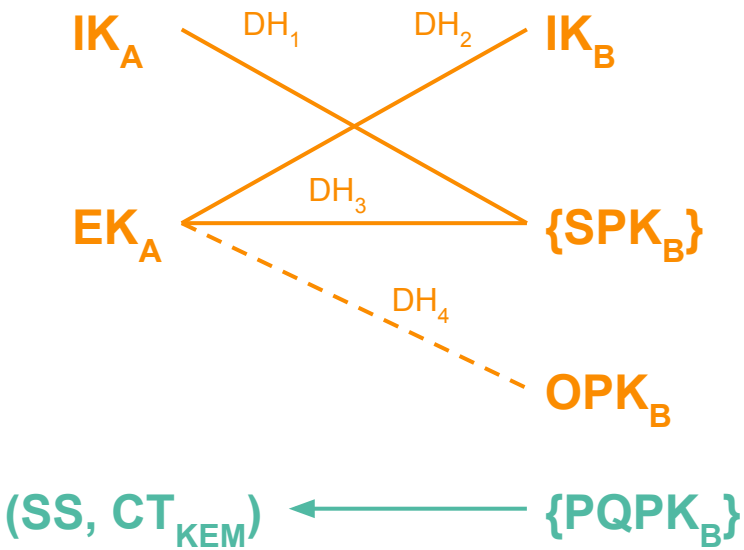
Non-goal: Protect against active quantum attackers

It *is* boring!

Take X3DH and add in a PQ-KEM encapsulated shared secret.



PQXDH: Key Agreement Core



$$SK = KDF(DH_1 \| DH_2 \| DH_3 \| DH_4 \| SS)$$

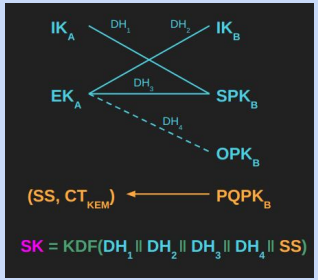
PQXDH Handshake (Synchronous Simplification)

Alex

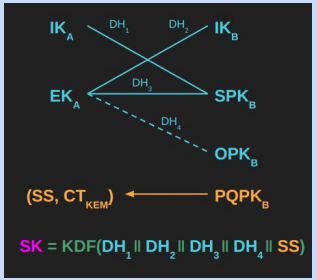
Blake

IK_B^{PK}, SPK_B^{PK}, $Sig(SPK_B^{PK})$, OPK_B, $PQPK_B^{PK}$, $Sig(PQPK_B^{PK})$

“PreKey Bundle” ←
(IK_B^{PK}, SPK_B^{PK}, $Sig(SPK_B^{PK})$,
OPK_B, $PQPK_B^{PK}$, $Sig(PQPK_B^{PK})$



CT, IK_A^{PK} EK_A^{PK}, SPK_B.id, OPK_B.id, CT_{KEM}, $PQPK_B$.id



AD ←IK_A^{PK} ‖ IK_B^{PK}

CT ←AEAD.Enc(SK, “hello”, AD)

SK

SK ←KDF(DH_1‖DH_2‖DH_3‖DH_4‖SS)
AD ←IK_A^{PK} ‖ IK_B^{PK}

PT ←AEAD.Dec(SK, CT, AD)

SK

This *looks like* it achieves our goals...

…but it still isn’t a protocol.

What properties do AEAD, KEM, DH have?

How are messages formatted? How are keys encoded?

What exactly do you sign and how?

We need specifics and proof.

This is where it gets interesting.

Formal Verification of PQXDH

With K. Bhargavan, C. Jacomme, and F. Kiefer (USENIX24)

- ProVerif, CryptoVerif, and what we model
- Writing security goal queries
- Combining CryptoVerif and ProVerif to find attacks

Formal Verification of PQXDH

ProVerif

- Symbolic Model (Dolev-Yao)
- Perfect primitives by default
- Fully automated
- Either gives an attack or a security claim

CryptoVerif

- Computational Model
- Gives guarantees similar to pen-and-paper proofs
- Supports (and often needs) manual proof guidance.
- Post-quantum sound!

