A formal analysis of OpenPGP's post-quantum public-key algorithm extension

Duong Dinh Tran[1], Kazuhiro Ogata[1], and Santiago Escobar[2]

[1]Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan

[2]VRAIN, Universitat Politècnica de València, Valencia, Spain

The 2nd International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols

November 21, 2023

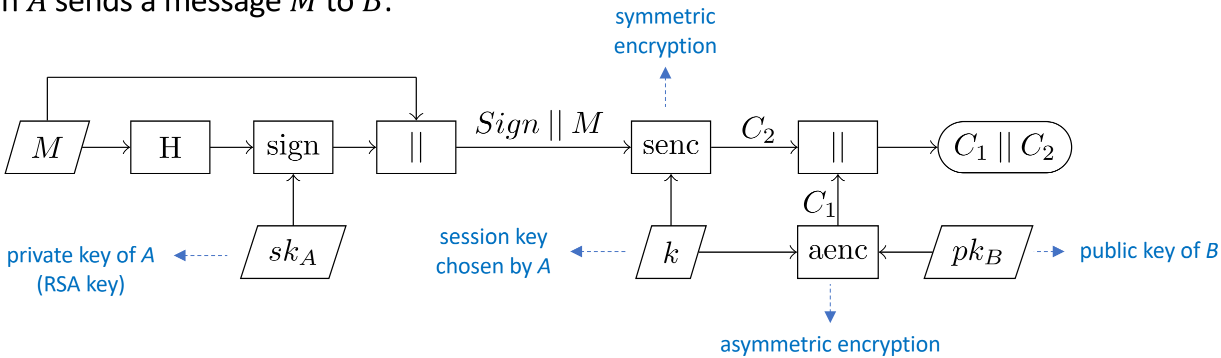
Overview

- OpenPGP
- A post-quantum extension of OpenPGP
- Maude tool
- Modeling PQ OpenPGP in Maude
- Formal analysis
- Summary

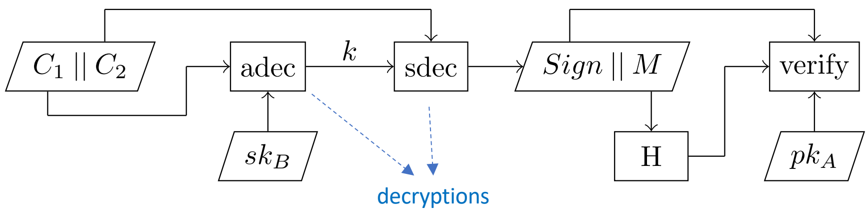
OpenPGP

OpenPGP is an open standard of PGP (Pretty Good Privacy), used for encrypting and decrypting information.

When A sends a message M to B:



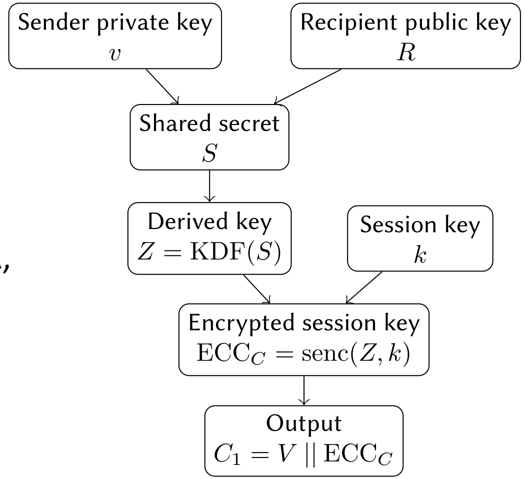
B decrypts the received message:



OpenPGP

- Elliptic Curve Cryptography (ECC) -based key exchange and signature replaced RSA to keep the efficiency.

For example, ECDH is used to encrypt the session key:



- However, all of those public-key algorithms, RSA, ECDH, and ECC-based digital signatures, are threatened by quantum computers.

