Formal specification and model checking of lattice-based key encapsulation mechanisms in Maude

Duong Dinh Tran[1], Kazuhiro Ogata[1], Santiago Escobar[2], Sedat Akleylek[3] and Ayoub Otmani[4]

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

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

[3] *Ondokuz Mayis University, Turkey*

[4] *University of Rouen Normandie, France*

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

October 24, 2022

Overview

- Motivation
- Key encapsulation mechanism and Kyber
- Modeling Kyber in Maude
- The property and the counterexample
- Summary

Motivation

- Shor algorithm can efficiently solve hard mathematical problems on which the current public-key cryptography is relying.
- Sufficient large-scale quantum computers can efficiently "break" the current public-key algorithms, such as RSA, Diffie-Hellman.

→ A threat: Lots of sensitive information transmitted over the Internet, such as emails, passwords, credit card numbers will not be secure anymore.

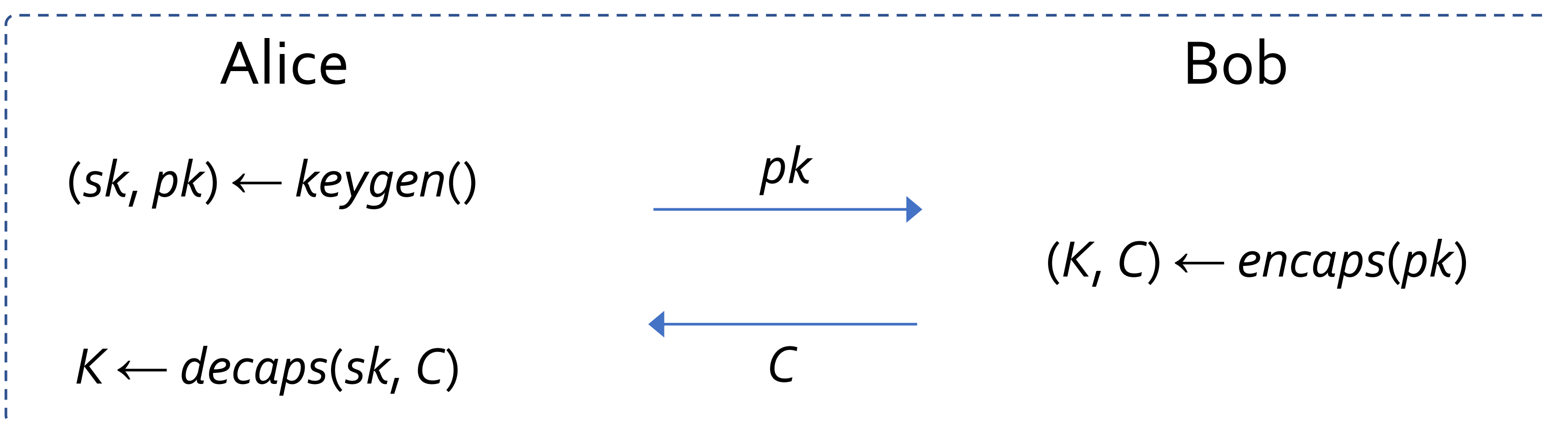
- Large numbers of post-quantum Key Encapsulation Mechanisms (KEMs) have been proposed to provide secure key establishment.

→ Formal specification and model checking lattice-based KEMs

Key Encapsulation Mechanism (KEM)

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

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



Kyber KEM

KEM.KeyGen()

$z \leftarrow B^{32}$
$(pk, sk') = \text{PKE.KeyGen}()$
$sk = (sk' || pk || H(pk) || z)$
return (pk, sk)

pk →

KEM.Enc(pk)

$m_0 \leftarrow B^{32};\ m \leftarrow H(m_0)$
$(\bar{K}, \bar{r}) = G(m || H(pk))$
$c = \text{PKE.Enc}(pk, m, \bar{r})$
$K = KDF(\bar{K} || H(c))$
return (K, c)

← c

KEM.Dec(c, sk)

$m' = \text{PKE.Dec}(\boldsymbol{s}, c)$
$(\bar{K}', \bar{r}') = G(m' || H(pk))$
$c' = \text{PKE.Enc}(pk, m', \bar{r}')$
if $c = c'$ **then** $K = KDF(\bar{K}' || H(c))$
else return $K = KDF(z || H(c'))$
return K

- ▷ H, G: hash functions
- ▷ B^{32}: the set of 32-length byte arrays
- ▷ KDF: key derive function
- ▷ bold lower(upper)-case: vectors (matrices)

