

# KPQC 공모전 1 라운드 격자기반 알고리즘 안전성 분석

2023.11.13.

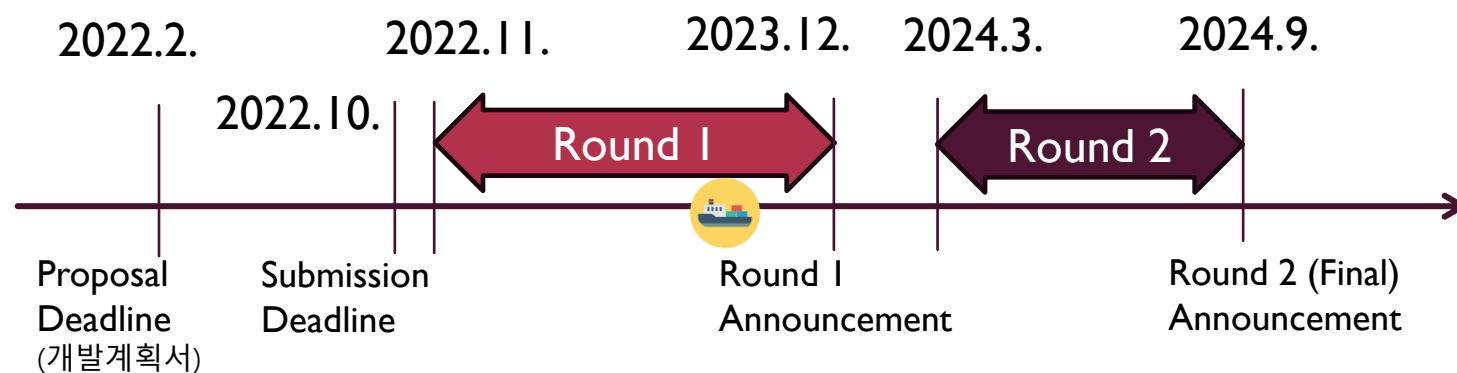
Sungshin Women's University

Joohee Lee

# CONTENTS

- KpqC Round I – Lattice-based Schemes (Summary)
- CCA Attack for NTRU+
- May's Meet-LWE Attack Costs for Lattice-based KEMs
- Security Evaluation of {LWE, LWR}-based schemes Using Lattice Estimator

# KPQC COMPETITION



- 16 algorithms in Round 1
  - 7 KEMs & 9 Signatures
- KpqC Bulletin : <https://groups.google.com/g/kpqc-bulletin>
  - Analysis reports
  - Benchmarks
  - Scheme Updates
  - Etc.

# KPQC ROUND 1 –LATTICE-BASED SCHEMES

- Among Round 1 candidates, 3 KEMs and 5 signatures are lattice-based schemes.

Category	Name	Base problem	Note
KEM	NTRU+	NTRU/RLWE	<ul style="list-style-type: none"><li>• RLWE with binary secrets/ternary errors</li><li>• <a href="#">Analysis Reported (6/14/23)</a></li></ul>
	SMAUG	MLWE/MLWR	<ul style="list-style-type: none"><li>• MLWE/MLWR with sparse secrets</li></ul>
	TiGER	RLWR/RLWE	<ul style="list-style-type: none"><li>• RLWR/RLWE with sparse secrets</li><li>• <a href="#">Analysis Reported (7/9/23)</a></li></ul>
Signature	GCKSign	GCK	<ul style="list-style-type: none"><li>• <a href="#">Analysis Reported (1/14/23)</a></li></ul>
	HAETAE	MLWE/MSIS	
	NCC-Sign	RLWE/RSIS	
	Peregrine	NTRU/SIS	<ul style="list-style-type: none"><li>• <a href="#">Analysis Reported (1/6/23)</a></li></ul>
	SOLMAE	NTRU/SIS	