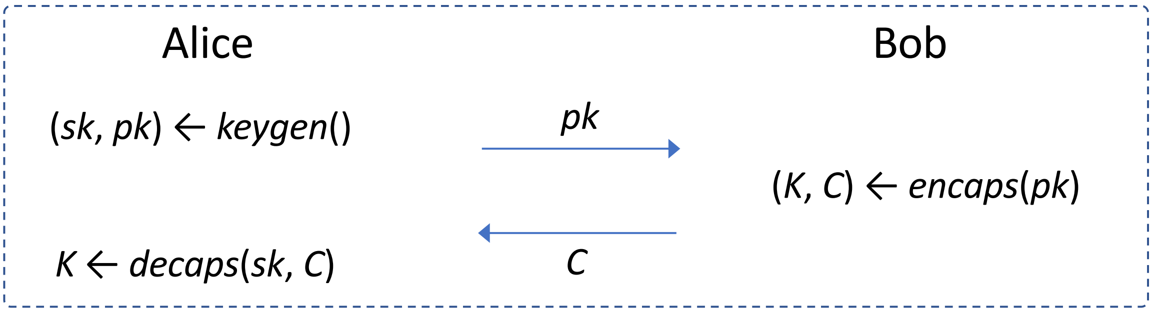
→ A post-quantum extension for the OpenPGP protocol has been proposed and being standardized.

https://datatracker.ietf.org/doc/draft-wussler-openpgp-pqc/02/

Key Encapsulation Mechanism (KEM)

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

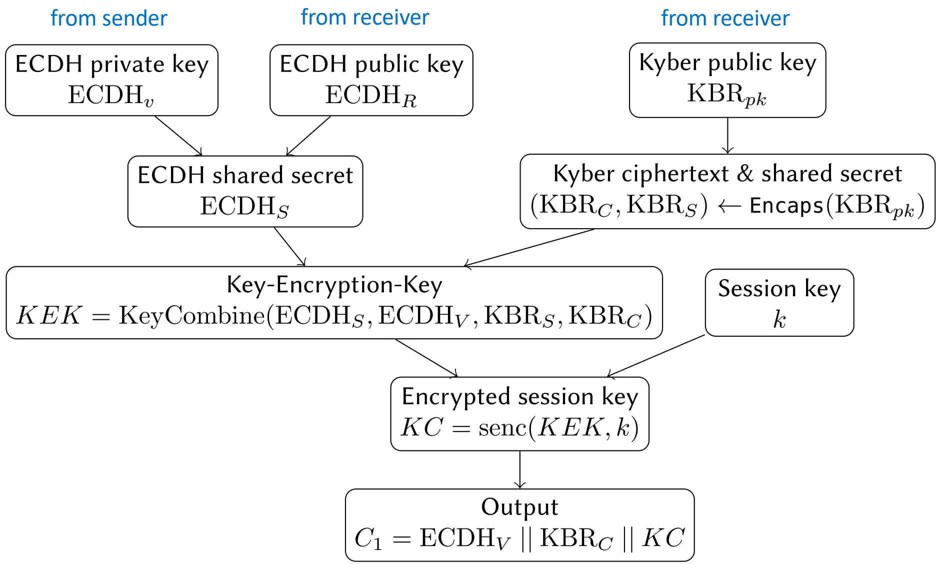
1. (*sk*, *pk*) ← *keygen*(): outputs a public key *pk* and a secret key *sk*.
2. (*K*, *C*) ← *encaps*(*pk*): takes the public key *pk*, and outputs a ciphertext *C* and a shared secret key *K*.
3. *K* ← *decaps*(*sk*, *C*): takes the secret key *sk*, a ciphertext *C* and outputs the shared secret key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



Post-quantum extension of OpenPGP

- PQ OpenPGP uses:
  1) ECDH in combination with CRYSTALS-Kyber, a post-quantum KEM,
  2) ECC-based signature algorithm with CRYSTALS-Dilithium, a post-quantum digital signature.

