Hybrid Post-Quantum TLS formal specification in Maude-NPA - toward its security analysis

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

[1]Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan
[2]VRAIN, Universitat Politècnica de València, Valencia, Spain

International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols 2022

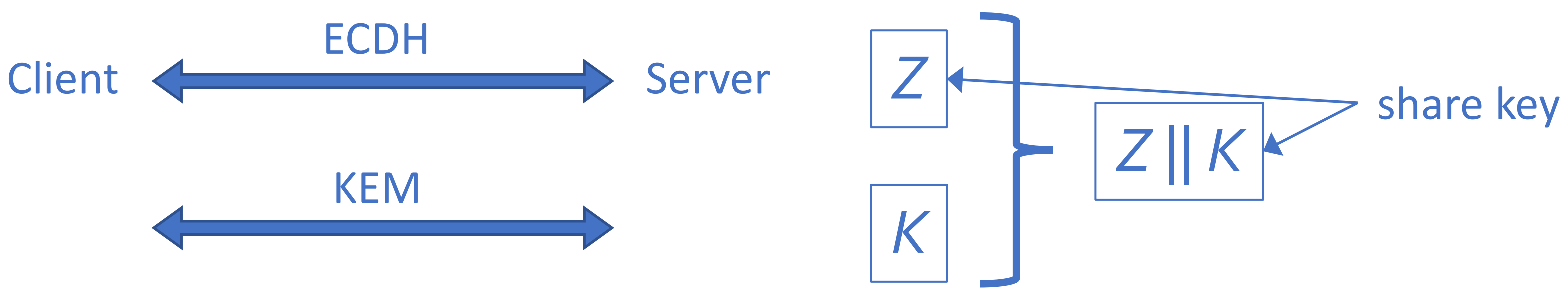
October 24, 2022

Overview

- Introduction
- Hybrid Post-Quantum TLS
- Maude-NPA formal specification by strands
- Hybrid PQ TLS formal specification
- Analysis experiments
- Summary

Introduction

- As a response to the quantum attack threat, AWS has proposed the Hybrid Post-Quantum (PQ) Transport Layer Security (TLS) Protocol*, a quantum-resistant version of TLS 1.2.
- The “hybrid” terminology refers to the hybrid key exchange mode used in the protocol:
  - a conventional key exchange algorithm, fixed as Elliptic Curve Diffie-Hellman (ECDH), and
  - a post-quantum key encapsulation mechanism (KEM), e.g., Kyber.



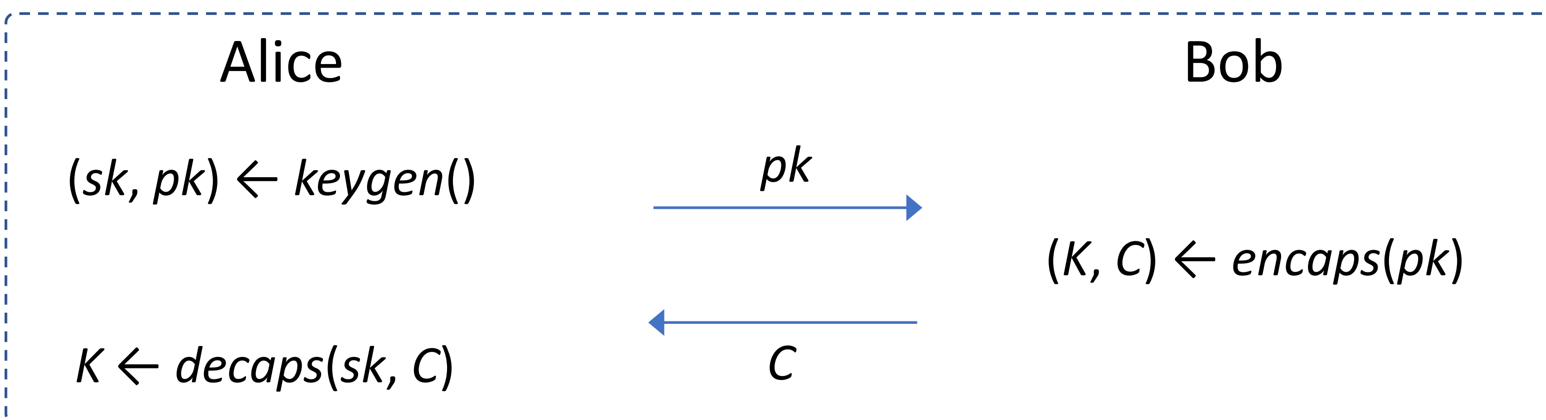
→ Hybrid PQ TLS: Formal specification and analysis with Maude-NPA

