Formal Specification and Verification of Post-quantum Cryptographic Protocols with Proof Scores

Kazuhiro Ogata
Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology（JAIST）

2nd Workshop on Logic, Algebra and Category Theory: LAC 2025
Fukuoka, September 29 – October 3, 2025

Outline

- Formal verification of post-quantum (PQ) hybrid OpenPGP, where both PQ cryptographic primitives and classical cryptographic primitives are used
- Formal verification of PQ hybrid SSH, where both PQ cryptographic primitives and classical cryptographic primitives are used

Quantum Computers as Security Threats

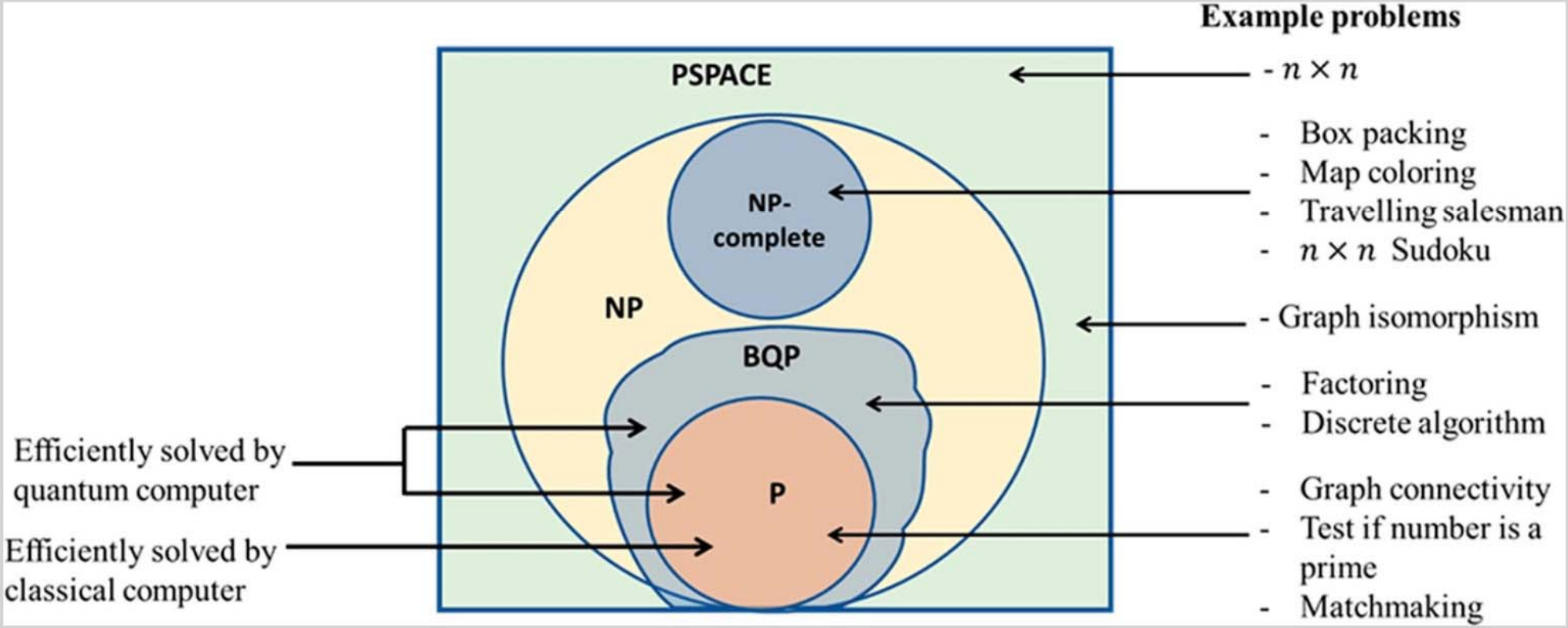
- An idea of quantum computers proposed by Faynman, etc. early-80's
- Google, etc. have been spending many resources (money, humans, etc.) toward implementation of large-scale quantum computers
- Shor invented the quantum algorithms that can efficiently solve Integer Factorization and Discrete Logarithm Problem in 1994.

Quantum Computers as Security Threats

- Public-key encryption schemes, such as RSA, currently used will become insecure and unsafe when large-scale quantum computers are available
- Cryptographic primitives, such as KEMs, resistant to quantum computers are actively studied (in the middle of selection of future standard ones by NIST)
- Development of technologies that can be used to guarantee that post-quantum cryptographic protocols are really secure and safe is an urgent research topic

Quantum Computers as Security Threats

Quantum computing supposes a threat to security



In computational complexity theory, **bounded-error quantum polynomial time** (**BQP**) is the class of decision problems solvable by a quantum computer in polynomial time, with an error probability of at most 1/3 for all instances. It is the quantum analogue to the complexity class **BPP**.（from Wikipedia）

Countermeasure

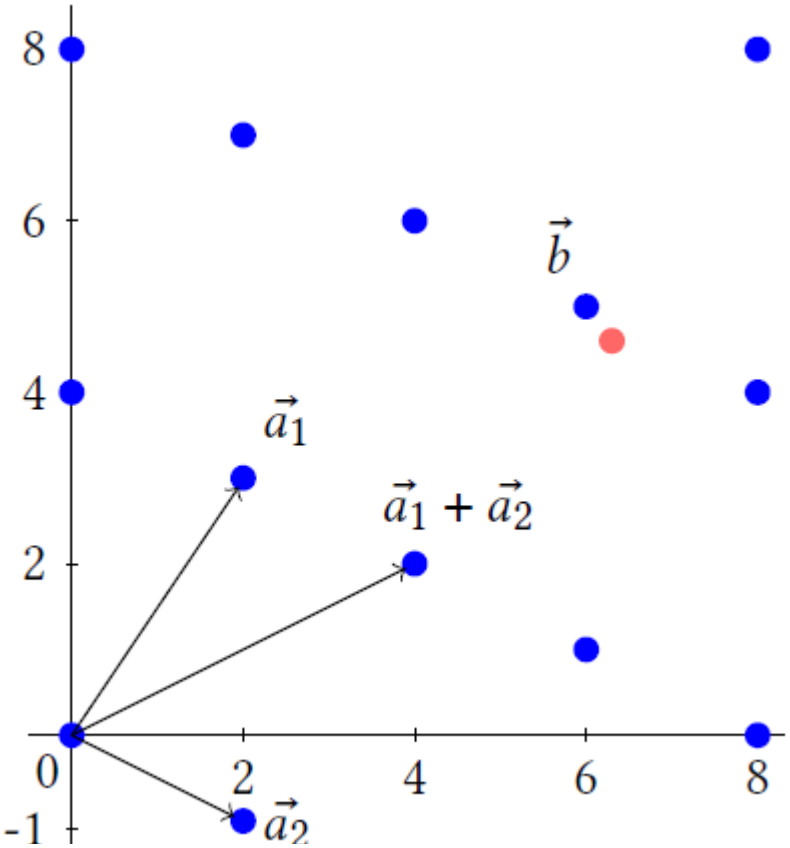
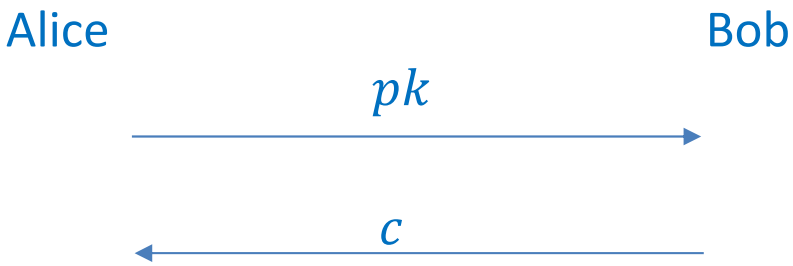