1) Composite algorithm with ECDH and CRYSTALS-Kyber to encrypt a session key k:



Maude

- A declarative language and high-performance tool.
- Can be used to formalize a system/protocol as a state machine.
- A functional module:

```
fmod M is
  Σ
  E
endfm
```

signature

equation set

```
fmod SIMPLE-NAT is
  sorts     Zero NzNat   Nat .
  subsorts Zero NzNat < Nat .
  op 0 :       -> Zero  [ctor] .
  op s : Nat -> NzNat [ctor] .
  op _+_ : Nat Nat -> Nat .
  vars N N' : Nat .
  eq 0 + N = N .
  eq s(N) + N' = s(N + N') .
endfm
```

- A system module: we can also declare a rewrite rule (specify state transitions)

crl [label] : l => r **if** c_1 /\ c_2 /\ ... .

If the condition c_1 /\ c_2 /\ ... holds under some substitution σ, $\sigma(l)$ can be replaced with $\sigma(r)$.

Modeling the protocol in Maude

1. Model cryptographic primitives used, such as ECDH, CRYSTALS-Kyber.
2. Specify the protocol execution.
3. Specify the threat model.

```
fmod ECDH is
  ---       public keys,   private keys,   shared secrets
  sorts   EdPubKey     EdPriKey      EdShareS .
  --- derivation of the associated public key from a private key
  op pk : EdPriKey -> EdPubKey .
  --- computation of shared secret from a public and a private keys
  op ss : EdPubKey EdPriKey -> EdShareS .
  --- constructor of a shared secret is a private key pair
  op _|_ : EdPriKey EdPriKey -> EdShareS [comm] .
  vars SK SK2 : EdPriKey .
  eq ss(pk(SK), SK2) = SK | SK2 .
endfm
```

Modeling CRYSTALS-Kyber

```
sorts KbPubKey KbPriKey KbShareS KbCipher .
--- KeyGen is a probabilistic algorithm ,
--- so keygen takes a private key as input and returns the public key
op keygen  : KbPriKey -> KbPubKey .
--- similarly, Encaps is probabilistic, so an argument of KbPriKey is added
op encapsC : KbPubKey KbPriKey -> KbCipher .   --- returns ciphertext
op encapsK : KbPubKey KbPriKey -> KbShareS .   --- returns shared secret
op decaps  : KbCipher KbPriKey -> KbShareS .
--- constructor of a shared secret is a private key pair
op _&_     : KbPriKey KbPriKey -> KbShareS .
vars SK SK2 : KbPriKey .
eq encapsK(keygen(SK), SK2) = (SK & SK2) .
eq decaps(encapsC(keygen(SK), SK2), SK) = (SK & SK2) .
```

Other primitives

```
--- generic sorts of all other sorts
sorts  Data DataL .
--- some other sorts in ...
subsorts  EdPubKey EdPriKey KbPubKey KbPriKey . . . <  Data .
subsort   Data  <  DataL .
--- concatenation
op _||_  :  DataL  DataL  ->  DataL [assoc ctor id: nilDL] .
op h     :  DataL          ->  Data .    --- hash function
. . .
```

Protocol execution

The protocol is modeled as a state machine, where each state is an AC-collection of name-value pairs, i.e., *observable components* in Maude. Some *observable components* used are:

- (ecdh[A]: < PK ; SK >): User A has an ECDH public/private key pair PK and SK.
- (kyber[A]: < PK ; SK >): User A has a Kyber KEM public/private key pair PK and SK.
- (nw: MS): The network, i.e., collection of messages exchanged, is MS.
- (e-knl: (D_1 ; D_2 ; …)): The intruder’s knowledge is (D_1 ; D_2 ; …).
- . . .

We define an initial state with the participation of two honest users together with a dishonest user (the Dolev-Yao intruder).