Kyber

PKE.KeyGen()

$d \leftarrow B^{32}$

$(\rho, \sigma) = G(d)$

$\boldsymbol{A} \in R_q^{k \times k}$: generated from ρ

$\boldsymbol{s} \in R_q^{k}$: sampled from σ

$\boldsymbol{e} \in R_q^{k}$: sampled from σ

$\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$

return $(pk \coloneqq \boldsymbol{t}||\rho, sk \coloneqq \boldsymbol{s})$

KEM.KeyGen()

$z \leftarrow B^{32}$

$(pk, sk') =$ PKE.KeyGen()

$sk = (sk'||pk||H(pk)||z)$

return (pk, sk)

$\xrightarrow{\quad pk \quad}$

KEM.Enc(pk)

$m_0 \leftarrow B^{32}; m \leftarrow H(m_0)$

$(\bar{K}, \bar{r}) = G(m||H(pk))$

$c =$ PKE.Enc$(pk, m, \bar{r})$

$K = KDF(\bar{K}||H(c))$

return (K, c)

$\xleftarrow{\quad c \quad}$

KEM.Dec(c, sk)

$m =$ PKE.Dec$(\boldsymbol{s}, c)$

$(\bar{K}', \bar{r}') = G(m'||H(pk))$

$c' =$ PKE.Enc$(pk, m', \bar{r}')$

if $c = c'$ **then** $K = KDF(\bar{K}'||H(c))$

else return $K = KDF(z||H(c'))$

return K

- ▷ R_q: the quotient polynomial ring $Z_q[X]/(X^n + 1)$
- ▷ R_q^k: k-dimension vectors of R_q
- ▷ bold lower(upper)-case: vectors (matrices)

Kyber

KEM.KeyGen()

$z \leftarrow B^{32}$
$(pk, sk') = \text{PKE.KeyGen()}$
$sk = (sk' || pk || H(pk) || z)$
return (pk, sk)

$\xrightarrow{pk}$

KEM.Enc($pk \coloneqq \boldsymbol{t} || \rho$)

$m_0 \leftarrow B^{32}; m \leftarrow H(m_0)$
$(\bar{K}, \bar{r}) = G(m || H(pk))$
$c = \text{PKE.Enc}(pk, m, \bar{r})$
$K = KDF(\bar{K} || H(c))$
return (K, c)

$\xleftarrow{c}$

KEM.Dec(c, sk)

$m = \text{PKE.Dec}(\boldsymbol{s}, c)$
$(\bar{K}', \bar{r}') = G(m' || H(pk))$
$c' = \text{PKE.Enc}(pk, m', \bar{r}')$
if $c = c'$ **then** $K = KDF(\bar{K}' || H(c))$
else return $K = KDF(z || H(c'))$
return K

PKE.Enc($pk \coloneqq \boldsymbol{t} || \rho, m, \bar{r}$)

$\boldsymbol{A} \in R_q^{k \times k}$: re-generated from ρ
$\boldsymbol{r} \in R_q^k$: sampled from $\bar{r}$
$\boldsymbol{e_1} \in R_q^k$: sampled from $\bar{r}$
$e_2 \in R_q$: sampled from $\bar{r}$
$\boldsymbol{u} = \boldsymbol{A}^T \boldsymbol{r} + \boldsymbol{e_1}$
$v = \boldsymbol{t}^T \boldsymbol{r} + e_2 + \text{Decompress}(m, 1)$
$c_1 = \text{Compress}(\boldsymbol{u}, d_u)$
$c_2 = \text{Compress}(v, d_v)$
return $c \coloneqq (c_1 || c_2)$

- ▷ $x \in \mathbb{Z}_q; d < \text{Celling}[\log_2 q]$
- ▷ $\text{Compress}(x, d) = Round\left[\frac{2^d}{q} x\right] mod\ 2^d$
- ▷ $\text{Decompress}(x, d) = Round\left[\frac{q}{2^d} x\right]$

Kyber

KEM.KeyGen()

$z \leftarrow B^{32}$

$(pk, sk') =$ PKE.KeyGen()

$sk = (sk' || pk || H(pk) || z)$

return (pk, sk)

pk →

KEM.Enc($pk := \boldsymbol{t} || \rho$)

$m_0 \leftarrow B^{32}; m \leftarrow H(m_0)$

$(\bar{K}, \bar{r}) = G(m || H(pk))$

$c =$ PKE.Enc($pk, m, \bar{r}$)

$K = KDF(\bar{K} || H(c))$

return (K, c)