A ProVerif Process

```
let Initiator(i:client, IKA_s:scalar) =
   …
   (* receive from the server the pre key bundles *)
   in(server, (SPKB_p:point,SPKB_sig:bitstring,PQPKB_p:kempub,PQPKB_sig:bitstring));
   (* Verify the signatures *)
   if verify(IKB_p,encodeEC(SPKB_p),SPKB_sig) then
   if verify(IKB_p,encodeKEM(PQPKB_p),PQPKB_sig) then
   (
    let (CT:bitstring,SS:bitstring) = pqkem_enc(PQPKB_p) in

    new EKA_s:scalar;

    let EKA_p = s2p(EKA_s) in
    let DH1 = dh(IKA_s,SPKB_p) in
    let DH2 = dh(EKA_s,IKB_p) in
    let DH3 = dh(EKA_s,SPKB_p) in
    let SK = kdf(concat5(DH1,DH2,DH3,DH4,SS)) in

    event InitDone(i,r,true,OPKB_p,SPKB_p,PQPKB_p,SK);

       let ad = concatIK(IKA_p,IKB_p) in
       new msg_nonce: bitstring;
       let msg = app_message(i,r,msg_nonce) in
       let enc_msg = aead_enc(SK,empty_nonce,msg,ad) in
       (* Send Message *)
       out(server, (IKA_p,EKA_p,CT,  OPKB_p, SPKB_p, PQPKB_p, enc_msg))
   )
 ).
```

What We Model

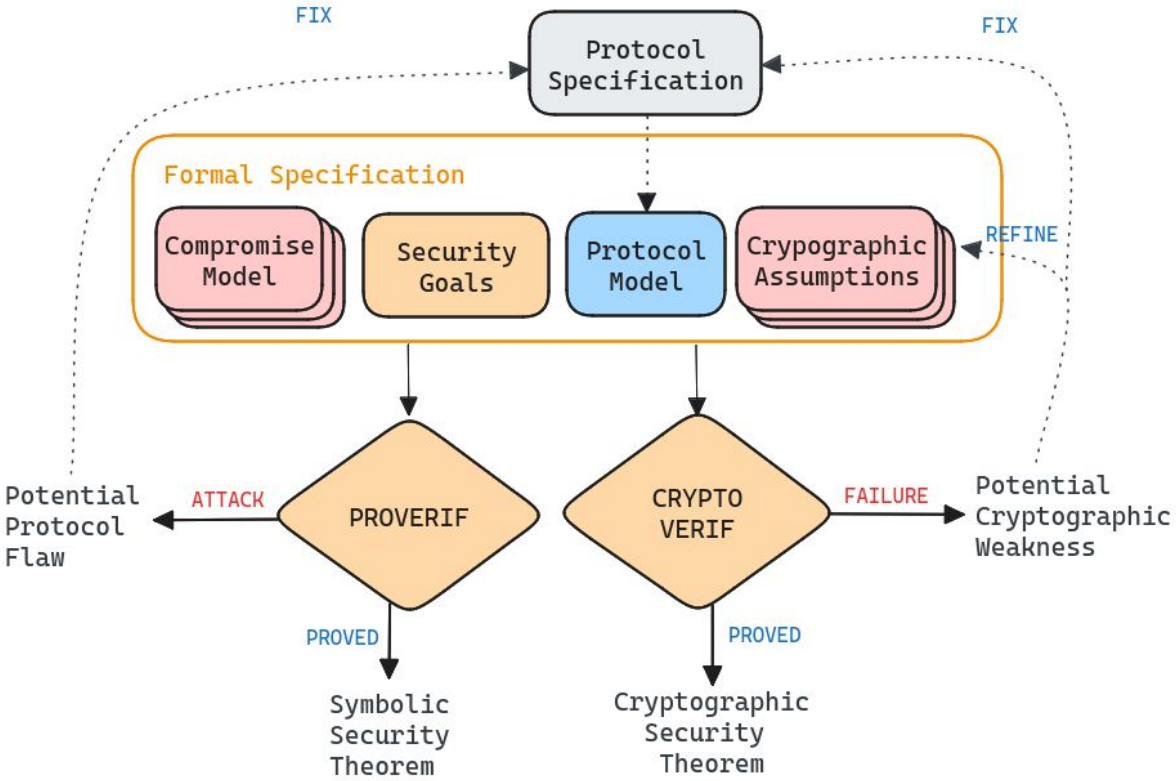
Protocol

- Arbitrary number of communicating agents
- Trusted PKI
- Untrusted Key Distribution Server
- Connection = 1 encrypted message from Alex to Blake (no follow-up messages)