Fig. 2. An illustration of the closest vector problem in 2-dimensional lattice $\mathcal{L}\{\vec{a}_1, \vec{a}_2\}$, where $\vec{a}_1 = (2, 3)$ and $\vec{a}_2 = (2, -1)$



Definition 3.1. A key encapsulation mechanism (KEM) is a tuple of algorithms (`KeyGen`, `Encaps`, `Decaps`) along with a finite key space $\mathcal{K}$:

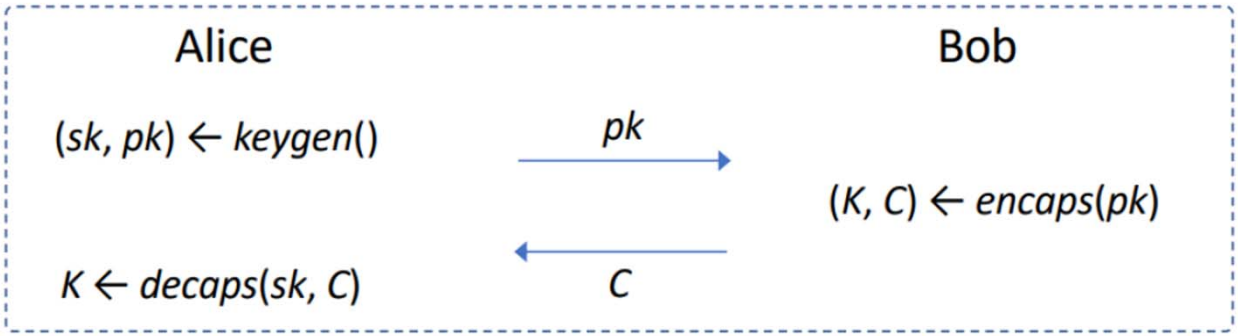
- `KeyGen`() $\rightarrow (pk, sk)$: A probabilistic *key generation* algorithm that outputs a public key pk and a secret key sk.
- `Encaps`(pk) $\rightarrow (c, k)$: A probabilistic *encapsulation* algorithm that takes as input a public key pk, and outputs an encapsulation (or ciphertext) c and a shared secret $k \in \mathcal{K}$.
- `Decaps`(c, sk) $\rightarrow k$: A (usually deterministic) *decapsulation* algorithm that takes as inputs a ciphertext c and a secret key sk, and outputs a shared secret $k \in \mathcal{K}$.

Countermeasure

Key Encapsulation Mechanism (KEM)

- A KEM is a tuple of algorithms (*keygen, encaps, decaps*):

1. *(sk, pk) <- keygen()*: a probabilistic function, outputs a public key *pk* and a secret key *sk*
2. *(K, C) <- encaps(pk)*: a probabilistic function, takes the public *pk*, and outputs a ciphertext *C* and a shared secret key *K*
3. *K <- decaps(sk,C)*: a deterministic function, takes the secret key *sk*, a ciphertext *C,* and outputs the shared secret key *K*



Tools & Techniques Used

- Observational Transition Systems (OTSs) – mathematical model formalizing protocols
- CafeOBJ – proof score-based interactive theorem proving
- CafeInMaude – World's 2nd implementation of CafeOBJ in Maude, equipped with a proof assistant (CiMPA) and a proof generator (CiMPG)
- Invariant Proof Score Generator (IPSG) –automatically generating proof scores for a formal specification nadf a property specification (and lemmas).

Adrian Riesco

Duong Dinh Tran

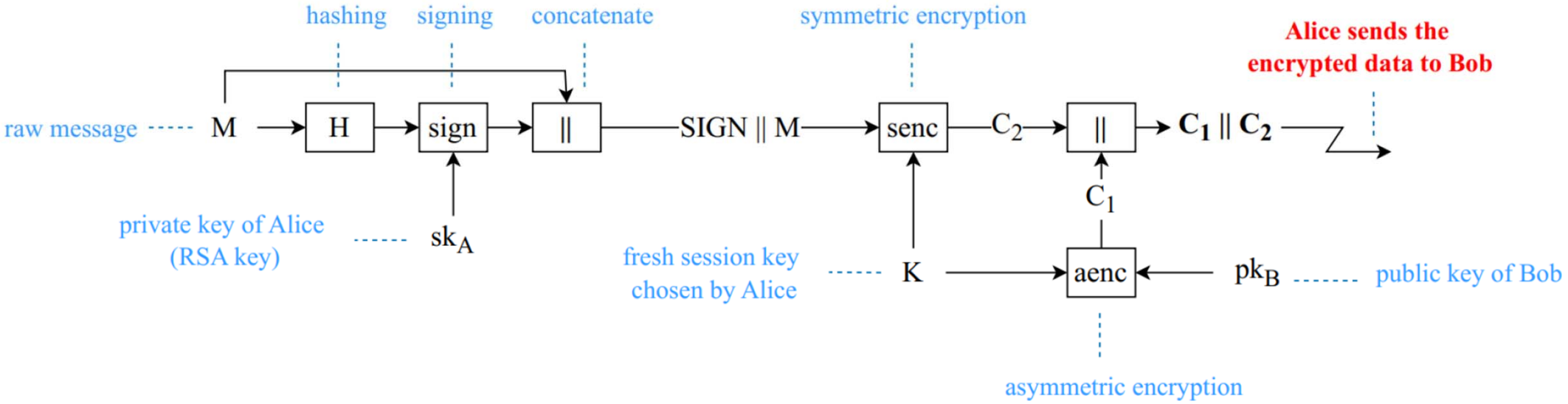
Formal Specification of PQ OpenPGP

- OpenPGP has been often used to secure emails
- A post-quantum (PQ) version has been proposed, where both classical and post-quantum cryptographic primitives are used, because the implementation of the latter may not be matured enough, while the implementation of the former has been matured.