# CCA ATTACK FOR NTRU+

# RESULTS

Game OW-CCA	$O_{dec}(c)$
1: $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$	1: <b>if</b> $c = c^*$
2: $(K^*, c^*) \leftarrow \text{Encaps}(pk)$	2: <b>return</b> $\perp$
3: $K' \leftarrow \mathcal{A}^{O_{dec}(\cdot)}(pk, c^*)$	3: <b>else return</b>
4: <b>return</b> $[K' = K^*]$	4: $K \leftarrow \text{Decaps}(sk, c)$

## \*OW-CCA SECURITY GAME

- Reported on 6/14/23, in the KpqC Bulletin
  - [Analysis of NTRU+ \(google.com\)](#), eprint:[A Novel CCA Attack for NTRU+ KEM \(iacr.org\)](#)
- For NTRU+ (CCA ver.), we can achieve the challenge encapsulated key  $K^*$ , and **win the CCA game** with 4 decapsulation queries in average.
  - It breaks the **OW-CCA** security and hence **NTRU+** is **\*not\*** **IND-CCA** secure.
- It can be fixed by adding a verification process in the decoding algorithm(“Inv”) to check if the intermediate value  $M + u_2$  (or the output) is binary and abort otherwise.
- We found some flaws in the security proof, some of which introduced our attack.

# SUMMARY ON NTRU+ KEM

- Security based on the **NTRU**, **RLWE** assumptions

- RLWE here uses (random) binary secrets and ternary errors

- Uses **NTT-friendly rings**

- $R_q = \mathbb{Z}_q[x]/(f(x))$ , where  $f(x) = x^n - x^{n/2} + 1$  and  $n = 2^i 3^j$  [1,2]

- Uses a new encoding named **SOTP** (Semi-generalized One Time Pad)

- In the CCA-secure KEM, they **remove re-encryption** in decapsulation by adjusting Fujisaki-Okamoto transform

- [1] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019 (<https://tches.iacr.org/index.php/TCHES/article/view/8293>).
- [2] Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, Gregor Seiler, and Dominique Unruh. A thorough treatment of highly-efficient NTRU instantiations. Public-Key Cryptography – PKC 2023 (<https://eprint.iacr.org/2021/1352>).

Parameters	Security level	n	q	Sizes (Bytes)			Cycles(ref)			Cycles(AVX2)			Claimed Security (bits)	
				pk	ct	sk	Keygen	Encaps	Decaps	Keygen	Encaps	Decaps	Classical	Quantum
NTRU+576	1	576	3,457	864	864	1,728	321,405	110,754	163,277	17,440	14,307	12,445	115	104
NTRU+768	1	768	3,457	1,152	1,152	2,304	313,669	145,658	227,028	16,032	17,514	15,848	164	148
NTRU+864	3	864	3,457	1,296	1,296	2,592	339,912	169,634	262,017	14,068	19,293	17,671	188	171
NTRU+1152	5	1,152	3,457	1,728	1,728	3,456	905,131	230,448	348,076	42,993	25,592	24,063	264	240

# CCA-NTRU+

- Use a new encoding **SOTP** defined by :
  - $m \in \{0,1\}^n, u = (u_1, u_2) \in \{0,1\}^{2n}$ 
    - $SOTP(m, u = (u_1, u_2)) := (m \oplus u_1) - u_2 \in \{-1,0,1\}^n$
    - $Inv(M \in \{-1,0,1\}^n, u = (u_1, u_2)) := (M + u_2) \oplus u_1 \in \{0,1\}^n$

Gen( $1^\lambda$ )

- 1:  $\mathbf{f}', \mathbf{g} \leftarrow \psi_1^n$
- 2:  $\mathbf{f} = 3\mathbf{f}' + 1$
- 3: **if**  $\mathbf{f}, \mathbf{g}$  are not invertible in  $R_q$
- 4:   restart
- 5: **return**  $(pk, sk) = (\mathbf{h} = 3\mathbf{g}\mathbf{f}^{-1}, \mathbf{f})$

Encap( $pk$ )