Threat Model

- Identity keys can be compromised at any time
- OPK, EK, and PQPK can be compromised for certain security goals
- Quantum adversary has explicit power to break DH primitives
- KEM security could break down at any time
- Model timestamps on attacks to reason about security before and after quantum attack

Formal Verification Methodology



Writing Queries for Security Goals

Building a Confidentiality Query in ProVerif

If the protocol is done and the attacker knows the key, what could have happened?

We want a comprehensive list of events so that if any one is missing we can find an attack.

```
event(RespondDone(b,a,spk,pqpk,sk))@i &&
attacker(sk) ⇒ ???
```

Building a Confidentiality Query in ProVerif

Inspecting the protocol we see that this can happen if Alex's IK is compromised, if Blake's SPK is compromised or if DH is broken:

```
event(RespondDone(b,a,spk,pqpk,sk))@i &&
attacker(sk) ⇒
    event(CompromiseIK(a))
    || event(BrokenDH())
    || event(CompromiseSPK(b, spk)
```

This gives us a proof, but it's not very interesting.

It tells us nothing
about forward secrecy.

So we try again.

Building a Confidentiality Query in ProVerif

The attacker could have compromised Alex before time i:

```
event(RespondDone(b,a,spk,pqpk,sk))@i &&
attacker(sk) ⇒
    (event(CompromiseIK(a))@j1 && j1 < i)
```

Building a Confidentiality Query in ProVerif

Or if Alex is not corrupted, DH could be broken before i

```
event(RespondDone(b,a,spk,pqpk,sk))@i &&
attacker(sk) ⇒
    (event(CompromiseIK(a))@j1 && j1 < i)
    || (event(BrokenDH())@j1 && j1 < i)
```

Building a Confidentiality Query in ProVerif

Or if Alex is not corrupted and DH is not broken, Blakes SPK could be corrupted before i:

```
event(RespondDone(b,a,spk,pqpk,sk))@i &&
attacker(sk) ⇒
    (event(CompromiseIK(a))@j1 && j1 < i)
    || (event(BrokenDH())@j1 && j1 < i)
    || (event(CompromiseSPK(b, spk)@j1 && j1 < i)
```

Building a Confidentiality Query in ProVerif

But what if DH was broken *after* time i?

If the attacker could break the KEM or impersonate Blake they get the secret:

```
(event(BrokenDH())@j1 && (
    j1 < i
    || event(BrokenKEM)
    || event(CompromisePQPK(b,pqpk))
    || (event(CompromisePQPK(b,pqpk2)@j2 && j2 < i)
    || (event(CompromiseIK(b))@j2 && j2 < i)))
```

Building a Confidentiality Query in ProVerif

And what if Blakes SPK was compromised *after* time i?

If the attacker could compromise Blake before i or break the KEM or impersonate Blake they get the secret:

```
(event(CompromiseSPK(b,spk))@j1 &&
    (j1 < i
        || (event(CompromiseIK(b))@j2 &&
            (j2 < i
                || event(CompromisePQPK(b,pqpk))
                || event(BrokenKEM)
                || (event(CompromisePQPK(b,pqpk2))@j3 &&
j3 <i)))))
```

Building a Confidentiality Query in ProVerif

Putting it together:

```
event(RespondDone(b,a,spk,pqpk,sk))@i && attacker(sk)   ⇒
    (event(CompromiseIK(a))@j1 && j1 < i)
    || (event(BrokenDH())@j1 && (j1 < i
        || event(BrokenKEM)
        || event(CompromisePQPK(b,pqpk))
        || (event(CompromisePQPK(b,pqpk2)@j2 && j2 < i)
        || (event(CompromiseIK(b))@j2 && j2 < i)))
    || (event(CompromiseSPK(b, spk)@j1 &&  && (j1 < i
        || (event(CompromiseIK(b))@j2 &&
            (j2 < i || event(CompromisePQPK(b,pqpk)) ||
event(BrokenKEM)
                || (event(CompromisePQPK(b,pqpk2))@j3 && j3 <i)))))
```

This gives us a security proof.