← c

KEM.Dec($c := c_1 || c_2, sk$)

$m' =$ PKE.Dec($\boldsymbol{s}, c$)

$(\bar{K}', \bar{r}') = G(m' || H(pk))$

$c' =$ PKE.Enc($pk, m', \bar{r}'$)

if $c = c'$ **then** $K = KDF(\bar{K}' || H(c))$

else return $K = KDF(z || H(c'))$

return K

PKE.Dec($\boldsymbol{s}, c := c_1 || c_2$)

$\boldsymbol{u} = \text{Decompress}(c_1, d_u)$

$v = \text{Decompress}(c_2, d_v)$

$m' := \text{Compress}(v - \boldsymbol{s}^T \boldsymbol{u}, 1)$

return m'

Formal specification: polynomials

```
fmod POLYNOMIAL is
   pr INT .        sort Poly .        subsort Int < Poly .
   op _p+_  : Poly Poly -> Poly [ctor assoc comm prec 33] .     --- addition
   op _p*_  : Poly Poly -> Poly [ctor assoc comm prec 31] .     --- multiplication
   op _p-_  : Poly Poly -> Poly [prec 33] .                     --- subtraction
   op neg_  : Poly      -> Poly [ctor] .                        --- negation
   vars P0 P1 P2 P3 : Poly .
   eq P1 p+ 0 = P1 .    eq P1 p* 0 = 0 .
   eq P1 p* 1 = P1 .
   eq P1 p* (P2 p+ P3) = (P1 p* P2) p+ (P1 p* P3) .
   eq P1 p- P2 = P1 p+ neg(P2) .
   eq P1 p+ neg(P1) = 0 .
   eq neg(neg(P1)) = P1 .
   eq neg(P1 p+ P2) = neg(P1) p+ neg(P2) .
endfm
```

Formal specification: name-value pairs

Each state is modeled as an AC-collection of the following name-value pairs:

- (prins : ps) - principals participating in the protocol;
- (nw : $msgs$) – AC-collection of messages exchange;
- (d[i] : d_0) - d_0 is the random seed d (used in **PKE.KeyGen**()) of principal i;
- (rd-d : $rdds$) - list of fresh values for the random seed d, i.e., an entry in $rdds$ is extracted when a principal needs to generate a random value of d;
- …

The initial state is defined as:

```
eq init =
    { (prins: (alice bob eve)) (d[alice]: 0) (d[bob]: 0) (nw: empty) (rd-d: (d1 , d2)) … } .
```

Three rewrite rules keygen, encaps, and decaps are defined specifying the three corresponding Kyber KEM algorithms.

Formal specification: honest parties

```
op msg1 : Prin    Prin    Prin         Vector  Poly   MsgState -> Msg [ctor] .
```

- Prin → actual creator
- Prin → seeming sender
- Prin → receiver
- Vector → $\boldsymbol{t}$
- Poly → ρ
- MsgState →
  - sent: message was sent
  - replied: message was sent and the receiver replied with another message
  - intercepted: message was intercepted by the intruder

```
crl [keygen] : { (rd-d: (D, PoL))   (d[A]: P1)   (prins: (A B PS))   (nw: MS) Ocs }
=> { (rd-d: PoL)   (d[A]: D)   (prins: (A B PS))
      (nw: (msg1(A, A, B, sample-A(Rho) m* sample-s(Sig) v+ sample-e(Sig),
                              Rho, sent)  MS)) OCs }
if Rho := 1st(G(D))  /\  Sig := 2nd(G(D)) .
```

- Rho ↔ ρ
- Sig ↔ σ

sample-A(Rho) m* sample-s(Sig) v+ sample-e(Sig) ↔ $\boldsymbol{As} + \boldsymbol{e} = \boldsymbol{t}$

- sample-A(Rho) ↔ $\boldsymbol{A} \in R_q^{k \times k}$
- sample-s(Sig) ↔ $\boldsymbol{s} \in R_q^{k}$
- sample-e(Sig) ↔ $\boldsymbol{e} \in R_q^{k}$

Formal specification: attacker model

We use the standard Dolev-Yao intruder model, that is a generic intruder, namely eve, can completely control the network. In particular, the intruder can:

- intercept any message and glean the data sent in that message, such as the values of $\boldsymbol{t}$, ρ, c_1, and c_2;
- randomly choose their private/random values, such as d and m, then build by themself $\boldsymbol{s}$, $\boldsymbol{e}$, $\boldsymbol{r}$, $\boldsymbol{e_1}$, and e_2;
- use such information above to fake some messages, impersonate some honest parties to send the messages to some other ones.

