An Analysis of Signal Messenger's PQXDH

Rolfe Schmidt (Signal)
Charlie Jacomme (Inria)
Karthikeyan Bhargavan (Cryspen)
Franziskus Kiefer (Cryspen)

Formal verification can **speed development** and **clarify security** of real world protocols.

This is important as many protocols are being updated to provide PQ security.

Let’s see how this process worked with the Signal Protocol.

The Signal Protocol

The Signal Protocol

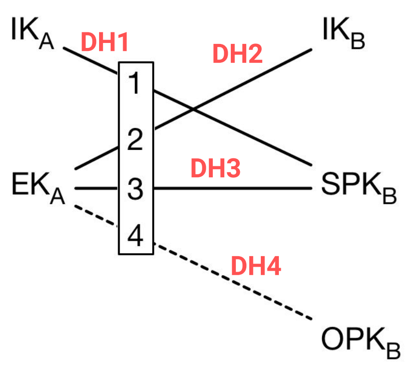
Two parts:

- X3DH handshake
- Double Ratchet for continuous key agreement

Important security guarantees:

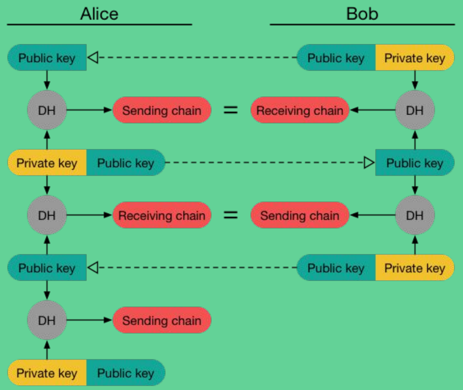
- Confidentiality
- Mutual authentication
- Post-compromise security
- Forward secrecy
- Deniability

X3DH



SK = KDF(**DH1** || **DH2** || **DH3** || **DH4**)

Double Ratchet



The Signal Protocol

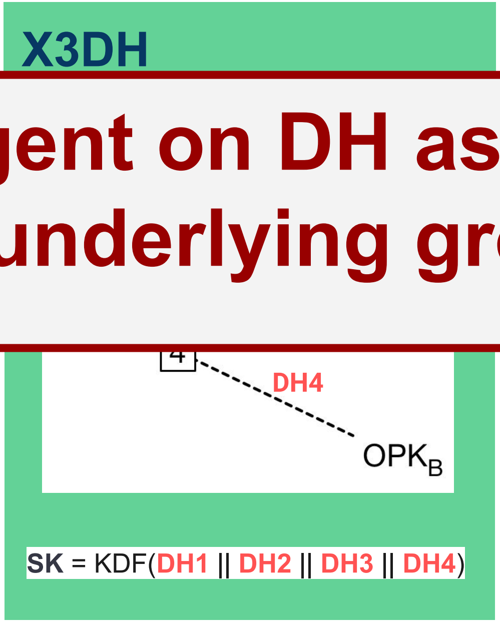
Two parts:

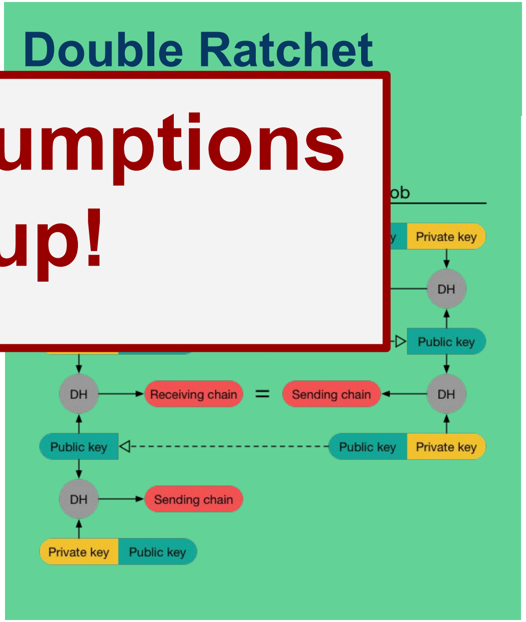
- X3DH handsl
- Double Ratch
  continuous k

Important security

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

Contingent on DH assumptions for the underlying group!





Signal is vulnerable to any future DL solver - quantum *or* classical.

Messages sent today
are vulnerable to
Harvest Now, Decrypt Later
(HNDL) attacks.

The PQXDH Key Agreement Protocol

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

PQXDH Protocol Requirements

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

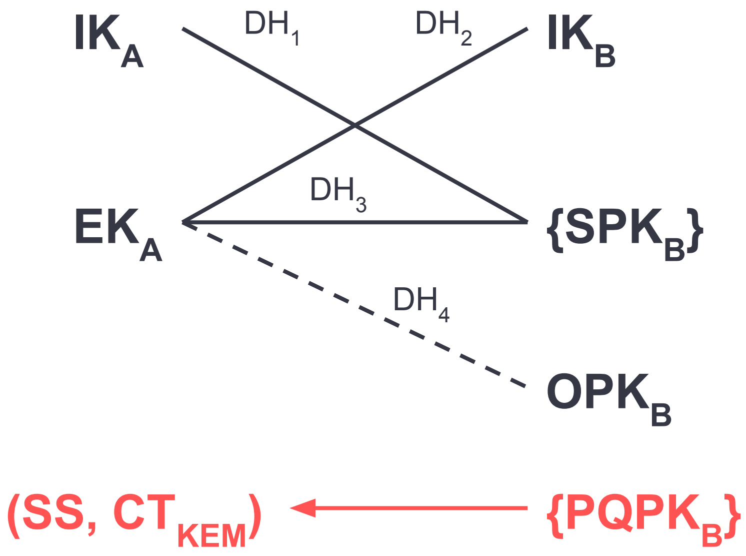
Non-goal: Protect against active quantum attackers

To achieve this we need to add PQ crypto to the X3DH handshake.

A simple idea:

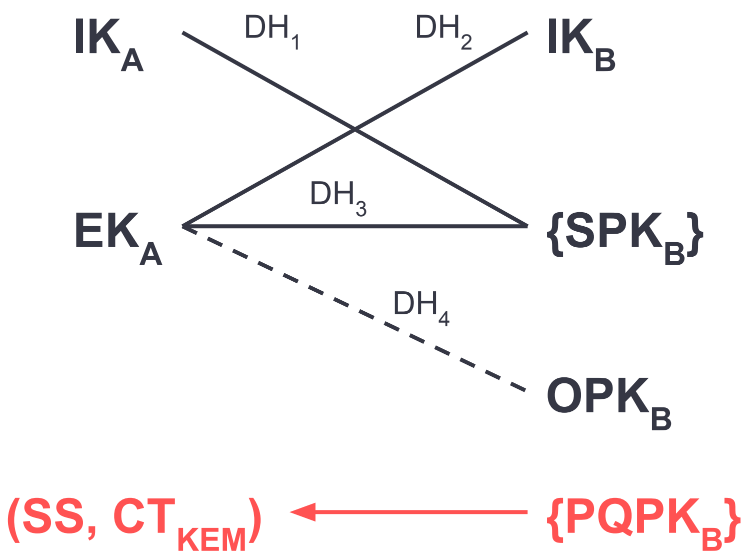
Take X3DH and
add in a PQ-KEM
encapsulated shared secret.

PQXDH Design



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

PQXDH Design



SK = KDF(DH_1 || DH_2 || DH_3 || DH_4 || **SS**)

After computing **SK**, Alex sends Blake:

- (**C**, **CT_{KEM}**, **EK_A^{PK}**) where
- **C** = AEAD.Enc(**SK**, *msg*, AD = **IK_A^{PK}** || **IK_B^{PK}**)

Blake processed the message by:

- Using their EC keys to compute the **DH**'s
- Using their KEM key to decapsulate **SS**
- Computing **SK**
- Computing AEAD.Dec(**SK**, **C**, AD)