Remove any one of the disjunctions and you'll find an attack.

This gives us a security proof.

Remov
disjun
find an attack.

Try it yourself!
Find the models here:
https://github.com/Inria-Prosecco/pqxdh-analysis

Using CryptoVerif and ProVerif together to Find Attacks

Finding Attacks 1

- Initially we were unable to prove confidentiality using CryptoVerif
- But if we added domain separation to the signatures of the elliptic curve and KEM prekeys, we *could* prove our security theorems.

Was this a weakness in the protocol?

Or a weakness in our ability to prove things?

Finding Attacks 2

To find out, we go to ProVerif

- Give the attacker explicit ability to confuse keys:

```
fun ECasKEM(point):kempub [typeConverter].
equation forall x:scalar;
encodeKEM(ECasKEM(SMUL(x,G))) = encodeEC(SMUL(x,G)).
```

Finding Attacks 2

To find out, we go to ProVerif

- Give the attacker explicit ability to confuse keys:

```
fun ECasKEM(point):kempub [typeConverter].
equation forall x:scalar;
encodeKEM(ECasKEM(SMUL(x,G))) = encodeEC(SMUL(x,G)).
```

- And give the attacker the ability to recover shared secrets from weak ciphertexts:

```
reduc forall x:scalar,k:bitstring;
weakECasKEM(penc(ECasKEM(SMUL(x,G)),k)) = k.
```

Finding Attacks 3

Now when we run ProVerif we get an explicit attack trace

1. Using the function info_x25519_sha512_kyber1024 the attacker may obtain info_x25519_sha512_kyber1024.
attacker(info_x25519_sha512_kyber1024).

2. Using the function zeroes_sha512 the attacker may obtain zeroes_sha512.
attacker(zeroes_sha512).

…

19. By 18, the attacker may know (SMUL(IK_s_1,G),SMUL(EKA_s_1,G),penc(SMUL(SPKB_s_3,G),ss_1),SMUL(y,G),SMUL(SPKB_s_2,G),SMUL(SPKB_s_3,G),aead_enc(hkdf(concat(ff_x25519,concat5(SMUL(IK_s_1,SMUL(SPKB_s_2,G)),SMUL(IK_s_2,SMUL(EKA_s_1,G)),SMUL(EKA_s_1,SMUL(SPKB_s_2,G)),SMUL(EKA_s_1,SMUL(y,G)),ss_1)),zeroes_sha512,info_x25519_sha512_kyber1024),empty_nonce,app_message(a,b,msg_nonce_1),concatIK(SMUL(IK_s_1,G),SMUL(IK_s_2,G)))).
Using the function 3-proj-7-tuple the attacker may obtain penc(SMUL(SPKB_s_3,G),ss_1).
attacker(penc(SMUL(SPKB_s_3,G),ss_1)).

20. By 19, the attacker may know penc(SMUL(SPKB_s_3,G),ss_1).
Using the function weakECasKEM the attacker may obtain ss_1.
attacker(ss_1).

In practice we prefix keys with type indicators.

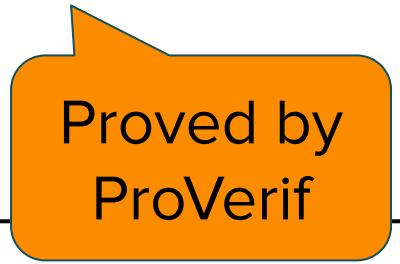
We added that to the spec.

After iterating we had a revised protocol and security proofs.

Symbolic Security Theorem

Theorem 1 PQXDH in the symbolic model provides:

- Peer-authentication
- Forward secrecy
- Resistance to key compromise impersonation
- Session independence
- Resistance to HNDL attacks in case of a DH breakdown
- Data agreement over the shared prekey



Classical Security

Theorem 2 (PQXDH classical computational security) If X25519 satisfies the gapDH assumption, the KDF is a Random Oracle, the AEAD is IND-CPA+INT-CTXT, and the signature scheme was unforgeable when some key exchange was completed, then secrecy of the derived key still holds in the future.