*https://datatracker.ietf.org/doc/html/draft-campagna-tls-bike-sike-hybrid

Key Encapsulation Mechanism (KEM)

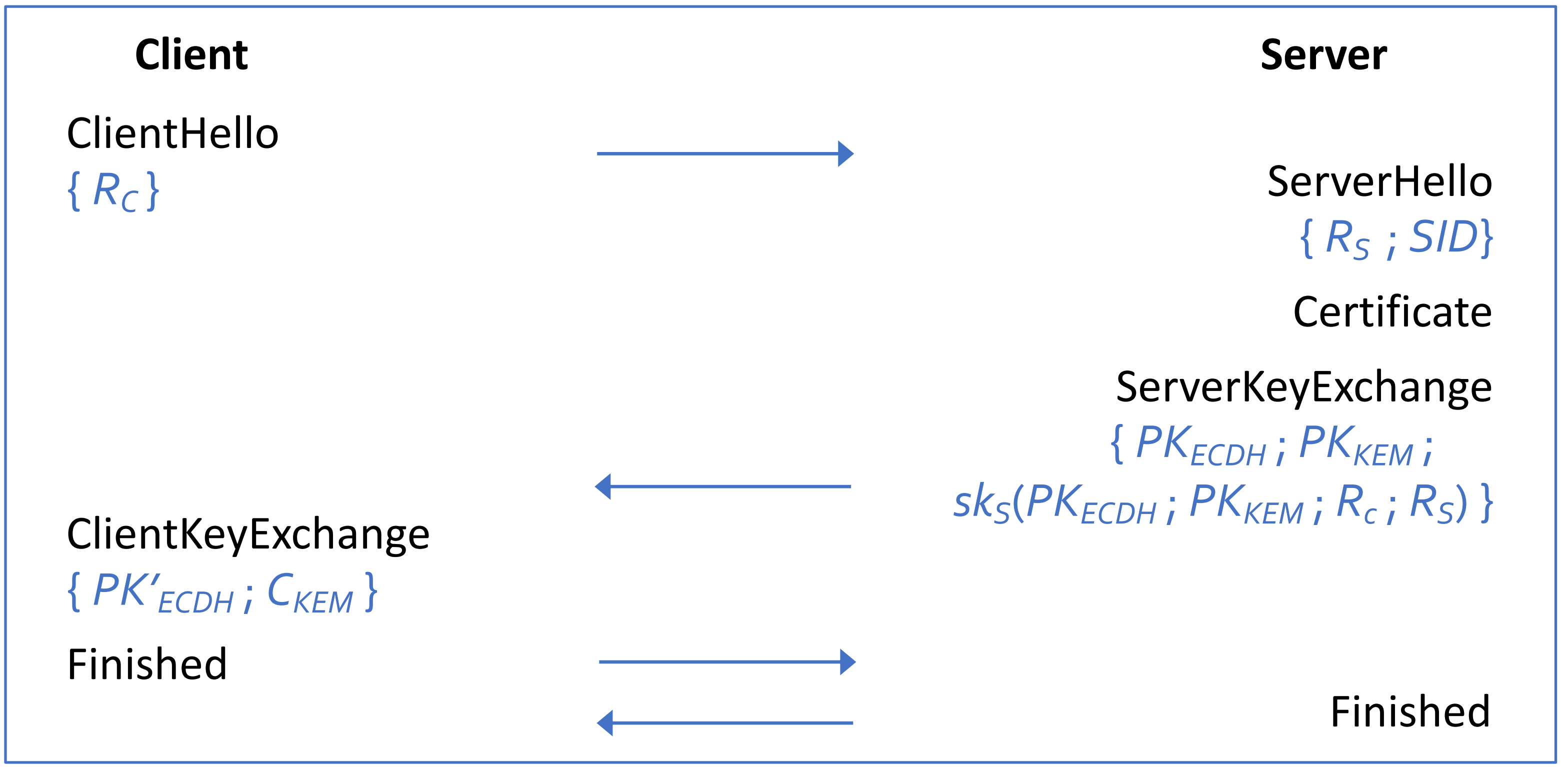
A KEM is a tuple of algorithms (*keygen*, *encaps*, *decaps*):

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*.