Formal Specification of PQ OpenPGP

OpenPGP

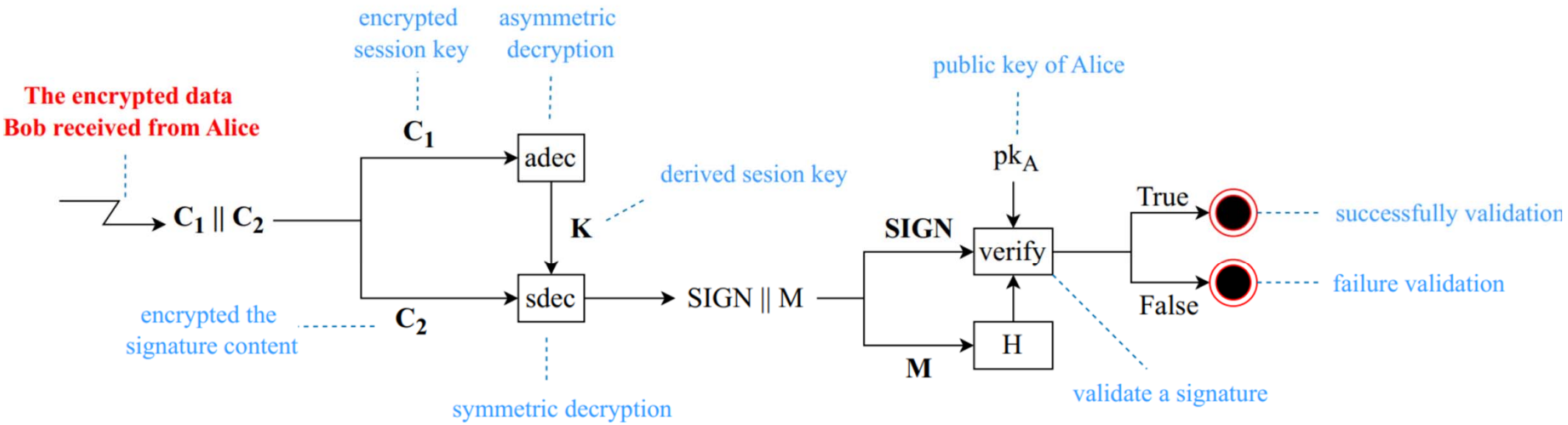
- OpenPGP is an open standard of PGP (Pretty Good Privacy), the most widely used email/file encryption standard.
- Alice sends the message to Bob:



Formal Specification of PQ OpenPGP

OpenPGP

- When Bob receives the message from Alice:



Formal Specification of PQ OpenPGP

- PQ OpenPGP uses:
  1. Elliptic Curve Diffie–Hellman (ECDH) + Module-lattice KEM (ML-KEM) for hybrid key encapsulations
  2. Edwards-curve Digital Signature Algorithm (EdDSA) + Module-lattice DSA (ML-DSA) for hybrid digital signatures