Protocol execution: Encrypt and send a message

```
crl [send] :
   {(ms: (M MS)) (rd-sesskey: (K KS))
    (ecsig[A]: (< PKES ; SKES > , SKS))  (dilit[A]: (< PKDI ; SKDI > , SKS2))
    (ecdh[B] : (< PKED ; SKED > , SKS3)) (ecdh[A] : (< PKED2 ; SKED2 > , SKS4))
    (kyber[B]: (< PKKB ; SKKB > , SKS5)) (kyber[A] : (< PKKB2 ; SKKB2 > , SKS6))
    (nw: NW) (e-knl: DS) (used-kyber[A]: SKS7) OCs}
=> {(ms: MS)     (rd-sesskey: KS)
    (ecsig[A]: (< PKES ; SKES > , SKS))  (dilit[A]: (< PKDI ; SKDI > , SKS2))
    (ecdh[B] : (< PKED ; SKED > , SKS3)) (ecdh[A] : (SKS4))
    (kyber[B]: (< PKKB ; SKKB > , SKS5)) (kyber[A] : (SKS6))
    (nw: (msg(A,B, PKED2 || KBC || KC || C2) NW))
    (e-knl: (DS ; PKED2 ; KBC ; KC ; C2))
    (used-kyber[A]: (SKS7 , < PKKB2 ; SKKB2 >)) OCs}
if H    := h(M) /\                                                1. hash message
   SIGN := ecSign(SKES,H)  /\  SIGN2 := diSign(SKDI,H) /\         2. sign
   EDSS := ss(PKED,SKED2)  /\  KBSS := encapsK(PKKB,SKKB2) /\     3. compute shared secrets
   KBC  := encapsC(PKKB,SKKB2) /\                                 4. compute KEM ciphertext
   KEK  := kcombine(EDSS,PKED2,KBSS,KBC) /\                       5. compute key encryption key
   KC   := senc(KEK, K) /\                                        6. encrypt session key
   C2   := senc(K, SIGN || SIGN2 || M) .                          7. final ciphertext
```

Threat model

We suppose the presence of an intruder with the following capabilities:

1) intercept any message sent in network to learn information in that message.
2) generate random components, such as, the session key.
3) apply any cryptographic primitive function to derive new information from the information learned.
4) have access to quantum computers, so that can break the security of ECDH and ECC-based signature schemes.

Intruder specification

1) intercept any message sent in network to learn information in that message.

```
crl [send] :
   {(nw: NW)
    (e-knl: DS)
    . . .  OCs}
=> {(nw: (msg(A,B, PKED2 || KBC || KC || C2)  NW))
    (e-knl: (DS ; PKED2 ; KBC ; KC ; C2))
    . . .  OCs}
```

public information

Intruder specification

3) apply any cryptographic primitive function to derive new information from the information learned.

```
--- (e-knl: (M ; DS)) says that M, a raw message, is in the intruder's knowledge
--- intruder can hash M and learn the result, i.e., h(M)
rl [hash] :
   {(e-knl :  (M ; DS))  OCs}   =>   {(e-knl :  (M ; DS ; h(M)))  OCs} .
--- intruder can compute Kyber KEM shared secret and encapsulation by Encaps
--- PKKB and SKKB are variables of Kyber public and private keys
crl [encaps] :
     {(e-knl :  (PKKB ; SKKB ; DS))  OCs}
=>  {(e-knl :  (PKKB ; SKKB ; DS ; encapsC(PKKB, SKKB) ; encapsK(PKKB, SKKB)))  OCs}
if  PKKB  =/=  keygen(SKKB) .
```

Intruder specification

4) have access to quantum computers, so that can break the security of ECDH and ECC-based signature schemes.

```
--- breaking ECDH, Eve can derive private keys from public keys
rl [break-ecdh] :
   {(e-knl: (pk(SKED) ; DS)) OCs}    =>  {(e-knl: (pk(SKED) ; DS ; SKED)) OCs} .
--- breaking ECC-based signature schemes
rl [break-ecc-sign] :
   {(e-knl: (pkes(SKES) ; DS)) OCs}  =>  {(e-knl: (pkes(SKES) ; DS ; SKES)) OCs} .
```