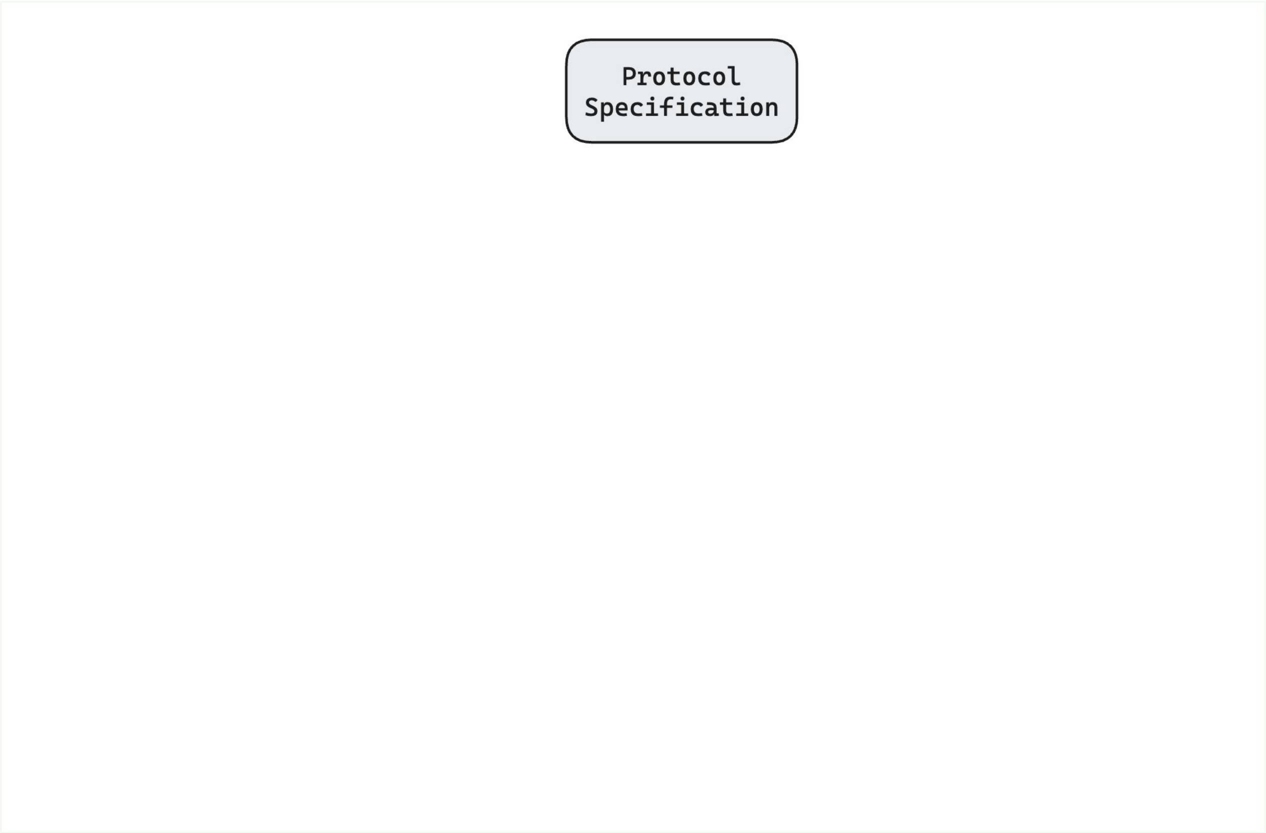
If the decryption succeeds, we have key agreement.

Does PQXDH achieve its goals?

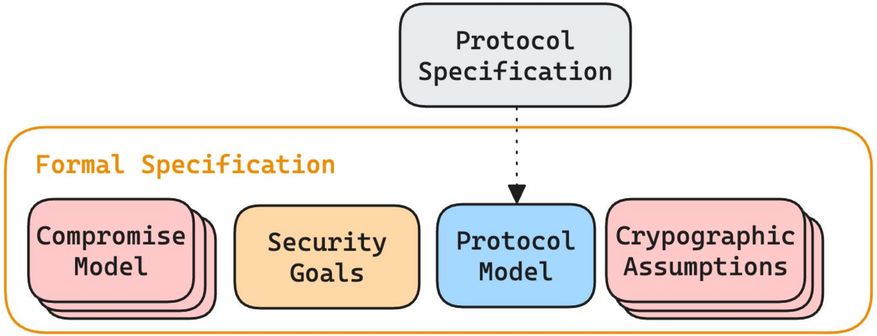
We need to **formally verify** it.