Proved by CryptoVerif

PQ Security

Theorem 3 (PQXDH post-quantum computational security) Under IND-CCA for the KEM, if the KDF is a PRF, the AEAD is IND-CPA+INT-CTXT, and the signature scheme was unforgeable when some key exchange was completed, then secrecy of the derived key still holds in the future.

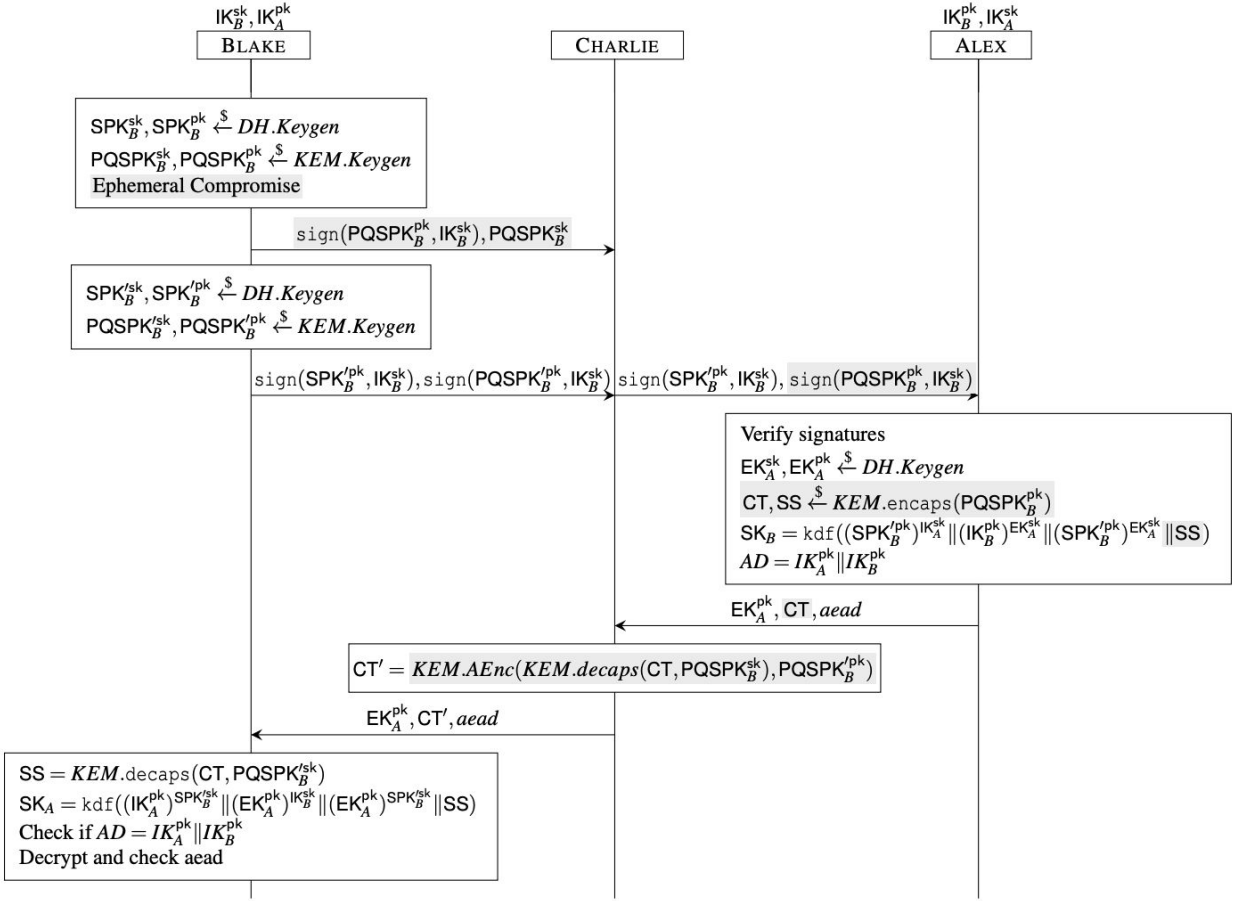
Proved by CryptoVerif with PQ Extension

KEM Security Properties

- KEM re-encapsulation attack on PQXDH v1
- KEM security properties

KEM Re-encapsulation Attack on PQXDH v1

KEM Re-encapsulation Attack



No session independence.
No agreement on the KEM public key.