Formal Specification of PQ OpenPGP

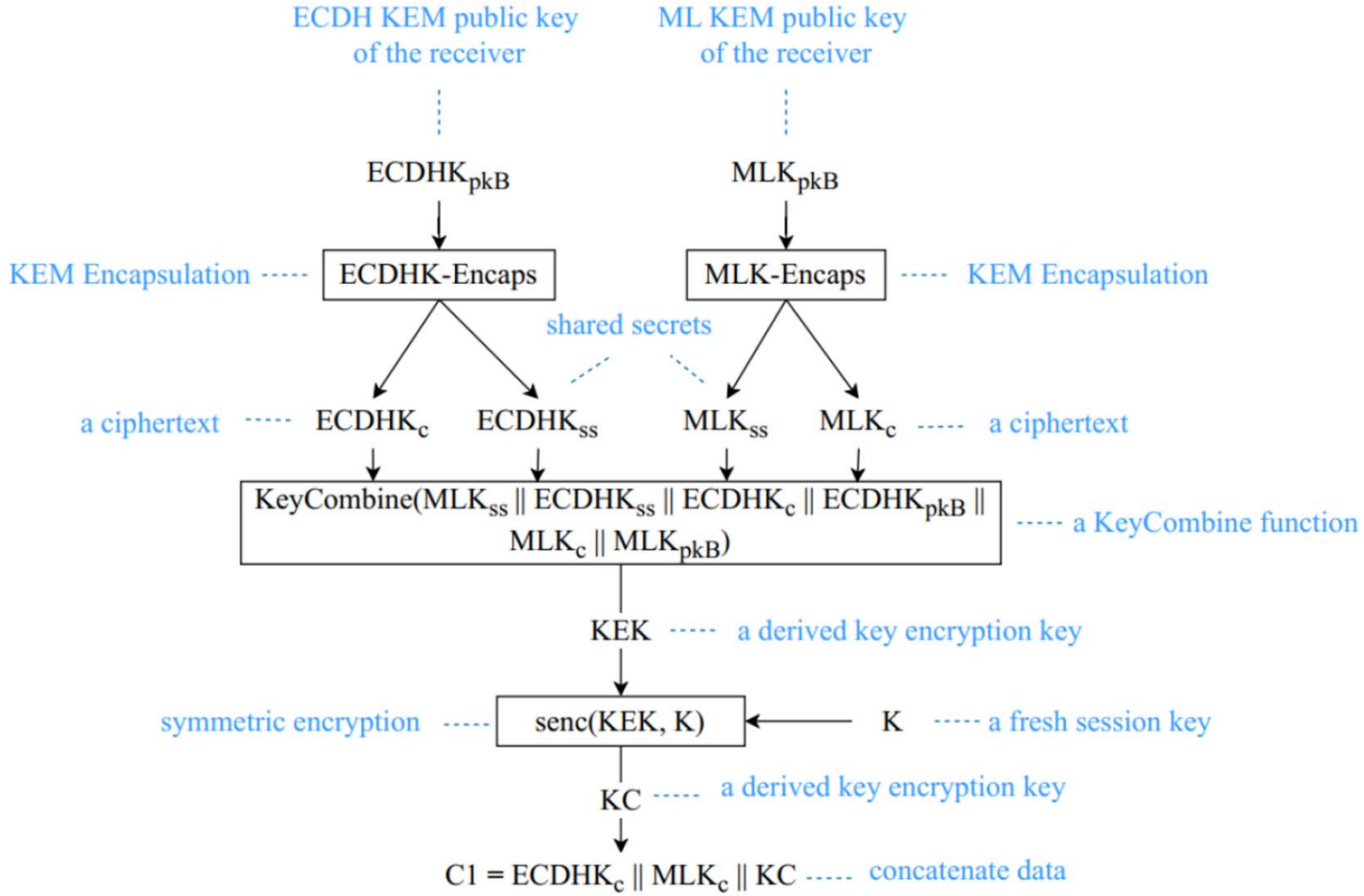
Post-quantum extension of OpenPGP

1) Composite KEMs

ML-KEM + ECDH KEM

for hybrid key

encapsulations

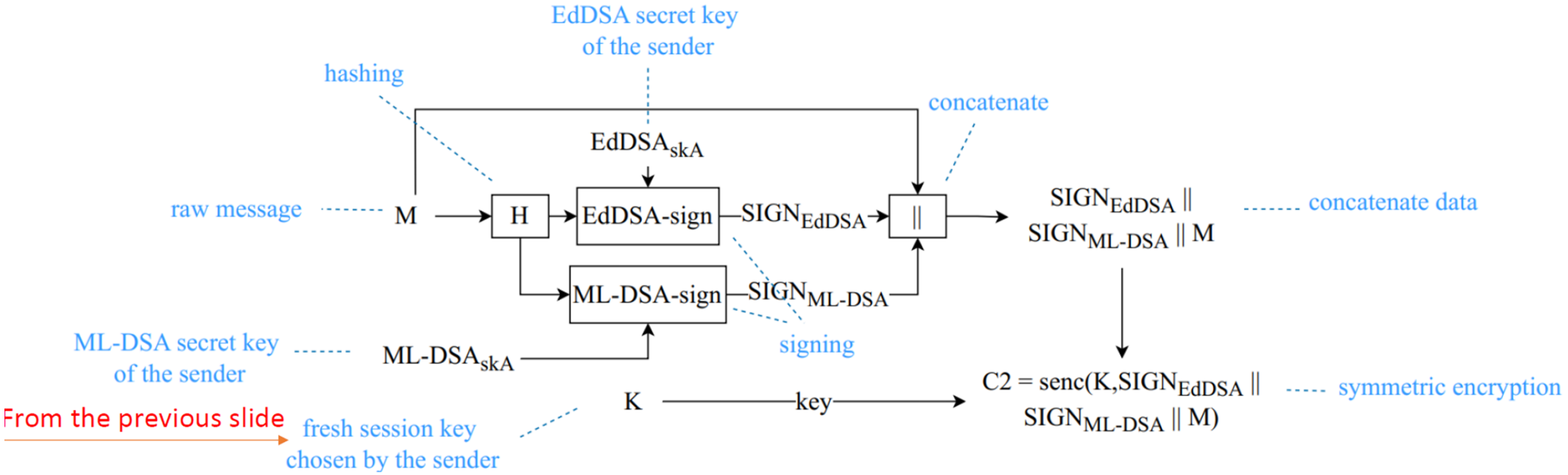


Formal Specification of PQ OpenPGP

Post-quantum extension of OpenPGP

2) Composite digital signatures ML-DSA + EdDSA

MUST successfully validate both signatures



Formal Specification of PQ OpenPGP

Modeling ML-KEM

- To specify a probabilistic function as a deterministic function in CafeOBJ, an argument is added as a random parameter.
- Original: keygen() -> (sk, pk)
- How to represent the mapping between sk and pk?
- CafeOBJ: keygen(sk) -> pk

```
1  vars K' K2' K3' : MLK-SecretK
2  vars PK'        : MLK-PublicK
3  vars C          : MLK-Cipher
4
5  op mlk-keygen   : MLK-SecretK                  -> MLK-PublicK {constr}
6  op mlk-encapsC  : MLK-PublicK MLK-SecretK      -> MLK-Cipher  {constr}
7  op mlk-encapsK  : MLK-PublicK MLK-SecretK      -> MLK-ShareS
8  op mlk-decaps   : MLK-Cipher MLK-SecretK       -> MLK-ShareS
9  op _&_          : MLK-SecretK MLK-SecretK      -> MLK-ShareS  {constr}
10
11 eq mlk-encapsK(PK',K')  = (mlk-getSeci
12 ceq mlk-decaps(C, K')   = (K' & mlk-ge
13                           if (mlk-getP                -keygen(K'))
                                .
14 ceq (mlk-decaps(C, K')  = (K' & K2')) = false
15                           if not(K2' = mlk-getSecret(C)) .
16 ceq (mlk-decaps(C, K')  = (K2' & K3')) = false
17                           if not(K'  = K2') .
```

Long-term private key from a recipient

Ephemeral private key from a sender

Formal Specification of PQ OpenPGP

Protocol execution: Encrypt a message

```
1  eq KEK(B,Mlk-SK2,Ecdhk-SK2)   = kcombine(
2                    mlk-encapsK(MLK-PubK(B),Mlk-SK2)          ||
3                    ecdhk-encapsK(ECDHK-PubK(B),Ecdhk-SK2)    ||
4                    ecdhk-encapsC(ECDHK-PubK(B),Ecdhk-SK2)    ||
5                    ECDHK-PubK(B)                             ||
6                    mlk-encapsC(MLK-PubK(B),Mlk-SK2)          ||
7                    MLK-PubK(B)) .
8  eq C1(B,Mlk-SK2,Ecdhk-SK2,K) =
9                    (ecdhk-encapsC(ECDHK-PubK(B),Ecdhk-SK2)   ||
10                   mlk-encapsC(MLK-PubK(B),Mlk-SK2)          ||
11                   senc(KEK(B,Mlk-SK2,Ecdhk-SK2),K)) .
12
13 eq SIGN(A,M)   = EdDSA-Sign(EdDSA-PriK(A),h(M))             ||
14                   MLDSA-Sign(MLDSA-PriK(A),h(M))            ||
15                   M .
16 eq C2(A,M,K)   = senc(K,SIGN(A,M)) .
```