Formally Modelling PQXDH

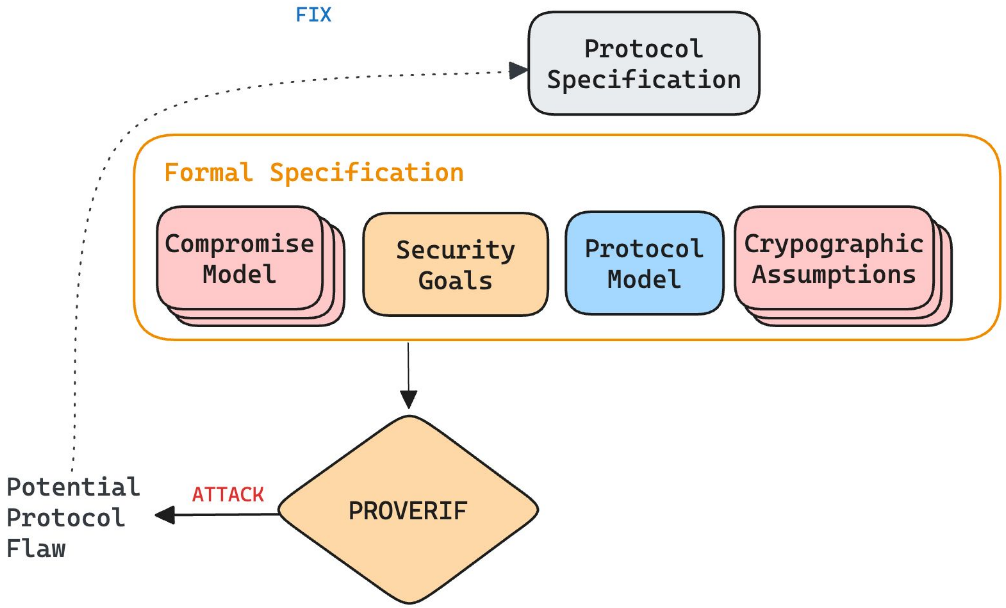
Our Formal Verification Methodology



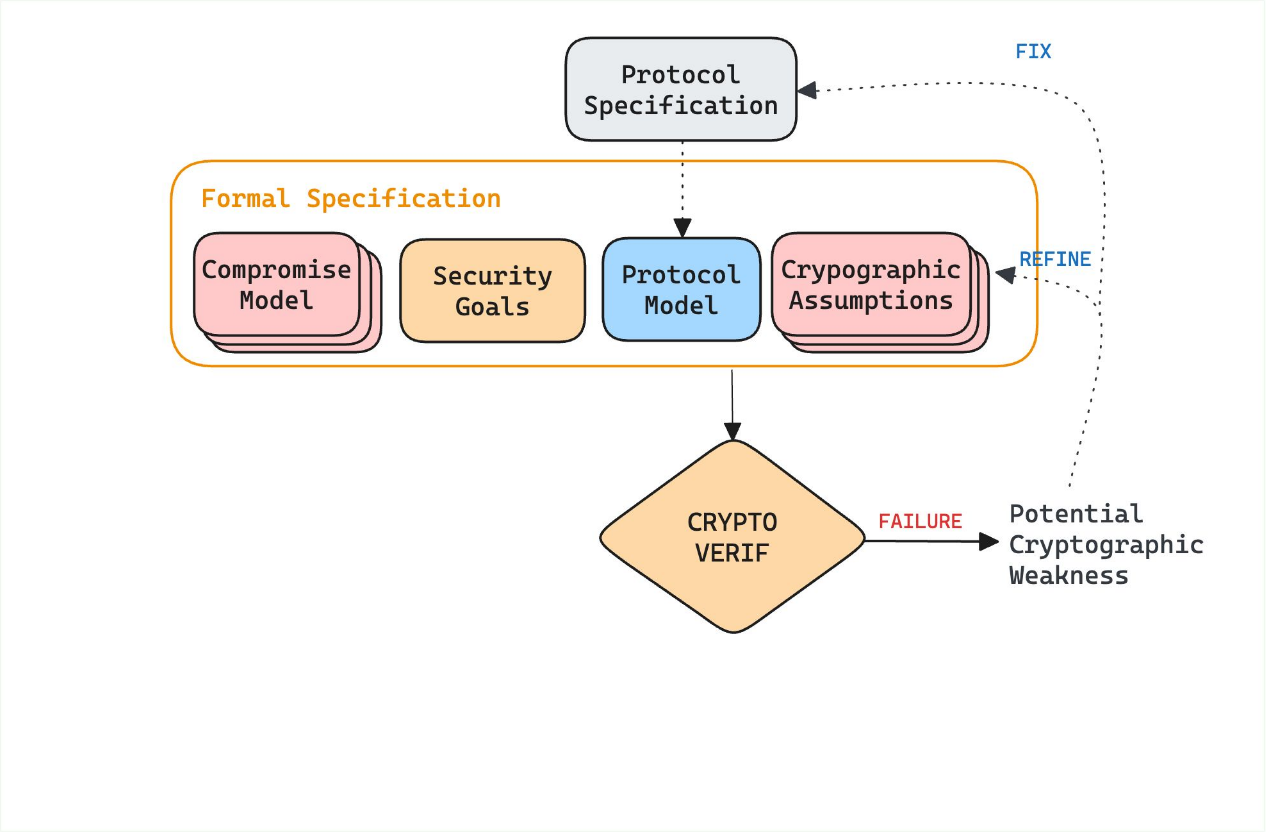
Our Formal Verification Methodology



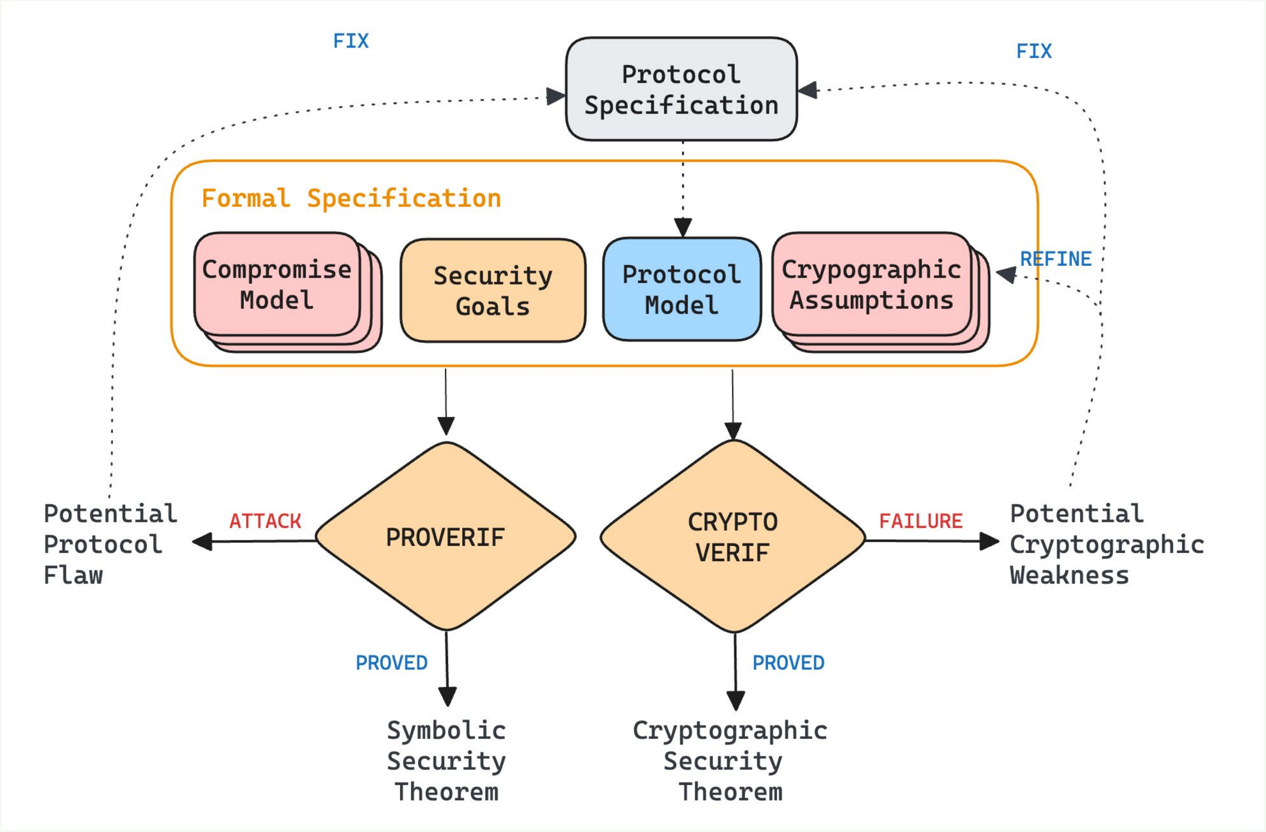
Our Formal Verification Methodology



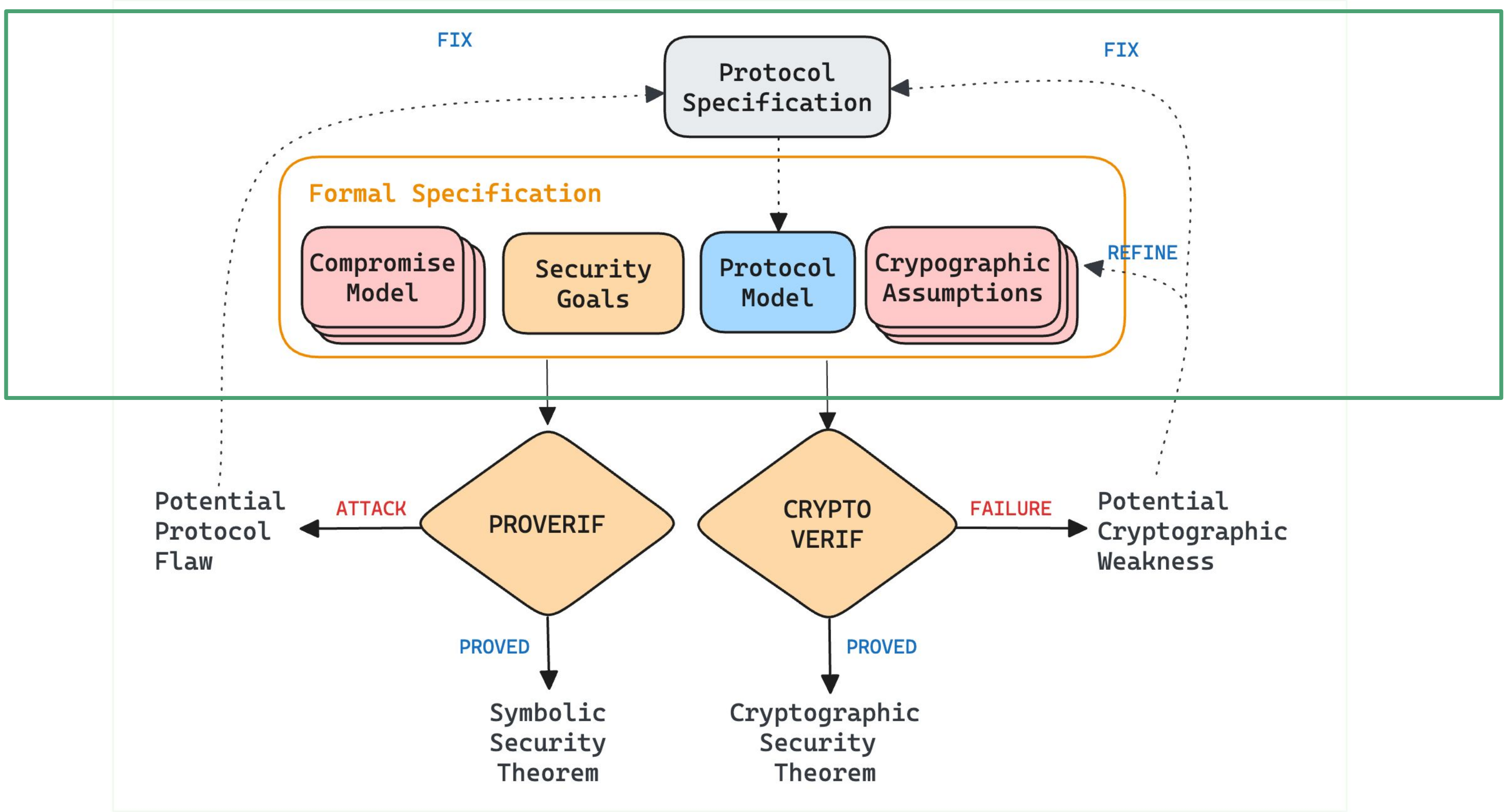
Our Formal Verification Methodology



Our Formal Verification Methodology



Our Formal Verification Methodology



What We Model

Single Message PQXDH Protocol

- Arbitrary number of PQXDH endpoints
- Any endpoint can play any role
- (Out-of-Band) Identity Key Verification
- Untrusted Key Distribution Server

Compromise Scenarios

- Identity keys can be leaked at any time
- OPK, EK, and PQPK can be leaked for certain security goals
- Quantum adversary has explicit power to break all DH primitives

```
let Initiator(i:client, IKA_s:scalar) =
    (* Download Responder Keys *)
    ...

    (* Verify the signatures *)
    if verify(IKB_p,encodeEC(SPKB_p),SPKB_sig) then
    if verify(IKB_p,encodeKEM(PQPKB_p),PQPKB_sig) then

    (* PQXDH Key Derivation*)
    let IKA_p = s2p(IKA_s) in
    let (CT:bitstring,SS:bitstring) =
        pqkem_enc(PQPKB_p) in (* PQ-KEM Encap *)
    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 DH4 = dh(EKA_s,OPKB_p) in
    let SK = kdf(concat5(DH1,DH2,DH3,DH4,SS)) in

    (* Send Message *)
    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

    out(server, (IKA_p,EKA_p,CT,OPKB_p,
                 SPKB_p,PQPKB_p,enc_msg))
```

Symbolic Analysis with ProVerif

Symbolic (Dolev-Yao) Crypto Model

- “Perfect” crypto primitives
- Unbounded number of sessions
- Previously used for Signal, TLS 1.3, ...

Quantum Adversary Model

- Adversary can invert DH

Security Analysis

- Queries for authentication and secrecy
- Fully automated analysis
- Finds attacks or establishes a theorem
- Easy to quickly test fixes

```
(* Post-Quantum Forward Secrecy Query *)
query A, B, spk, pqpk, sk, i, j;
      event(BlakeDone(A,B,spk,pqpk,sk))@i
            ⇒ not(attacker(sk))
              | (event(LongTermComp(A))@j & j < i)
              | (event(QuantumComp)@j & j < i)
```

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).