Formal specification: intruder capabilities

5 rewrite rules specify the intruder capabilities:

- keygen-eve: intruder intercepts a 1st message sent from A → B, fakes and sends a new 1st message to B.
- encaps-eve: after intercepting the 1st message sent from A → B, intruder fakes and sends a 2nd message to A, and computes a shared secret key with A.
- decaps-eve : intruder intercepts a 2nd message replied from B → A, and computes a shared secret key with B.
- build-ds: intruder builds the random d.
- build-ms: intruder builds the random m_0.

Formal specification: intruder capabilities

When there exists a message msg1 sent from A to B in the network, the intruder can intercept that message, fake a new message, and send it to B:

```
crl [keygen-eve] :
    { (ds: (D PoC1))  (nw: (msg1(A, A, B, TA, RhoA, sent)  MS))  OCs}
=> { (ds: (D PoC1))  (nw: (msg1(A, A, B, TA, RhoA, intercepted)
       msg1(eve, A, B,  sample-A(Rho) m* sample-s(Sig) v+ sample-e(Sig),
              Rho, sent)  MS))   OCs}
if Rho := 1st(G(D))  /\  Sig := 2nd(G(D)) .
```

For the random seed D, the intruder can only construct it by randomly choosing a fresh value:

```
rl [build-ds] :
     { (rd-d: (D, PoL))   (ds: PoC1)       OCs}
=>  { (rd-d: PoL)          (ds: (PoC1 D))   OCs} .
```

Kyber – MITM attack

```
search [1] in KYBER :
init =>* {(keys[alice]: key(K1,bob))   (keys[bob]: key(K2,alice))
          (glean-keys: (key(K1,alice)   key(K2,bob) KS))
           OCs} .
```

- (keys[alice]: key(K1,bob)) : key that Alice obtained after she communicated (in her belief) with Bob,
- (keys[bob]: key(K2,alice)) : key that Bob obtained after he communicated (in his belief) with Alice,
- (glean-keys: (key(K1,alice) key(K2,bob) KS)) : collection of keys gleaned by intruder.

Kyber – MITM attack

Alice		**Eve**		**Bob**
$d \leftarrow B^{32}$ $(\rho, \sigma) = G(d)$ $\boldsymbol{A} \leftarrow \rho;\ \boldsymbol{s}, \boldsymbol{e} \leftarrow \sigma$ $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$	$pk \coloneqq \boldsymbol{t} \| \rho$ →			
		$d' \leftarrow B^{32}$ $(\rho', \sigma') = G(d')$ $\boldsymbol{A}' \leftarrow \rho';\ \boldsymbol{s}', \boldsymbol{e}' \leftarrow \sigma'$ $\boldsymbol{t}' = \boldsymbol{A}'\boldsymbol{s}' + \boldsymbol{e}'$	$pk' \coloneqq \boldsymbol{t}' \| \rho'$ →	
			← c	$m_0 \leftarrow B^{32};\ m = H(m_0)$ $(\bar{K}, \bar{r}) = G(m \| H(pk'))$ $c =$ PKE.Enc$(pk', m, \bar{r})$ $K_{\mathrm{B}} = KDF(\bar{K} \| H(c))$
		$m =$ PKE.Dec$(\boldsymbol{s}', c)$ $(\bar{K}, \bar{r}) = G(m \| H(pk'))$ $K_{\mathrm{B}} = KDF(\bar{K} \| H(c))$		
	← c'	$m'_0 \leftarrow B^{32};\ m' = H(m'_0)$ $(\bar{K}', \bar{r}') = G(m' \| H(pk))$ $c' =$ PKE.Enc$(pk, m', \bar{r}')$ $K_{\mathrm{A}} = KDF(\bar{K}' \| H(c'))$		
$m' =$ PKE.Dec$(\boldsymbol{s}, c')$ $(\bar{K}', \bar{r}') = G(m' \| H(pk))$ $K_{\mathrm{A}} = KDF(\bar{K}' \| H(c'))$				

Summary

- We must emphasize that: this kind of attack is not a novel attack since Kyber is not equipped with any feature for dealing with authentication.
- We instead illustrates one symbolic approach for reasoning about KEMs rather than focusing on this kind of attack.
- In the same manner, we also conducted model checking with two other KEMs: Saber and MLWR.
- The same kind of attack was found.
- Future work: security analysis of post-quantum cryptographic protocols, which use post-quantum cryptographic primitives, such as KEMs.

Thank you for your attention!