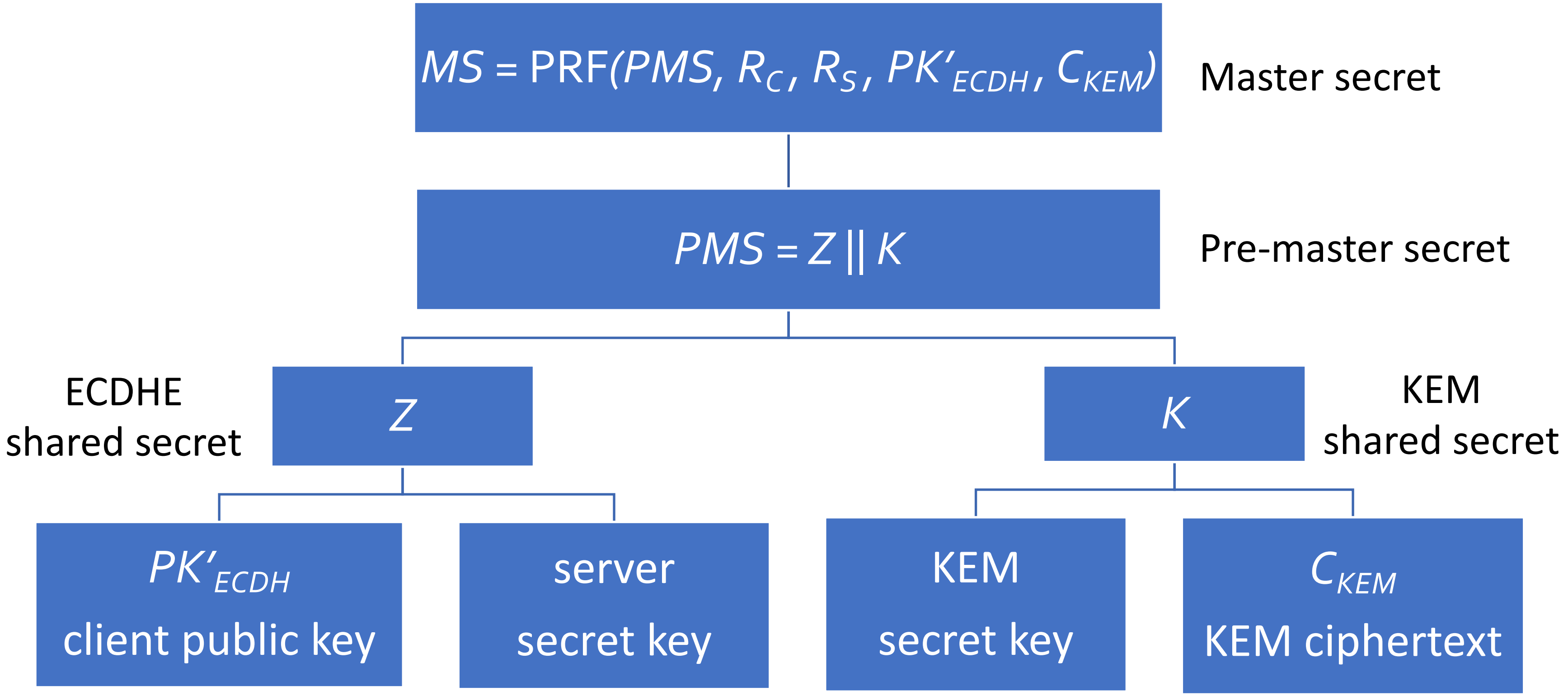
Hybrid PQ TLS protocol

We consider a simplified version of the Hybrid PQ TLS as follows:



- $sk_S(PK_{ECDH} ; PK_{KEM} ; R_c ; R_S)$: encrypt the plaintext by the private key of S.
- Finished message: hash of all messages before encrypted by the handshake key negotiated, whose calculation is explained in the next slide.

Key calculation



- PRF: Pseudorandom function
- We use the master secret as the symmetric key for encryption of the Finished messages

Maude-NPA formal specification by strands

A strand is in the following form:

$$:: r_1, \ldots, r_k :: [+(m_1), -(m_2), \ldots, -(m_i) \mid +(m_{i+1}), \ldots]$$

- $r_1, \ldots, r_k$ denote unique freshes generated in the strand.
- $+(m)$ and $-(m)$ denote sending and receiving the message m, respectively.
- messages appearing before "|" were sent/received in the past.
- messages appearing after "|" will be sent/received in the future.