3. We assume as hypothesis that
attacker(a).

4. We assume as hypothesis that
attacker(b).

5. The message b that the attacker may have by 4 may be received at input {2}.
So the entry identity_pubkeys(b,SMUL(IK_s_2,G)) may be inserted in a table at i
table(identity_pubkeys(b,SMUL(IK_s_2,G))).

Computational Proofs with CryptoVerif

Computational Crypto Model

- Precise Cryptographic Assumptions
- Probabilistic Polynomial-Time Adversary

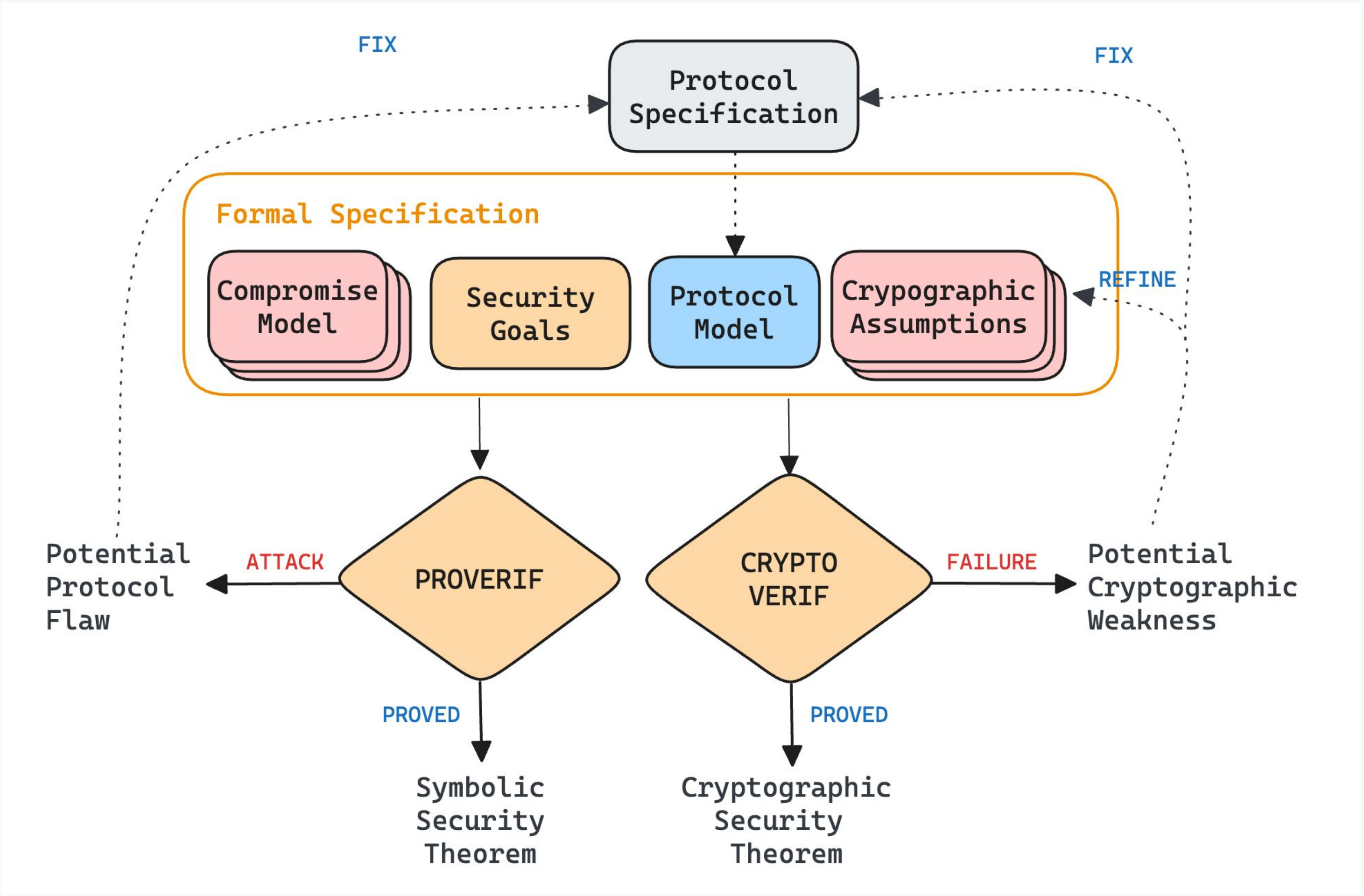
Quantum Adversary Model

- Adversary can (passively) break DH
- Uses new Post-Quantum Soundness results for CryptoVerif proofs

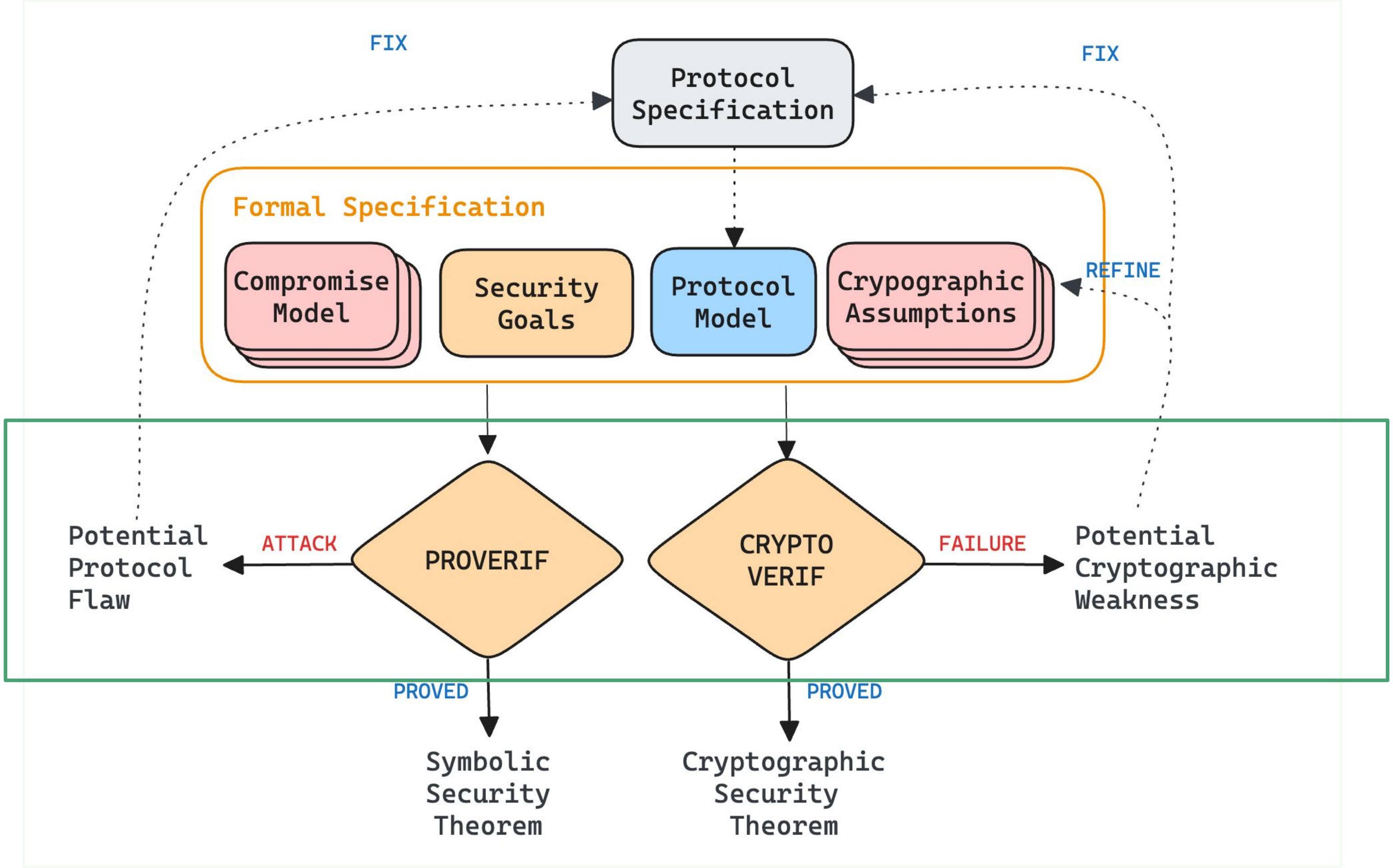
Security Analysis

- Queries for authentication and secrecy
- Game-based machine-checked proofs
- Similar guarantees to pen-and-paper proofs
- Requires manual guidance

```
proof {
crypto uf_cma_corrupt(sign) signAseed;
out_game "g1.cv" occ;

insert before "EKSecA1 <-R Z" ...
insert after "RecvOPK(" ...
out_game "g11.cv" occ;

insert after "OH_1(" ...
crypto rom(H2);
out_game "g2.cv" occ;

insert before "EKSecA1p <-R Z" ...
insert after "RecvNoOPK(" ...
out_game "g12.cv"occ;

insert after "OH(" ...
crypto rom(H1);
out_game "g3.cv";

crypto gdh(gexp_div_8) ...
crypto int_ctxt(enc) *;
crypto ind_cpa(enc) **;
out_game "g4.cv";

crypto int_ctxt_corrupt(enc) r_23;
crypto int_ctxt_corrupt(enc) r_50;
success
}
```