a KeyCombine function to produce Key Encryption Key

hybrid key encapsulations

composite digital signatures

encrypted digital signatures

Formal Specification of PQ OpenPGP

Protocol execution: Send a message

```
1  eq c-send(S,Ecdhk-SK2,Mlk-SK2,K)                                    =
2   (not(Ecdhk-SK2 \in usecret(S)) and
3    not(Mlk-SK2   \in usecret(S)) and
4    not(K         \in usecret(S))) .
5
6  ceq nw(send(S,A,B,M,K,Ecdhk-SK2,Mlk-SK2))                           =
7   (msg(A,A,B, C1(B,Mlk-SK2,Ecdhk-SK2,K) ||
8   C2(A,M,K), time(S)), nw(S))
9   if c-send(S,Ecdhk-SK2,Mlk-SK2,K) .
10
11 ceq usecret(send(S,A,B,M,K,Ecdhk-SK2,Mlk-SK2))                      =
12  (K Ecdhk-SK2 Mlk-SK2 usecret(S))
13  if c-send(S,Ecdhk-SK2,Mlk-SK2,K) .
14
15 ceq time(send(S,A,B,M,K,Ecdhk-SK2,Mlk-SK2))                         =
16  s(time(S)) if c-send(S,Ecdhk-SK2,Mlk-SK2,K) .
17
18 ceq knl(send(S,A,B,M,K,Ecdhk-SK2,Mlk-SK2))                          =
19  (C1(B,Mlk-SK2,Ecdhk-SK2,K) || C2(A,M,K) || knl(S))
20  if c-send(S,Ecdhk-SK2,Mlk-SK2,K) .
```

an effective condition: none of the three keys is in the set of used secrets

a message sent from A to B in the network

three keys just used are added in the set of used secrets

the time when the message is sent

encrypted content in the intruder's knowledge

Formal Specification of PQ OpenPGP

Threat intruder model

- A generic Dolev-Yao intruder who has the following capabilities:
  1. Intercept messages over the network and extract contents
  2. Generate random components, such as the session key, etc.
  3. Use available information to compute KEM ciphertexts, derive shared secrets, etc.
  4. Apply any cryptographic primitive function to obtain useful information
  5. Forge messages using available data
  6. Compromise secrets, such as long-term private keys, etc.
  7. Exploit quantum computing to break the security of traditional cryptography like ECDH and EdDSA

Formal Specification of PQ OpenPGP

Intruder specification

```
1  op fkMsg    : Sys Prin Prin ECDHK-Cipher MLK-Cipher Data Data
2                -> Sys {constr}
3  op c-fkMsg  : Sys Prin Prin ECDHK-Cipher MLK-Cipher Data Data
4                -> Bool
5
6  eq c-fkMsg(S,A,B,Ecdhk-Ci,Mlk-Ci,KC,C2)      =
7       (Ecdhk-Ci \in knl(S) and
8        Mlk-Ci   \in knl(S) and
9        KC       \in knl(S) and
10       C2       \in knl(S)) .
11
12 ceq nw(fkMsg(S,A,B,Ecdhk-Ci,Mlk-Ci,KC,C2))  =
13      (msg(intru,A,B, Ecdhk-Ci || Mlk-Ci || KC || C2,
14      time(S)) , nw(S)) if c-fkMsg(S,A,B,Ecdhk-Ci,Mlk-Ci,KC,C2) .
15 eq time(fkMsg(S,A,B,Ecdhk-Ci,Mlk-Ci,KC,C2)) = s(time(S)) .
```

encrypted data in the intruder's knowledge

a fake message from the intruder in the network

Formal Specification of PQ OpenPGP

Intruder specification

An intruder has the ECDH public key, then derives the ECDH private key using a large quantum computer.

```
1 op intruBreakECDH              : Sys Prin -> Sys {constr}
2 op c-intruBreakECDH            : Sys Prin -> Bool
3
4 eq c-intruBreakECDH(S,A)       = (ECDHK-PubK(A) \in knl(S)) .
5 ceq knl(intruBreakECDH(S,A)) = (ecdhk-getSecret(ECDHK-PubK(A))
6                                 || knl(S)) if      c-intruBreakECDH(S,A) .
7 ceq intruBreakECDH(S,A)        = S if not c-intruBreakECDH(S,A) .
```

Formal Verification of PQ OpenPGP

Secrecy of session key

- A message sent from A to B,
- B derives KEK, then decrypts to get K
- Not in leak sources:
  - The session key K
  - ML-KEM shared secret by Decaps
  - ML-KEM long-term private key
- K is not in the intruder's knowledge
- Proofs: Seven lemmas required
- Execution time: 1,8 seconds

```
op keySe : Sys Prin Prin Prin SessionKey MLK-Cipher ECDHK-Cipher
           Data Data Nat -> Bool .
eq keySe(S,A2,A,B,K,Mlk-Ci,Ecdhk-Ci,KC,C2,N2) =
(
  not(A = intru or B = intru)                                  and
  msg(A2,A,B, Ecdhk-Ci || Mlk-Ci || KC || C2, N2) \in nw(S)    and
  sdec(kcombine(
    mlk-decaps(Mlk-Ci, MLK-PriK(B))          ||
    ecdhk-decaps(Ecdhk-Ci, ECDHK-PriK(B))    ||
    Ecdhk-Ci                                 ||
    ECDHK-PubK(B)                            ||
    Mlk-Ci                                   ||
    MLK-PubK(B)), KC) = K                                      and
  not K \in leakscr(S)                                         and
  not mlk-decaps(Mlk-Ci, MLK-PriK(B)) \in leakscr(S)           and
  not (MLK-PriK(B) \in' leakscr(S))
) implies (not K \in knl(S)) .
```