Analysis: Secrecy of messages

```
search [1,10] in PQOPENPGP : init =>*
 {(ecsig[A] : (< PKES ; SKES > , SKS))    (dilit[A]   : (< PKDI ; SKDI > ,  SKS2))
  (ecdh[B] : (< PKED ; SKED > , SKS3)) (kyber[B]: (< PKKB ; SKKB > , SKS5))
  (nw    :  (msg(A,B, PKED2 || KBC || KC || C2)  NW))
  (e-knl :  (M ; DS))  OCs}
such that
  (A =/= eve  and  B =/= eve) /\
  EDSS := ss(PKED2, SKED)   /\    KBSS := decaps(KBC, SKKB) /\
  KEK   := kcombine(EDSS, PKED2, KBSS, KBC) /\
  K       := sdec(KEK, KC) /\
  SIGN || SIGN2 || M  :=  sdec(K, C2) /\
  ecVerify(PKES, SIGN, h(M))  /\  diVerify(PKDI, SIGN2, h(M)) .
```

1. compute shared secrets

2. compute key encryption key

3. decrypt session key

4. decrypt the message

5. verify the two signatures

Search a state with bounded depth 10 in which:

- there exists an encrypted message sent from A to B,
- B decrypts the raw message M and successfully verifies the composite signatures,
- M exists in Eve's knowledge.

Maude did not find such a state after 1m39s, the protocol enjoys the property up to depth 10.

Analysis: experimental results

We also consider two other properties:

- *Secrecy of ECDH shared secrets*: Experiment shows that the intruder can learn such secrets.
- *Authenticity of messages*: if Bob decrypts an encrypted message apparently sent from Alice and successfully verifies the composite signatures with Alice's verifying keys, obtaining a raw message M, then Alice indeed sent M to Bob.

Property	Depth	Result	Time (h:m:s)	No. States
Secrecy of messages	8	∅	0:00:6.7	46317
	9	∅	0:00:22.2	98943
	10	∅	0:01:39	206972
	11	∅	0:08:31	430750
	12	∅	0:42:34	903344
	13	∅	5:08:16	1929731
Authenticity of messages	8	∅	0:06:40	46317
	9	∅	0:23:39	98943
	10	∅	1:26:30	206972
	11	∅	6:54:14	430750

Table 1
Experimental results. ∅ means that Maude did not find solution(s) for the given search command.

Summary

- We have presented a formal analysis of the OpenPGP's post-quantum extension.
- The experimental results have confirmed that the protocol enjoys two properties: *secrecy of messages* and *authenticity of messages* up to some specific depths.

Limitations:

- The number of the state space generated is huge. We were unable to proceed with the experiments at deeper depths due to time limitations.

A possible future work:

- Verification based on interactive theorem proving.

Thank you for your attention!

Experiments: Time difference

	Property	Depth	Result	Time (h:m:s)	No. States
(1)	Secrecy of messages	8	∅	0:00:6.7	46317
		9	∅	0:00:22.2	98943
		10	∅	0:01:39	206972
		11	∅	0:08:31	430750
		12	∅	0:42:34	903344
		13	∅	5:08:16	1929731
(2)	Authenticity of messages	8	∅	0:06:40	46317
		9	∅	0:23:39	98943
		10	∅	1:26:30	206972
		11	∅	6:54:14	430750

With same depth, checking (1) is significantly faster than checking (2) mostly because:

In each state, there very huge number of substitutions for the following pattern in the **search** command of (2):

(e-knl: (PKED2 ; KBC ; KC ; C2 ; DS))

where KC and C2 are variables of the sort Data, and so they can be substituted by any terms of Data or its subsorts.

Note also that ; is AC, making the number of substitution solutions increase.