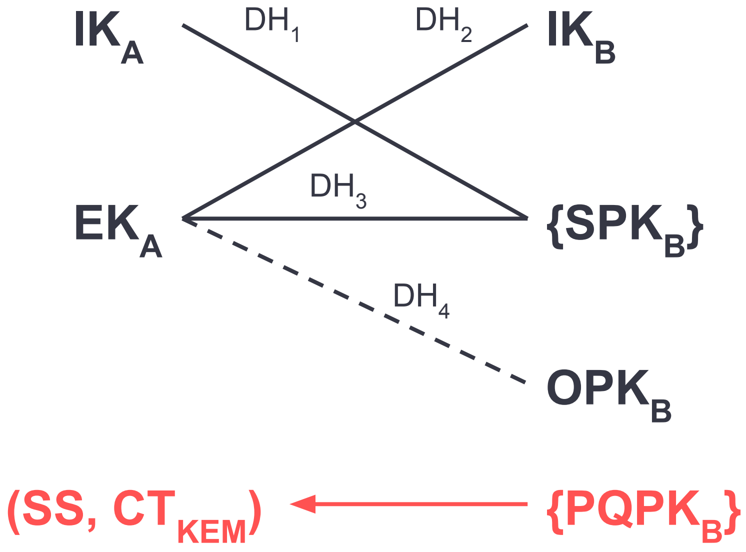
FIX
FIX
Protocol Specification
Formal Specification
Compromise Model
Security Goals
Protocol Model
Crypographic Assumptions
REFINE
Potential Protocol Flaw
ATTACK
PROVERIF
CRYPTO VERIF
FAILURE
Potential Cryptographic Weakness
PROVED
PROVED
Symbolic Security Theorem
Cryptographic Security Theorem



FIX
Protocol Specification
FIX
Formal Specification
Compromise Model
Security Goals
Protocol Model
Crypographic Assumptions
REFINE
Potential Protocol Flaw
ATTACK
PROVERIF
CRYPTO VERIF
FAILURE
Potential Cryptographic Weakness
PROVED
PROVED
Symbolic Security Theorem
Cryptographic Security Theorem

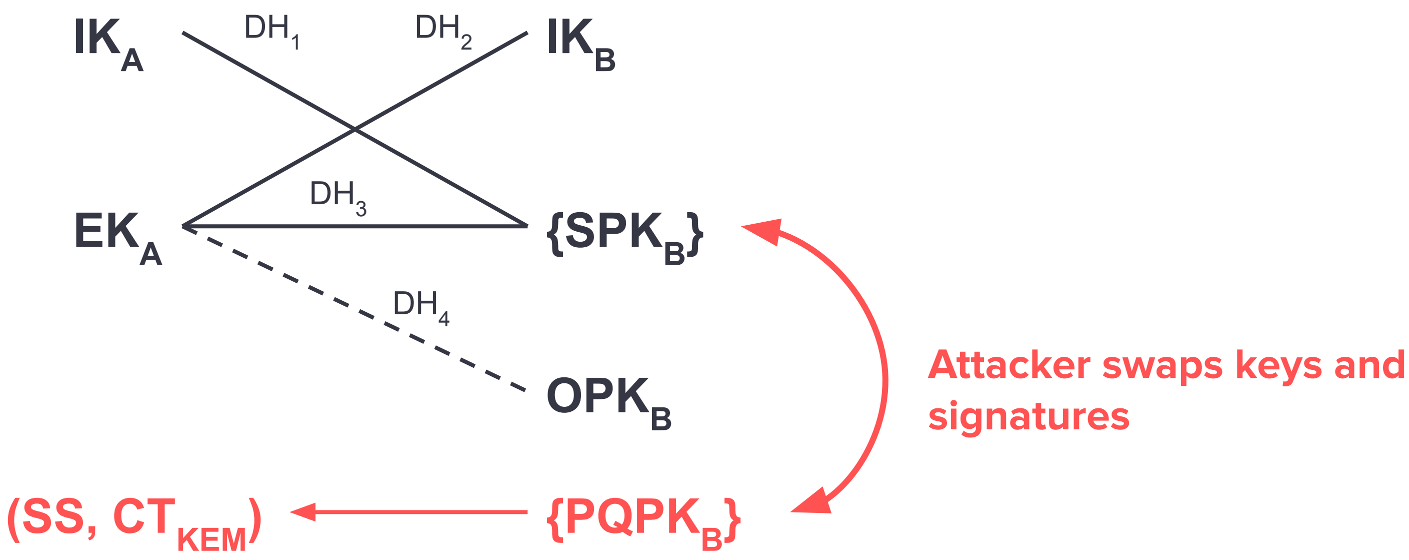
Finding and Confirming Weaknesses

Key Confusion Attack



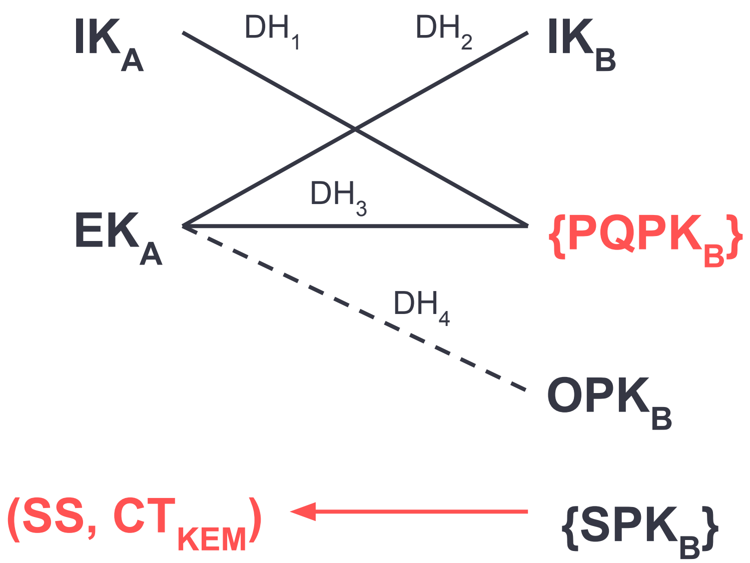
$SK = KDF(DH_1 \parallel DH_2 \parallel DH_3 \parallel DH_4 \parallel SS)$

Key Confusion Attack



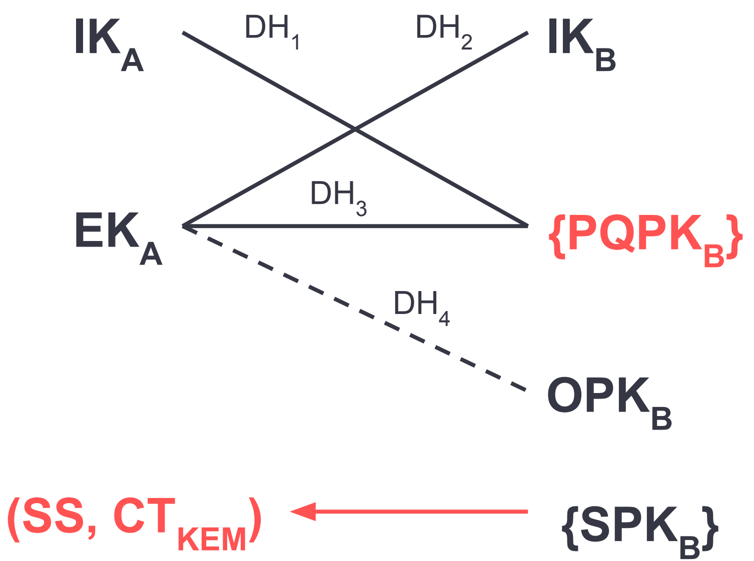
$SK = KDF(DH_1 \| DH_2 \| DH_3 \| DH_4 \| SS)$

Key Confusion Attack



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

Key Confusion Attack



SK = KDF(DH₁ || DH₂ || DH₃ || DH₄ || SS)

Now Alex computes :
(**SS**, **CT**) = KEM.Encaps(**SPK_B^{PK}**)

Without further assumptions about KEM **this is an insecure computation.**

Given **CT** the attacker can now compute **SS**.

We lose PQ security.

This is representative of a general class of cross-protocol attacks between classical and PQ crypto.

Fix: Ensure all key encodings have disjoint co-domains.

KEM Re-encapsulation Vulnerability

First: Attacker obtains the private key for one PQ-KEM key.