Formal Verification of PQ OpenPGP

Forward secrecy of the session key

- Additional conditions:
  - ML-KEM long-term private key in leak sources
  - Only leaked after the message has been sent
- Proofs: Eight lemmas required
- Execution time: 2 seconds

```
op fwdSe : Sys Prin Prin Prin SessionKey MLK-Cipher ECDHK-Cipher
     Data Data Nat
          -> Bool .
eq fwdSe(S,A2,A,B,K,Mlk-Ci,Ecdhk-Ci,KC,C2,N2) =
(
  not(A = intru or B = intru)                                          and
  msg(A2,A,B, Ecdhk-Ci || Mlk-Ci || KC || C2, N2) \in nw(S)            and
  sdec(kcombine(
    mlk-decaps(Mlk-Ci, MLK-PriK(B))          ||
    ecdhk-decaps(Ecdhk-Ci, ECDHK-PriK(B))    ||
    Ecdhk-Ci                                 ||
    ECDHK-PubK(B)                            ||
    Mlk-Ci                                   ||
    MLK-PubK(B)), KC) = K                                              and
  not K \in leakscr(S)                                                 and
  not mlk-decaps(Mlk-Ci, MLK-PriK(B)) \in leakscr(S)                   and
  (MLK-PriK(B) \in' leakscr(S)) implies N2 < timeLeak(MLK-PriK(B), leakscr(S))
) implies (not K \in knl(S)) .
```

Formal Verification of PQ OpenPGP

Authenticity

- Additional conditions:
  - B decrypts C2
  - B verifies two digital signatures
- A has indeed sent the message to B
- Proofs: Five lemmas required
- Execution time: 2,5 seconds

```
op auth : Sys Prin Prin Prin RawMsg SessionKey MLK-Cipher ECDHK-
    Cipher Data Data Data
          Data Nat Nat -> Bool .
eq auth(S,A2,A,B,M,K,Mlk-Ci,Ecdhk-Ci,KC,C2,SIGN1,SIGN2,N2,?T) =
(
  not(A = intru or B = intru)                                    and
  msg(A2,A,B, Ecdhk-Ci || Mlk-Ci || KC || C2, N2) \in nw(S)      and
  sdec(K,C2) = SIGN1 || SIGN2 || M                               and
  MLDSA-Verify(MLDSA-PubK(A), SIGN2, h(M))                       and
  EdDSA-Verify(EdDSA-PubK(A), SIGN1, h(M))                       and
  sdec(kcombine(
    mlk-decaps(Mlk-Ci, MLK-PriK(B))                  ||
    ecdhk-decaps(Ecdhk-Ci, ECDHK-PriK(B))            ||
    Ecdhk-Ci                                         ||
    ECDHK-PubK(B)                                    ||
    Mlk-Ci                                           ||
    MLK-PubK(B)), KC) = K                                        and
  not K \in leakscr(S)                                           and
  not mlk-decaps(Mlk-Ci, MLK-PriK(B)) \in leakscr(S)             and
  not (MLK-PriK(B) \in' leakscr(S))
) implies
  (msg(A,A,B, Ecdhk-Ci || Mlk-Ci || KC || C2, ?T) \in nw(S)) .
```

Formal Verification of PQ OpenPGP

Experimental results

Category	Name	Auxiliaries	Case splitting	Time (ms)
Invariants	keySe (secrecy of session key)	inv{1, 2, 3, 8}	119	1,764
	fwdSe (forward secrecy)	inv{1, 2, 3, 7, 8}	132	2,181
	auth (authenticity)	inv{6}	109	2,497
Lemmas	inv1	inv{8}	72	822
	inv2	inv{4}	87	1,066
	inv3	inv{1, 8, 9}	91	1,616
	inv4	inv{1, 9}	76	892
	inv5	inv{1, 9}	79	944
	inv6	inv{3, 4, 5, 9}	157	8,409
	inv7	no lemma	90	1,069
	inv8	inv{9, 10}	80	944
	inv9	inv{10}	87	1,071
	inv10	no lemma	30	298

Formal Verification of PQ OpenPGP

Challenges

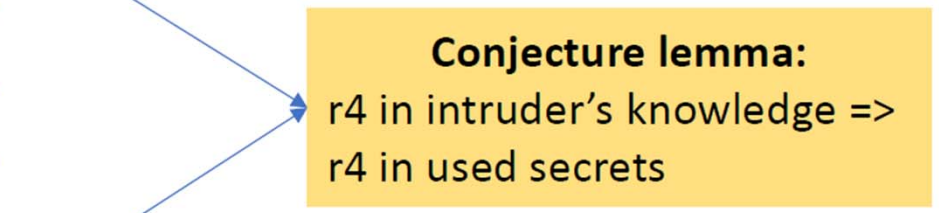
- Understanding the protocol
- Formalizing the protocol
- Validating the specification
- How to discharge a false case?
  - **Finding a conjecture lemma***
  - Prove the lemma
  - If all true, then uses to prove

**A non-trivial task*

```
--> true, use inv1 as a lemma
open INV .
  op a : -> Prin .
  op a2 : -> Prin .
  op b : -> Prin .
  op c2 : -> Data .
  op ecdhk-ci : -> ECDHK-Cipher .
  op ecdhk-sk2 : -> ECDHK-SecretK .
  op k : -> SessionKey .
  op kc : -> Data .
  op mlk-ci : -> MLK-Cipher .
  op mlk-sk2 : -> MLK-SecretK .
  op n2 : -> Nat .
  op r1 : -> Prin .
  op r2 : -> Prin .
  op r3 : -> RawMsg .
  op r4 : -> SessionKey .
  op r5 : -> ECDHK-SecretK .
  op r6 : -> MLK-SecretK .
  op s : -> Sys .
  eq (r4 \in usecret(s)) = false .
  eq (r5 \in usecret(s)) = false .
  eq (r6 \in usecret(s)) = false .
  eq a = r1 .
  eq a2 = r1 .
  eq b = r2 .
  eq (r1 = intru) = false .
  eq (r2 = intru) = false .
  eq c2 = senc(r4,(EdDSA-Sign(EdDSA-PriK(r1),h(r3)) || MLDSA-Sign(MLDSA-PriK(r1),h(r3)) || r3)) .
  eq ecdhk-ci = ecdhk-encapsC(ecdhk-keygen(ECDHK-PriK(r2)),r5) .
  eq kc = senc(kcombine(((MLK-PriK(r2) & r6) || (ECDHK-PriK(r2) & r5) || ecdhk-encapsC(ecdhk-keygen(ECDHK-PriK(r2)),r5) ||
  ecdhk-keygen(ECDHK-PriK(r2)) || mlk-encapsC(mlk-keygen(MLK-PriK(r2)),r6) || mlk-keygen(MLK-PriK(r2))),r4) .
  eq k = r4 .
  eq mlk-ci = mlk-encapsC(mlk-keygen(MLK-PriK(r2)),r6) .
  eq time(s) = n2 .
  eq (r4 \in knl(s)) = true .
  eq (r4 \in leakscr(s)) = false .
  eq (MLK-PriK(r2) \in' leakscr(s)) = false .
  eq ((MLK-PriK(r2) & r6) \in leakscr(s)) = false .
  eq (msg(r1,r1,r2,(ecdhk-encapsC(ecdhk-keygen(ECDHK-PriK(r2)),r5) || mlk-encapsC(mlk-keygen(MLK-PriK(r2)),r6) || senc(kcombine(((MLK-PriK(r2) & r6) ||
  (ECDHK-PriK(r2) & r5) || ecdhk-encapsC(ecdhk-keygen(ECDHK-PriK(r2)),r5) || ecdhk-keygen(ECDHK-PriK(r2)) || mlk-encapsC(mlk-keygen(MLK-PriK(r2)),r6) ||
  mlk-keygen(MLK-PriK(r2))),r4) || senc(r4,(EdDSA-Sign(EdDSA-PriK(r1),h(r3)) || MLDSA-Sign(MLDSA-PriK(r1),h(r3)) || r3)),n2) \in nw(s)) = false .
  red inv1(s,r4) implies (keySe(s,a2,a,b,k,mlk-ci,ecdhk-ci,mlk-sk2,ecdhk-sk2,kc,c2,n2) implies
  keySe(send(s,r1,r2,r3,r4,r5,r6),a2,a,b,k,mlk-ci,ecdhk-ci,mlk-sk2,ecdhk-sk2,kc,c2,n2)) .
close
```