Let's consider the Needham-Schroeder Public Key (NSPK) protocol:

i) $A \rightarrow B : pk(B, A ; N_A)$

ii) $B \rightarrow A : pk(A, N_A ; N_B)$

iii) $A \rightarrow B : pk(B, N_B)$

; denotes the concatenation

N_A: unique nonce generated by A

$pk(A, m)$: m encrypted by A's public key

Maude-NPA formal specification by strands

```
--- principal names and nonces
sorts      Name Nonce .
subsort    Name Nonce < Msg .

--- a nonce is in form of n(A,r), where r (of sort Nonce) guarantees its uniqueness
op n     : Name Fresh    -> Nonce [frozen] .

--- public & private encryption
op pk    : Name Msg      -> Msg    [frozen] .
op sk    : Name Msg      -> Msg    [frozen] .
```

Strand specifying the protocol execution from the A side:

```
:: r :: [ nil | +(pk(B, A ; n(A,r))), -(pk(A, n(A,r) ; N)), +(pk(B, N)), nil ]
```

Strand specifying the protocol execution from the B side:

```
:: r :: [ nil | -(pk(B, A ; N)), +(pk(A, N ; n(B,r))), -(pk(B, n(B,r))), nil ]
```

Hybrid PQ TLS specification: KEMs

```
--- public keys, secret keys, encapsulations, and shared keys
sorts     PqPk PqSk Cipher PqKey
subsort   PqPk PqSk Cipher PqKey < Msg .

op pqSk     : Name Fresh  -> PqSk  [frozen] .
op pqPk     : PqSk        -> PqPk  [frozen] .
op $pqKey   : PqSk PqSk   -> PqKey [frozen] .

op encapCipher  : PqPk PqSk -> Cipher [frozen] .     --- encaps
op encapKey     : PqPk PqSk -> PqKey  [frozen] .     --- encaps

op decap : Cipher PqSk -> PqKey [frozen] .           --- decaps

--- algebraic properties of KEMs
eq pqKey(pqPk(S:PqSk), S2:PqSk) = $pqKey(S:PqSk, S2:PqSk) [variant] .
eq decap(encapCipher(pqPk(S:PqSk), S2:PqSk), S:PqSk)
                            = $pqKey(S:PqSk, S2:PqSk) [variant] .
```

Hybrid PQ TLS specification: ECDH

```
--- points (on the curve), scalars, and ECDH shared keys
sorts     Point Scalar ECKey .
subsort   Point < ECKey .

op p    : -> Point .                           --- the point generator
op sk   : Name Fresh -> Scalar [frozen] .      --- a unique scalar serving as a secret key

--- gen(p, s) produces a public key pk (to send to the opposite peer)
--- gen(pk, s) produces a shared key (where pk is received from the opposite peer)
op gen : Point Scalar -> Point [frozen] .

--- multiplication on scalars.
op _*_ : Scalar Scalar -> Scalar [frozen assoc comm] .

--- algebraic properties of ECDH
eq gen(gen(P:Point, K1:Scalar), K2:Scalar)
     = gen(P:Point, K1:Scalar * K2:Scalar) [variant] .
```

Hybrid PQ TLS specification

```
--- Pre-Master Secrets & Master Secrets
op pms   : ECKey PqKey -> PreMasterSecret [frozen] .
op ms    : PreMasterSecret Rand Rand ECKey Cipher -> MasterSecret [frozen] .

--- signature, certificates, randoms, and session IDs
op sig   : Name Msg     -> Msg        [frozen] .
op cert  : Name         -> Cert       [frozen] .
op rd    : Name Fresh   -> Rand       [frozen] .
op sess  : Name Fresh   -> Session    [frozen] .

--- encryption & decryption
op enc   : MasterSecret Msg  -> Msg [frozen] .
op dec   : MasterSecret Msg  -> Msg [frozen] .
```

Protocol execution specification: client side

```
eq STRANDS-PROTOCOL
= :: r1,r2,r3 ::
[ nil |
+(ch ; rd(C,r1)),
-(sh ; N ; SS),
-(sc ; cert(S)),
-(ske ; PK1 ; PK2 ; sig(S, PK1 ; PK2 ; rd(C,r1) ; N)),
+(cke ; gen(p, sk(C,r2)) ; cipher(PK2, pqSk(C,r3))),
+(cf ; enc(ms(pms(gen(PK1, sk(C,r2)), pqKey(PK2, pqSk(C,r3)) ),
          rd(C,r1), N, gen(p,sk(C,r2)), cipher(PK2, pqSk(C,r3)) ),
     (ch ; rd(C,r1)) ++
     (sh ; N ; SS) ++
     (sc ; cert(S)) ++
     (ske ; PK1 ; PK2 ; sig(S, PK1 ; PK2 ; rd(C,r1) ; N)) ++
     (cke ; gen(p,sk(C,r2)) ; cipher(PK2, pqSk(C,r3)) ) )
 ),
. . .
```