Blake's Phone

IK_B

SPK_B

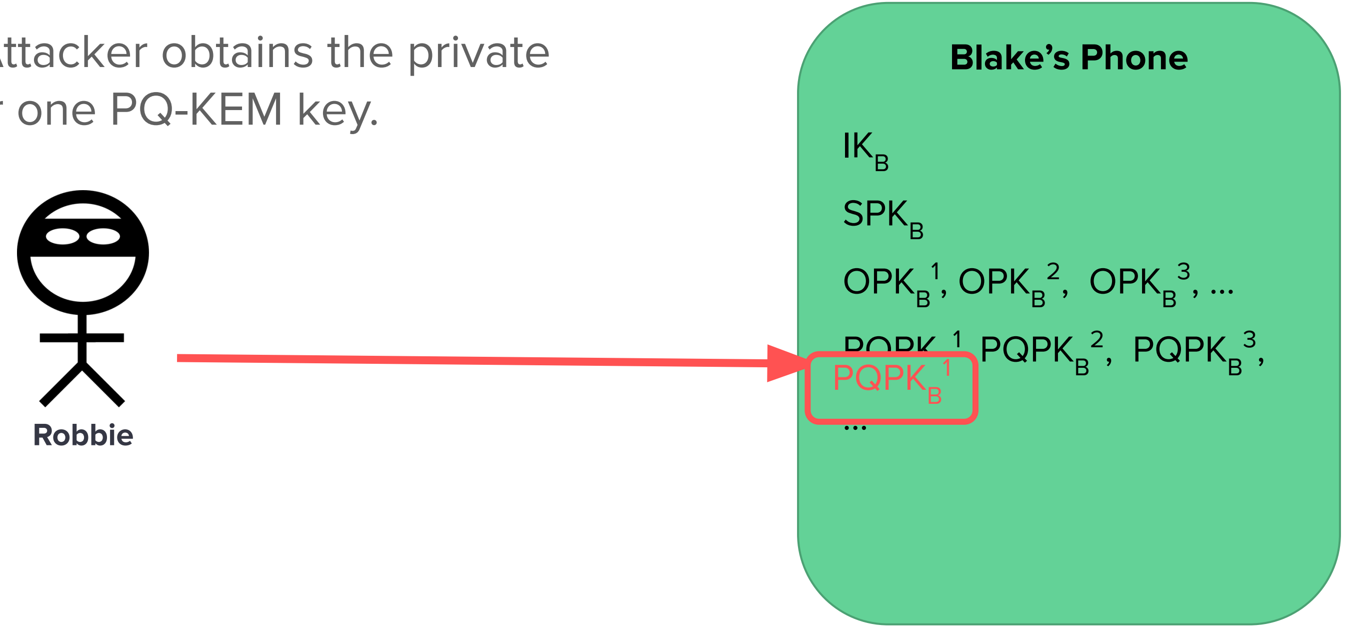
OPK_B^1, OPK_B^2, OPK_B^3, ...

$PQPK_B^1$, $PQPK_B^2$, $PQPK_B^3$,

...

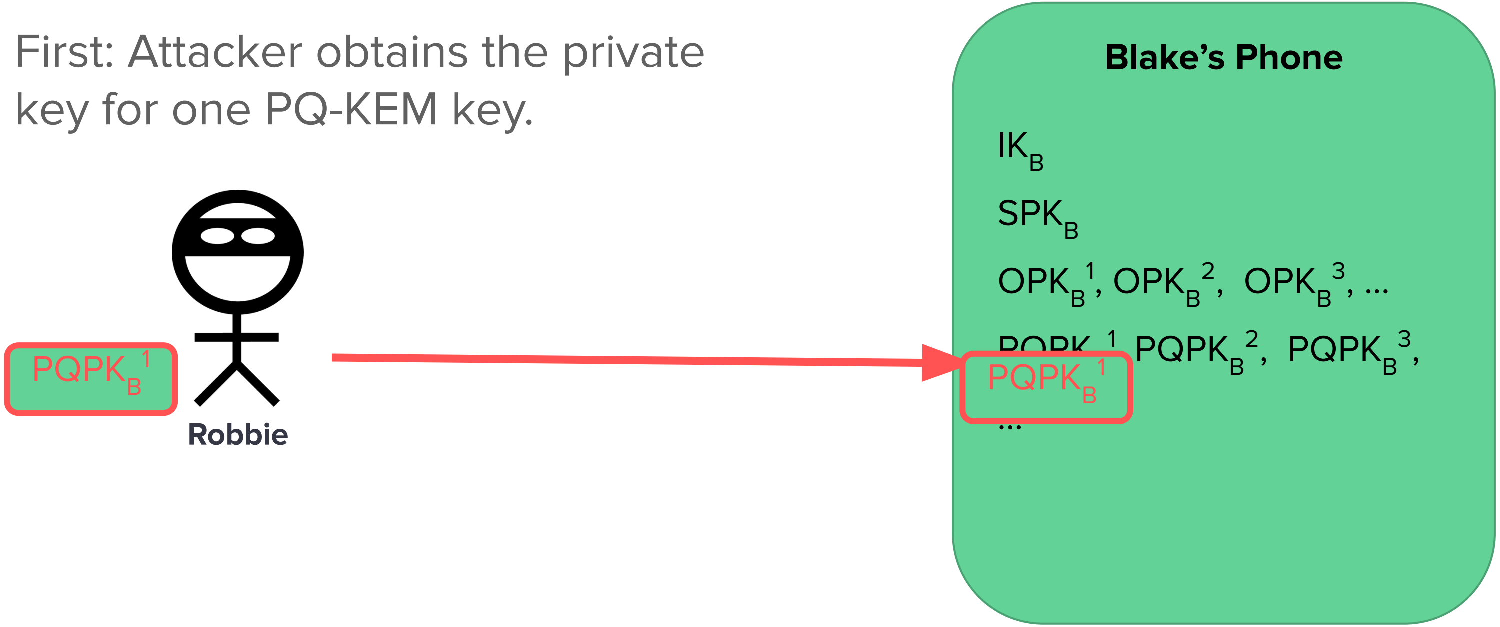
KEM Re-encapsulation Vulnerability

First: Attacker obtains the private key for one PQ-KEM key.



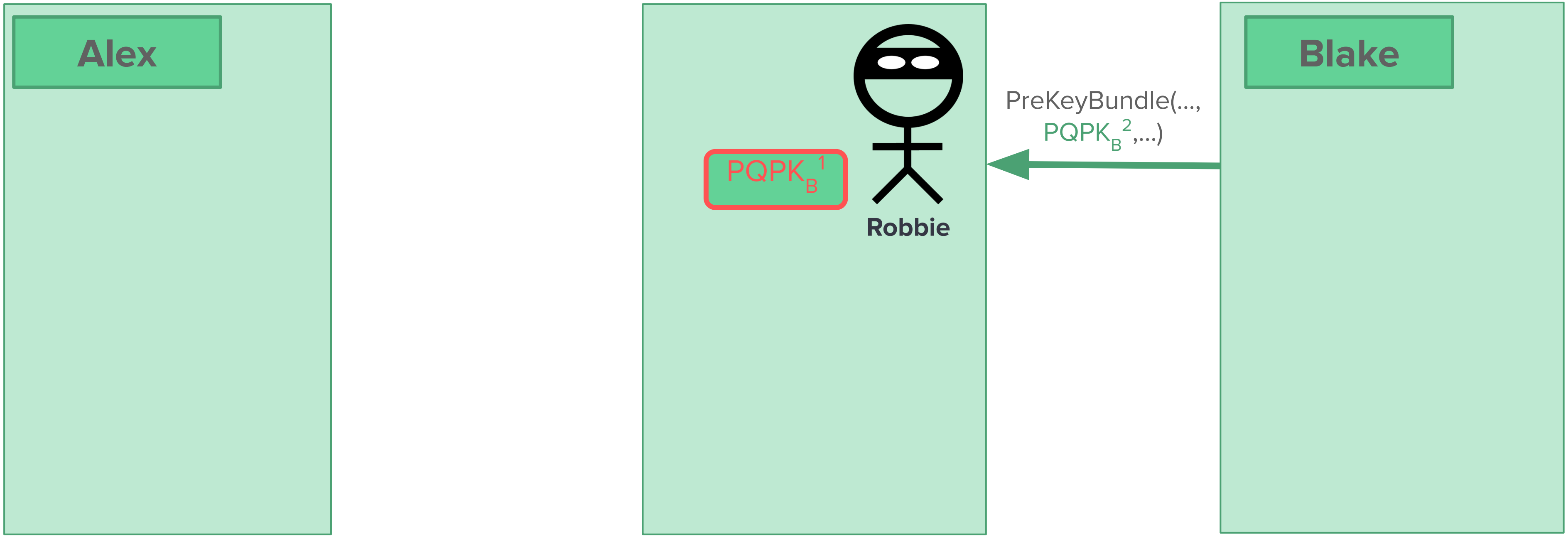
KEM Re-encapsulation Vulnerability

First: Attacker obtains the private key for one PQ-KEM key.