A compromise of one PQPK breaks HNDL security for all other PQPKs of a party.

What security properties do we need in a KEM to prevent this?

KEM Security Properties

Semi-honest collision resistance (SH-CR)

We can state a security property that prevents this attack.

- An agent encapsulates with attacker chosen **pk'** to produce **(ss, ct).**
- Different agent decapsulates attacker chosen ciphertext **ct'** with their own honestly generated but compromised **sk**.
- Attacker wins if
  - The malicious **ct'**
  - decaspulates to same **ss**
  - under honestly generated but compromised **sk**.

$$\mathbf{Adv}^{SH\text{-}CR}_{\mathcal{A},KEM} = \Pr\left[decaps(ct', \mathsf{sk}) = ss \wedge (ct \neq ct' \vee \mathsf{pk} \neq \mathsf{pk}') \;\middle|\; \begin{array}{l} \mathsf{sk}, \mathsf{pk} \xleftarrow{\$} keygen() \\ \mathsf{pk}' \xleftarrow{\$} \mathcal{A}(\mathsf{sk}) \\ ss, ct \xleftarrow{\$} encaps(\mathsf{pk}') \\ ct' \xleftarrow{\$} \mathcal{A}(\mathsf{sk}, ss, ct) \end{array} \right]$$

Semi-honest collision resistance (SH-CR)

We can state a security property that prevents this attack.

- An agent encapsulates with attacker chosen **pk'** to produce **(ss, ct).**
- Different agent decapsulates attacker chosen ciphertext **ct'** with their own honestly generated but compromised **sk**.
- Attacker wins if
  - The malicious **ct'**
  - decaspulates to same **ss**
  - under honestly generated but compromised **sk**.

$$\mathrm{Adv}^{SH\text{-}CR}_{\mathcal{A},KEM} = \Pr\left(decaps(ct', \mathsf{sk}) = ss \wedge (ct \neq ct' \vee \mathsf{pk} \neq \mathsf{pk}') \;\middle|\; \begin{array}{l} \mathsf{sk}, \mathsf{pk} \xleftarrow{\$} keygen() \\ \mathsf{pk}' \xleftarrow{\$} \mathcal{A}(\mathsf{sk}) \\ ss, ct \xleftarrow{\$} encaps(\mathsf{pk}') \\ ct' \xleftarrow{\$} \mathcal{A}(\mathsf{sk}, ss, ct) \end{array} \right)$$

Theorem: Kyber is SH-CR

Theorem: PQXDH guarantees agreement over PQSPK if the KEM is SH-CR

Semi-honest collision resistance (SH-CR)

We can state a security property that prevents this attack.

- An agent encapsulates with attacker chosen **pk'** to produce **(ss, ct).**
- Different agent decapsulates attacker chosen ciphertext **ct'** with their own honestly generated but compromised **sk**.
- Attacker wins if
  - The malicious **ct'**
  - decaspulates to same **ss**
  - under honestly generated but compromised **sk**.

$$\mathrm{Adv}^{SH\text{-}CR}_{\mathcal{A},KEM} = \Pr\left(decaps(ct', \mathsf{sk}) = ss \wedge (ct \neq ct' \vee \mathsf{pk} \neq \mathsf{pk}') \;\middle|\; \begin{array}{l} \mathsf{sk}, \mathsf{pk} \xleftarrow{\$} keygen() \\ \mathsf{pk}' \xleftarrow{\$} \mathcal{A}(\mathsf{sk}) \\ ss, ct \xleftarrow{\$} encaps(\mathsf{pk}') \\ ct' \xleftarrow{\$} \mathcal{A}(\mathsf{sk}, ss, ct) \end{array} \right)$$

Theorem: Kyber is SH-CR

Theorem: PQXDH guarantees agreement over PQSPK if the KEM is SH-CR

We are not saying that this is the security property that KEMs should target!