Protocol execution specification: server side

```
:: r1,r2,r3,r4 ::
[ nil |
-(ch ; N),
+(sh ; rd(S,r1) ; sess(S,r2)),
+(sc ; cert(S)),
+(ske ; gen(p,sk(S,r3)) ; pqPk(pqSk(S,r4)) ;
     sig(S, gen(p,sk(S,r3)) ; pqPk(pqSk(S,r4)) ; N ; rd(S,r1))),
-(cke ; PK1 ; CP),
-(cf ; enc(ms(pms(gen(PK1,sk(S,r3)), decap(CP, pqSk(S,r4)) ),
         N, rd(S,r1), PK1, CP ),
    (ch ; N) ++
    (sh ; rd(S,r1) ; sess(S,r2)) ++
    (sc ; cert(S)) ++
    (ske ; gen(p,sk(S,r3)) ; pqPk(pqSk(S,r4)) ;
         sig(S, gen(p,sk(S,r3)) ; pqPk(pqSk(S,r4)) ; N ; rd(S,r1))) ++
    (cke ; PK1 ; CP) )
 ), . . .
```

Intruder capabilities

```
--- messages concatenation & deconcatenation
:: nil :: [ nil | -(M1 ; M2), +(M1), nil ] &
:: nil :: [ nil | -(M1 ; M2), +(M2), nil ] &
:: nil :: [ nil | -(M1), -(M2), +(M1 ; M2), nil ]
--- generate any random number and any point
:: r :: [ nil | +(rd(i,r)), nil ] &
:: r :: [ nil | +(sk(i,r)), nil ]
--- with KEMs
:: nil :: [ nil | -(PK2), -(SK), +(encapCipher(PK2,SK)), nil ] &
:: nil :: [ nil | -(PK2), -(SK), +(encapKey(PK2,SK)), nil ] &
:: nil :: [ nil | -(CP), -(SK), +(decap(CP,SK)), nil ]
--- with ECDH, we suppose that the intruder can break
--- the key exchange security by utilizing the power of quantum computers
:: nil :: [ nil | -(gen(p,K1)), -(gen(p,K2)), +(gen(p,K1 * K2)), nil ]
```

Analysis: learning ECDH shared key

```
eq ATTACK-STATE(0)
= :: r1,r2,r3 ::
[ nil,
+(ch ; rd(c,r1)),
-(sh ; rd(s,r1') ; sess(s,r')),
-(sc ; cert(s)),
-(ske  ; gen(p, sk(s,r2')) ; pqPk(pqSk(s,r3')) ; . . . ),
+(cke ; gen(p, sk(c,r2)) ; cipher(pqPk(pqSk(s,r3')), pqSk(c,r3)) ),
+(cf ; . . .),
-(sf ; . . .) | nil ]
|| gen(p, sk(s,r2') * sk(c,r2))  inI,  empty              --- ECDH shared secret key
|| nil
|| nil
|| nil
[nonexec] .
```

Analysis: secrecy property

```
eq ATTACK-STATE(1)
= :: r1,r2,r3 ::
[ nil,
+(ch ; rd(c,r1)),
-(sh ; rd(s,r1') ; sess(s,r')),
-(sc ; cert(s)),
-(ske ; gen(p,sk(s,r2')) ; pqPk(pqSk(s,r3')) ; . . . ),
+(cke ; gen(p,sk(c,r2)) ; cipher(pqPk(pqSk(s,r3')), pqSk(c,r3)) ),
+(cf ; . . .),
-(sf ; . . .) | nil ]
|| $pqKey(pqSk(s,r3'), pqSk(c,r3))  inI,  empty       --- PQ KEM shared secret key
|| nil
|| nil
|| nil
[nonexec] .
```

Analysis: experiment results

ATTACK-STATE	Result/Bound	Maude-NPA (h:m:s)	Par-Maude-NPA (h:m:s)	Percentage of improvement
0	✗/12	18:23:01	05:35:06	69.6 %
1	✓/12	06:24:29	02:00:06	68.8 %

Thank you for your attention!