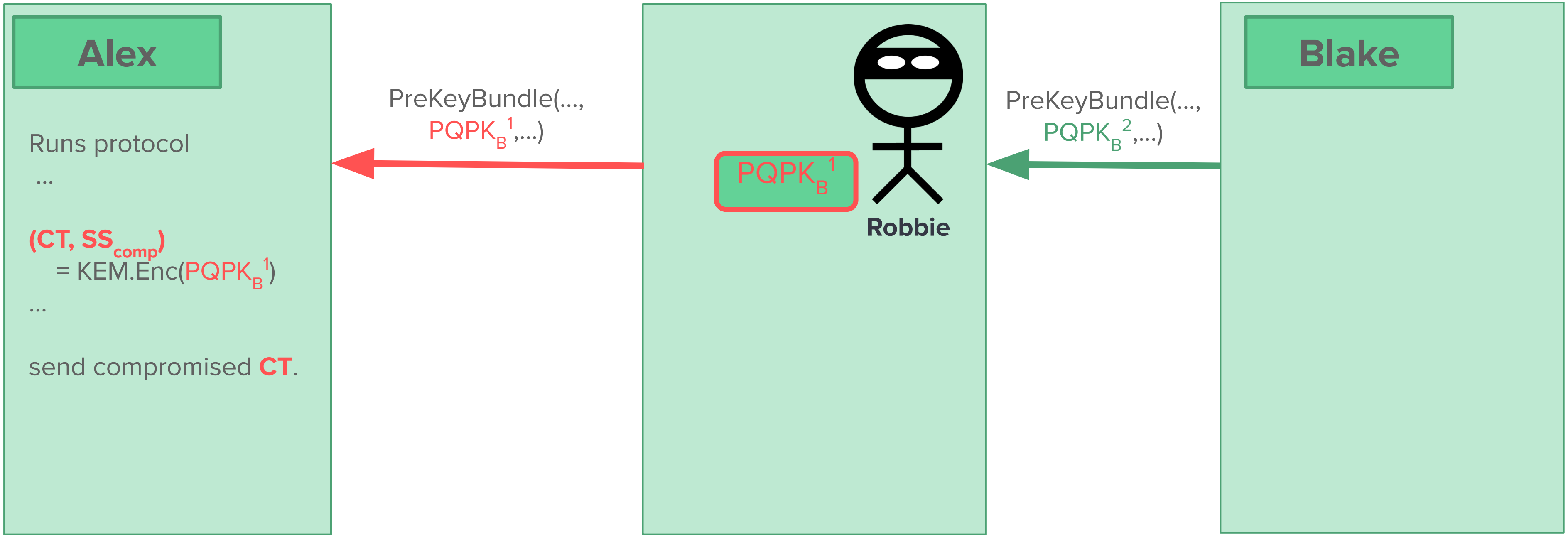
KEM Re-encapsulation Vulnerability

Then: Attacker breaks session independence between multiple PQ-KEM keys using a single compromised key.



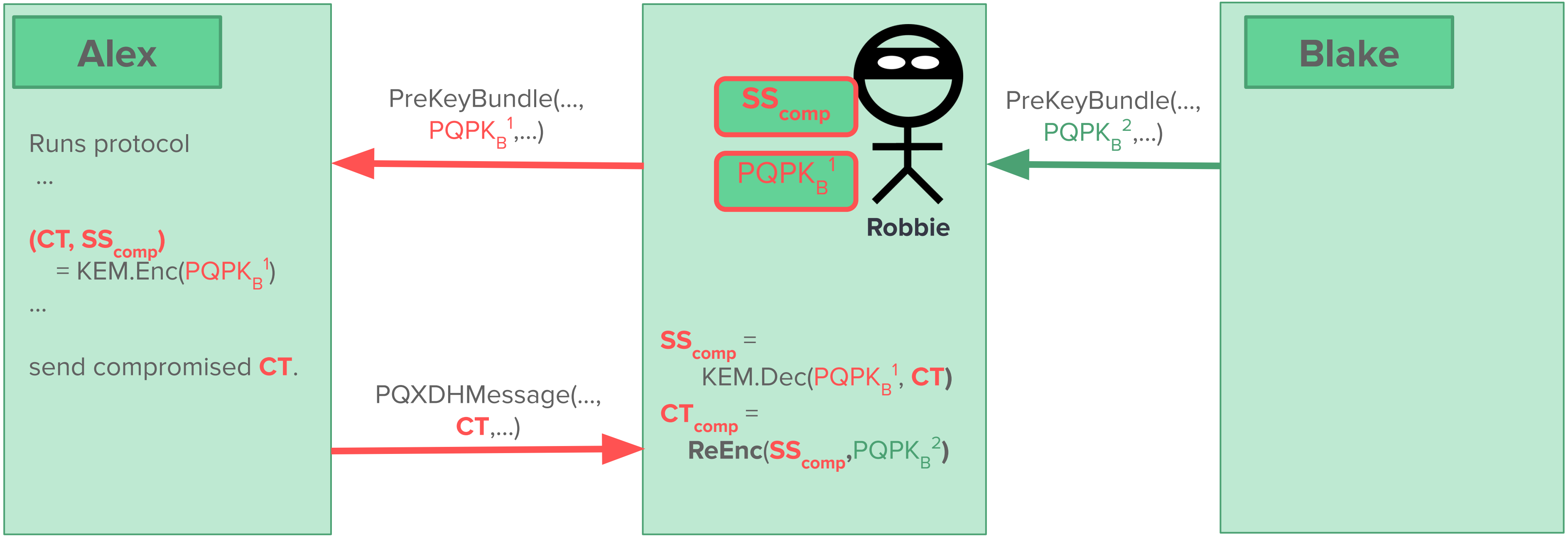
KEM Re-encapsulation Vulnerability

Then: Attacker breaks session independence between multiple PQ-KEM keys using a single compromised key.



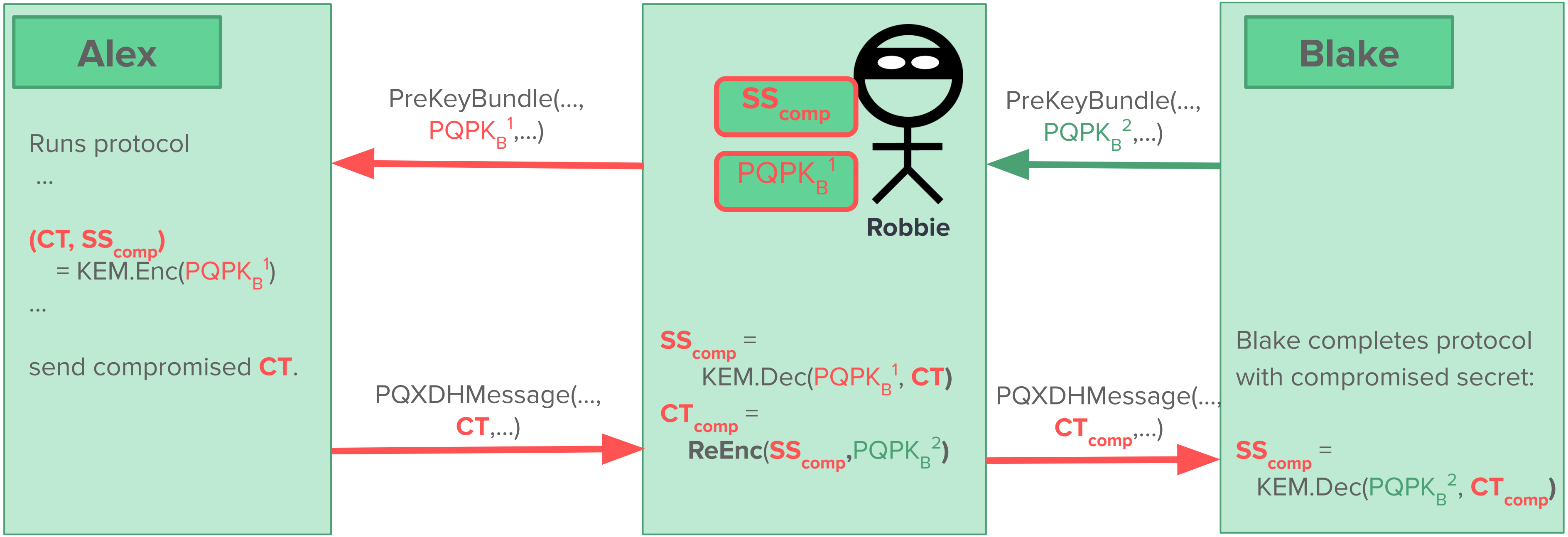
KEM Re-encapsulation Vulnerability

Then: Attacker breaks session independence between multiple PQ-KEM keys using a single compromised key.