Formal Verification of PQ OpenPGP

```
eq (r4 \in usecret(s))                    = false .
eq (r5 \in usecret(s))                    = false .
eq (r6 \in usecret(s))                    = false .
eq (r1 = intru)                           = false .
eq (r2 = intru)                           = false .
eq (r4 \in knl(s))                        = true .
eq (r4 \in leakscr(s))                    = false .
eq (MLK-PriK(r2) \in' leakscr(s))         = false .
eq ((MLK-PriK(r2) & r6) \in leakscr(s))   = false .
eq (msg(r1,r1,r2,...,n2) \in nw(s))       = false .
```



Trong Binh Hoang: Formal specification and analysis of Post-quantum OpenPGP protocol in CafeOBJ, Master's Thesis, JAIST, September, 2025.

Trong Binh Hoang, Duong Dinh Tran, Canh Minh Do and Kazuhiro Ogata: Formal Specification and Analysis of Post-quantum OpenPGP Protocol in CafeOBJ. 37th International Conference on Software Engineering and Knowledge Engineering (SEKE25), KSI Research Inc., pp.159-145, (2025)

Trong Binh Hoang

Duong Dinh Tran

Canh Minh Do

Formal Verification of PQ SSH

version exchange	VERSION_EX	$C \rightarrow S:$	Version_C
	VERSION_EX	$S \rightarrow C:$	Version_S
key exchange algorithms	KEX_ALGR	$C \rightarrow S:$	Suites_C
	KEX_ALGR	$S \rightarrow C:$	Suites_S
key exchange initiation	KEX_HBR_INIT	$C \rightarrow S:$	$\text{ECDH}_{\text{PK}_C}, \text{KEM}_{\text{PK}_C}$
key exchange reply	KEX_HBR_REPLY	$S \rightarrow C:$	$\text{LK}_S, \text{ECDH}_{\text{PK}_S}, \text{KEM}_{C_S}, \text{SIGN}$