KEM Security Properties

X-Wing [BCD+24] introduces CCR

- With an honestly generated, compromised key
- Attacker tries to
  - find two ciphertexts
  - That decapsulate to the same value

Theorem:

- CCR ⇏ SH-CR
- SH-CR ⇏ CCR

KEM Security Properties

X-Wing [BCD+24] introduces CCR

- With an honestly generated, compromised key
- Attacker tries to
  - find two ciphertexts
  - That decapsulate to the same value

Theorem:
- CCR ⇏ SH-CR
- SH-CR ⇏ CCR

Keeping Up with the KEMs [CDM23] introduces a rich set of security properties.

None directly match the semi honest setting of SH-CR.

Proposition: MAL-Bind-K,CT-PK ⇒ SH-CR

“KEMs that satisfy our key-binding properties will leave fewer pitfalls for protocol designers.” - CDM23

But in the end the responsibility lies with the protocol designer.

There is a lot more…

“Boring” Crypto Work

- Noise protocol for enclave-enclave and enclave-client communication
  - Noise HFS implementation in progress now
  - **Secure Value Recovery and Contact Discovery HNDL risk**
- Custom Hybrid PKE
  - Sealed Sender, linked devices, …
- Signatures
  - Less urgent

- Signal and its PQ needs
- Boring PQ cryptography is interesting (and we are making practical progress)
- **Fancy cryptography is interesting (and we have a lot of questions)**
- Conclude

“Fancy” Crypto Work: Handshake

PQXDH provides HNDL protection.

We will want a fully (hybrid) post quantum replacement.

This will likely involve new primitives

- Ring Signatures
- Designated Verifier Signatures
- Split KEMs
- …

“Fancy” Crypto Work: Private Groups

The Signal Private Group system uses keyed verifier anonymous credentials based on

- Algebraic MACs
- Verifiable encryption
- Tailored credential presentations/ZKPs

All depend on DH assumptions for Ristretto255.

We use variants for other purposes (badges, call links…)

We see no primitive we can drop in and replace.

“Fancy” Crypto Work: Private Groups

Verifiable encryption is an HNDL issue:

- We hold a database of encrypted group membership rosters.
- Verifiable encryption is proof friendly, **not** quantum safe:
  - $\mathtt{Enc(UID)} = (a_1 * \mathtt{Hash(UID)},\ a_1 a_2 * \mathtt{Hash(UID)} + \mathtt{Encode(UID)})$
- Quantum attacker who guesses one pt-ct pair can recover secret.

“Fancy” Crypto Work: Private Groups

We want to eliminate this HNDL exposure.

Verifiable encryption from block cipher + SNARK?

Do we need succintness?

We need to evaluate the full group system, including Sender Keys and Sealed Sender.

“Fancy” Crypto Work: Blind Signatures

We’re also using lightweight Privacy Pass-like credentials based on partially blind signatures.

We rely heavily on the homomorphic properties of the signatures.

It’s largely a spam fighting tool right now, but we may use them more widely.

We don’t have a post-quantum replacement.

“Fancy” Crypto Work: OPRFs

Our new Secure Value Recovery system uses Password Protected Secret Sharing (PPSS).

- The main tool is an OPRF
- It is DH-based

It is not post quantum.

It is an important HNDL target.

“Fancy” Crypto Work: OPRFs

Heavy handed HNDL protection:

- OPRF keys are in enclaves (need this anyway)
- Clients connect through Noise HFS or Noise + pq pre-shared key

What does a fully post-quantum replacement look like?

“Fancy” Crypto Work: VRFs

We have a possible use for Verifiable Random Functions:

- We’re looking at an Ed25519 based VRF
- The “put it in a Noise HFS channel” trick doesn’t work for this application.

We need “fancy” crypto.

Drop in replacements for primitives would be great...

…but the important thing is for us to securely provide our features.

- Signal and its PQ needs
- Boring PQ cryptography is interesting (and we are making practical progress)
- Fancy cryptography is interesting (and we have a lot of questions)
- **Conclude**

Takeaways

- Signal is carefully moving towards post quantum cryptography.
- HNDL protection is an immediate priority.
- We do not want to take away security guarantees from users - so we must hybrid for now.
- Even protocols built with standard primitives have pitfalls - formal verification helps us avoid them.
- We have barely begun, and for our fancy cryptography the problems seem wide open.

We have problems.
We put solutions into practice.

We'd love your help!

Thank you!

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