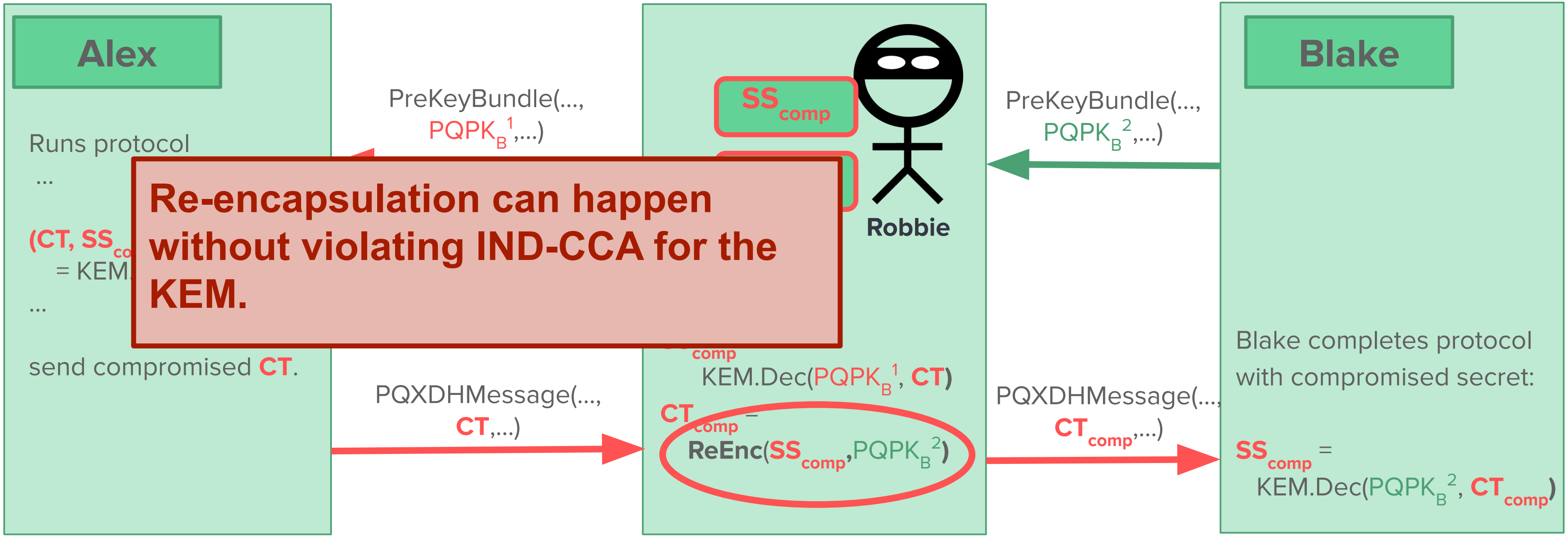
KEM Re-encapsulation Vulnerability

Then: Attacker breaks session independence between multiple PQ-KEM keys using a single compromised key.



KEM Re-encapsulation Vulnerability

Then: Attacker breaks session independence between multiple PQ-KEM keys using a single compromised key.



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.

Fix: Require that the KEM encapsulation binds the recipient's public key

A New Protocol Revision

The Signal Implementation is Secure

Our open-source implementation was never vulnerable:

- Key encodings have disjoint co-domains (and key sizes are different).
- Kyber public keys are contributory to the KEM shared secret.

The Signal Implementation is Secure

Our open-source implementation was never vulnerable:

- Key encodings have disjoint co-domains (and key sizes are different).
- Kyber public keys are contributory to the KEM shared secret.

But we did want to add restrictions to the protocol description.

After iterating, the models:

- reflected our security goals,
- captured key implementation details,
- guided a new protocol revision,
- and yielded security proofs.

PQXDH Protocol Revisions

The findings led to a new revision of the protocol:

- We added **AEAD** as a parameter and required it to be post-quantum **IND-CPA** and **INT-CTXT**
- Added description of key identifier use
- Restricted the ranges of encodings to be disjoint
- Added $\mathbf{PQPK_B^{PK}}$ to AD when it isn't contributory to the KEM

PQXDH Protocol Revisions

The findings led to a new revision of the protocol:

- We added **AEAD** as a parameter and required it to be post-quantum **IND-CPA** and **INT-CTXT**
- Added description of key identifier use **Not security relevant**
- Restricted the ranges of encodings to be disjoint
- Added $\mathbf{PQPK_B^{PK}}$ to AD when it isn't contributory to the KEM

PQXDH Protocol Revisions

The findings led to a new revision of the protocol:

- We added **AEAD** as a parameter and required it to be post-quantum **IND-CPA** and **INT-CTXT**
- Added description of key identifier use
- Restricted the ranges of encodings to be disjoint **Prevent Key Confusion Attack**
- Added $\mathbf{PQPK_B^{PK}}$ to AD when it isn't contributory to the KEM

PQXDH Protocol Revisions

The findings led to a new revision of the protocol:

- We added **AEAD** as a parameter and required it to be post-quantum **IND-CPA** and **INT-CTXT**
- Added description of key identifier use
- Restricted the ranges of encodings to be disjoint
- Added **$PQPK_B^{PK}$** to AD when it isn't contributory to the KEM

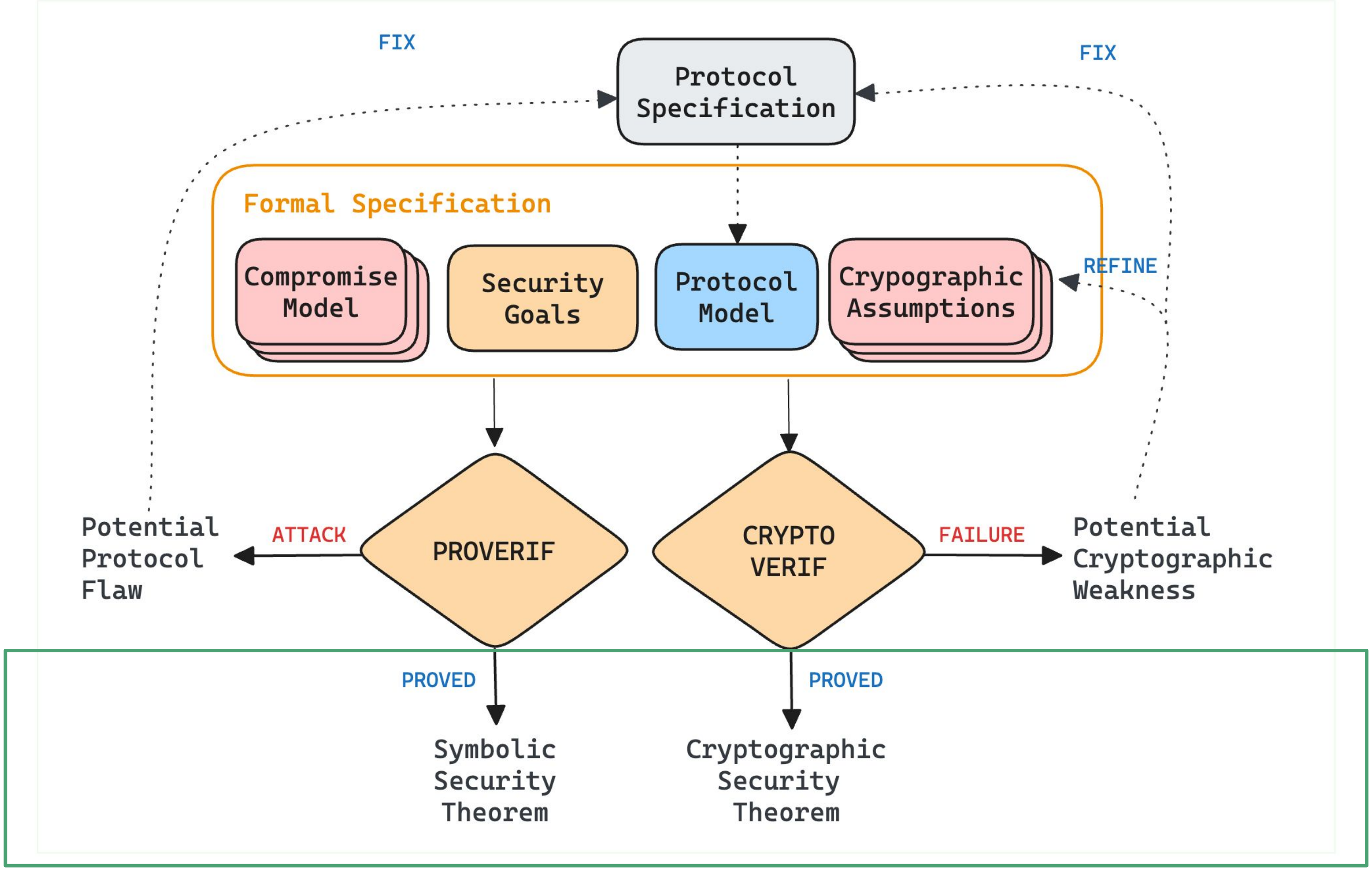
Prevent KEM Re-encapsulation Attack

PQXDH Protocol Revisions

The findings led to a new revision of the protocol:

- We added **AEAD** as a parameter and required it to be post-quantum **IND-CPA** and **INT-CTXT**
- Added description of key identifier use
- Restricted the ranges of encodings to be disjoint
- Added $\mathbf{PQPK_B^{PK}}$ to AD when it isn't contributory to the KEM

With these changes we can prove that PQXDH meets its classical and PQ security requirements in the symbolic, computational, and HNDL models.


FIX
Protocol Specification
FIX
Formal Specification
Compromise Model
Security Goals
Protocol Model
Crypographic Assumptions
REFINE
Potential Protocol Flaw
ATTACK
PROVERIF
CRYPTO VERIF
FAILURE
Potential Cryptographic Weakness
PROVED
PROVED
Symbolic Security Theorem
Cryptographic Security Theorem

Conclusion

- Designing PQ protocols is about more than just swapping in PQ crypto.
- There are many potential pitfalls, some of which we found in PQXDH.

- Formal verification can help find and prevent attacks in PQ protocols.
- Combining symbolic and computational analyses gives better results.

- Close collaboration between protocol designers and proof engineers can provide quick turnaround and help guide protocol revisions
- Signal will continue using formal verification to analyze future protocols.