Fig. 3. Messages exchanged in the PQ SSH protocol

Formal Verification of PQ SSH

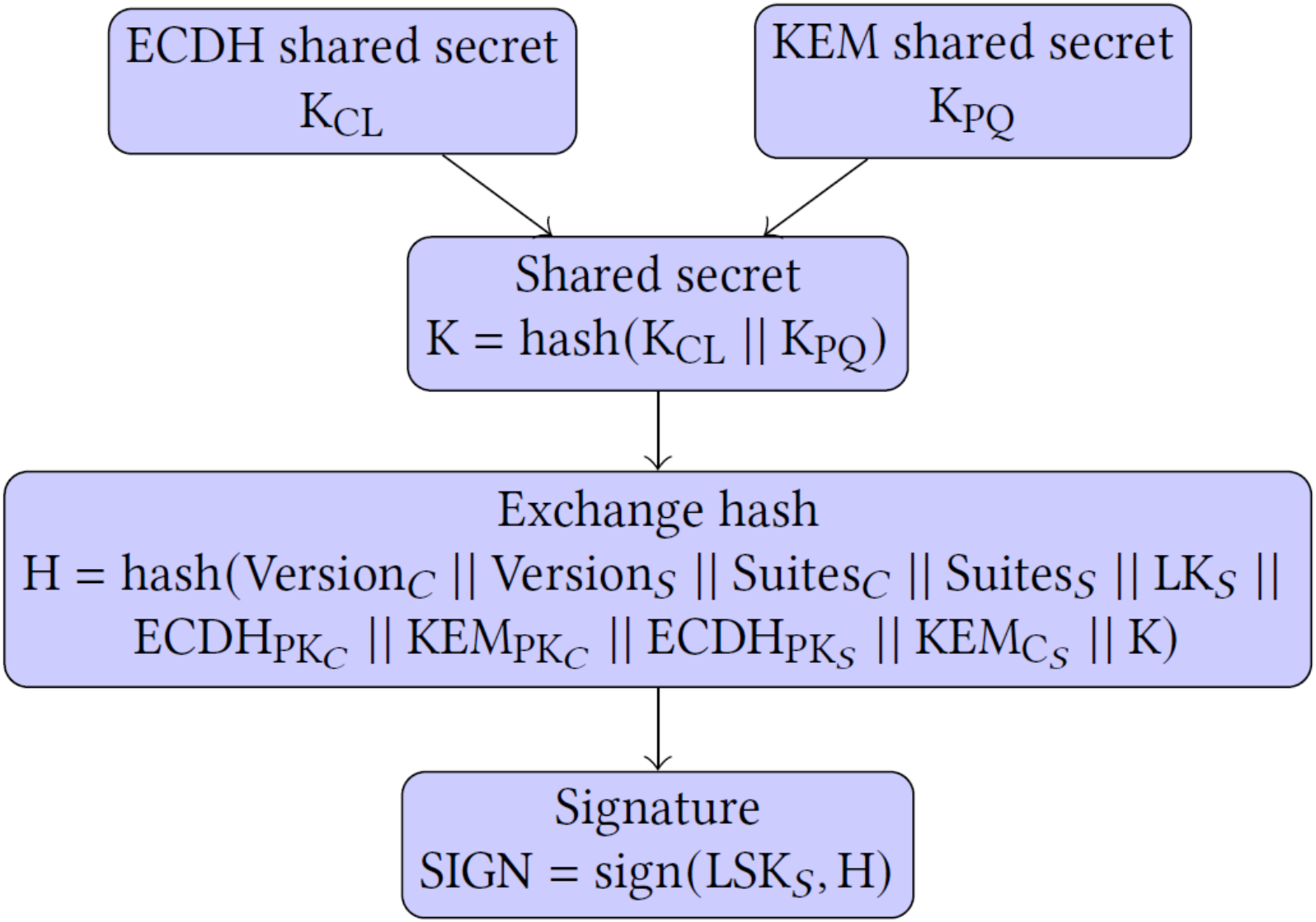


Fig. 4. Exchange hash and signature calculation

Formal Verification of PQ SSH

Step-1	A	$A \rightarrow B$:	$\mathrm{ECDH}_{\mathrm{PK}} \,\|\|\, \mathrm{KEM}_{\mathrm{PK}}$
Step-2	I	learns	$\mathrm{ECDH}_{\mathrm{PK}} \,\|\|\, \mathrm{KEM}_{\mathrm{PK}}$
Step-3	I	$A_2 \rightarrow B$:	$\mathrm{ECDH}_{\mathrm{PK}} \,\|\|\, \mathrm{KEM}_{\mathrm{PK}}$
Step-4	B	$B \rightarrow A_2$:	$\mathrm{LK}_B \,\|\|\, \mathrm{ECDH}_{\mathrm{PK}_2} \,\|\|\, \mathrm{KEM}_{\mathrm{C}} \,\|\|\, \mathrm{SIGN}$
Step-5	I	learns	$\mathrm{LK}_B \,\|\|\, \mathrm{ECDH}_{\mathrm{PK}_2} \,\|\|\, \mathrm{KEM}_{\mathrm{C}} \,\|\|\, \mathrm{SIGN}$
Step-6	I	$B \rightarrow A$:	$\mathrm{LK}_B \,\|\|\, \mathrm{ECDH}_{\mathrm{PK}_2} \,\|\|\, \mathrm{KEM}_{\mathrm{C}} \,\|\|\, \mathrm{SIGN}$

where I denotes the intruder

Fig. 5. Counterexample of auth

Formal Verification of PQ SSH

$$\mathrm{H} = \mathrm{hash}(\mathrm{Version}_C \,||\, \mathrm{Versions}_S \,||\, \mathrm{Suites}_C \,||\, \mathrm{Suites}_S \,||\, \mathrm{LK}_S$$
$$||\, \mathrm{ECDH}_{\mathrm{PK}_C} \,||\, \mathrm{KEM}_{\mathrm{PK}_C} \,||\, \mathrm{ECDH}_{\mathrm{PK}_S} \,||\, \mathrm{KEM}_{\mathrm{C}_S} \,||\, \mathrm{K} \,||\, \mathrm{A} \,||\, \mathrm{B})$$

Client and server identifications added

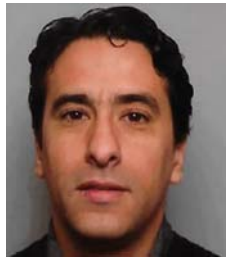
D.D. Tran, K. Ogata, S. Escobar, S. Akleylek, A. Otmani: Formal analysis of Post-Quantum Hybrid Key Exchange SSH Transport Layer Protocol, IEEE Access 12: 1672-1687 (2024)

OGATA Kazuhiro (Japan), Santiago Escobar (Spain), Ayoub Otmani (France), Sedat Akleylek (Turkey): Formal Analysis and Verification of Post-Quantum Cryptographic Protocols (FAVPQC), **ICT for Resilient, Safe and Secure Society, EIG CONCERT-Japan, SICORP, JST**, FY2021 - FY2023

Kazuhiro Ogata (Japan)

Santiago Escobar (Spain)

Sedat Akleylek (Turkey/Estonia)



Ayoub Otmani (France)

Canh Minh Do, Adrian Riesco, Santiago Escobar, Kazuhiro Ogata: Parallel Maude-NPA for Cryptographic Protocol Analysis, IEEE Transactions on Dependable and Secure Computing (IEEE TDSC), Pages 1 – 18, IEEE, 2025. to appear

Canh Minh Do, Tsubasa Takagi, Kazuhiro Ogata: Automated Quantum Protocol Verification Based on Concurrent Dynamic Quantum Logic, ACM Transactions on Software Engineering and Methodology (ACM TOSEM), 34(6): Article No.: 182, 1–36 (2025), ACM, 2025.

D.D. Tran, K. Ogata, S. Escobar, S. Akleylek, A. Otmani: Symbolic verification of Hybrid Post-Quantum TLS 1.2, under review for journal publication

Adrian Riesco, Kazuhiro Ogata, Masaki Nakamura, Daniel Gaina, Duong Dinh Tran, KokichiFutatsug: Proof Scores: A Survey, ACM Computing Surveys (ACM CSUR), 57 (10), Article No.: 251, Pages 1 – 37, ACM, 2025

Some future directions

- A more generic intruder model that can be used for formal verification of other PQ security protocols
- More case studies of PQ protocol formal verification
- Automatic/systematic lemma conjecture
- Making IPSG more scalable

Some future directions

- Formal verification of quantum security protocols in which quantum cryptographic primitives, such as BB84, are used, including an intruder model for it
- To this end, we need to comprehend quantum circuits/protocols/programs better, for which we have been working on formal verification of quantum circuits/protocols/programs

Some future directions

- The following Kaken project has been accepted, where part of the future directions are carried out:

  Logical foundation and formal verification of quantum-resistant security protocols
  (Fostering Joint International Research)
  2024-09-09 – 2028-03-31





Tsubasa Takagi

Acknowledgement

The staff members of the four funding agencies (JST, CNRS, AEI and TUBITAK) and the four universities (JASIT, Polytechnic University of Valencia, University of Rouen Normandie and Ondokuz Mayis University) have always supported FAVPQC.

We appreciate their efforts and time.

Dr. Yuzuru Tanaka, Program Officer, has always encouraged us to conduct good research.

We are grateful to him for his endless encouragement.

Thank you for listening!