- 1:  $m \leftarrow \{0,1\}^n$
- 2:  $(\mathbf{r}, K) = H(m)$
- 3:  $\mathbf{m} = \text{SOTP}(m, G(\mathbf{r}))$
- 4:  $\mathbf{c} = \mathbf{h}\mathbf{r} + \mathbf{m}$
- 5: **return**  $(\mathbf{c}, K)$

Decap( $sk, \mathbf{c}$ )

- 1:  $\mathbf{m} = (\mathbf{c}\mathbf{f} \bmod^{\pm q}) \bmod^{\pm 3}$
- 2:  $\mathbf{r} = (\mathbf{c} - \mathbf{m})\mathbf{h}^{-1}$
- 3:  $m = \text{Inv}(\mathbf{m}, G(\mathbf{r}))$
- 4:  $(\mathbf{r}', K) = H(m)$
- 5: **if**  $\mathbf{r} = \mathbf{r}'$
- 6:   **return**  $K$
- 7: **else**
- 8:   **return**  $\perp$

Figure 15: CCA-NTRU+



# CCA-NTRU+

- Use a new encoding **SOTP** defined by :

- $m \in \{0,1\}^n, u = (u_1, u_2) \in \{0,1\}^{2n}$ 
  - $SOTP(m, u = (u_1, u_2)) := (m \oplus u_1) - u_2 \in \{-1,0,1\}^n$
  - $Inv(M \in \{-1,0,1\}^n, u = (u_1, u_2)) := (M + u_2) \oplus u_1 \in \{0,1\}^n$

**Caution!**  $M + u_2$  has to be binary (if not, they make the result binary by computing  $\&0x1$ )

Gen( $1^\lambda$ )

- 1:  $\mathbf{f}', \mathbf{g} \leftarrow \psi_1^n$
- 2:  $\mathbf{f} = 3\mathbf{f}' + 1$
- 3: **if**  $\mathbf{f}, \mathbf{g}$  are not invertible in  $R_q$
- 4:   restart
- 5: **return**  $(pk, sk) = (\mathbf{h} = 3\mathbf{g}\mathbf{f}^{-1}, \mathbf{f})$

Encap( $pk$ )

- 1:  $m \leftarrow \{0,1\}^n$
- 2:  $(\mathbf{r}, K) = H(m)$
- 3:  $\mathbf{m} = \text{SOTP}(m, G(\mathbf{r}))$
- 4:  $\mathbf{c} = \mathbf{h}\mathbf{r} + \mathbf{m}$
- 5: **return**  $(\mathbf{c}, K)$

Decap( $sk, \mathbf{c}$ )

- 1:  $\mathbf{m} = (\mathbf{c}\mathbf{f} \bmod^{\pm q}) \bmod^{\pm 3}$
- 2:  $\mathbf{r} = (\mathbf{c} - \mathbf{m})\mathbf{h}^{-1}$
- 3:  $m = \text{Inv}(\mathbf{m}, G(\mathbf{r}))$
- 4:  $(\mathbf{r}', K) = H(m)$
- 5: **if**  $\mathbf{r} = \mathbf{r}'$
- 6:   **return**  $K$
- 7: **else**
- 8:   **return**  $\perp$

# CCA-NTRU+

- Use a new encoding SOTP defined by :
  - $m \in \{0,1\}^n, u = (u_1, u_2) \in \{0,1\}^{2n}$ 
    - $SOTP(m, u = (u_1, u_2)) := (m \oplus u_1) - u_2 \in \{-1,0,1\}^n$
    - $Inv(M \in \{-1,0,1\}^n, u = (u_1, u_2)) := (M + u_2) \oplus u_1 \in \{0,1\}^n$

**Caution!**  $M + u_2$  has to be binary (if not, they make the result binary by computing  $\&0x1$ )

```
void poly_sotp_inv(unsigned char *msg, const poly *e, const unsigned char *buf)
{
    uint32_t t1, t2, t3;

    for(int i = 0; i < 2; i++)
    {
        for(int j = 0; j < 8; j++)
        {
            t1 = load32_littleendian(msg[32*i + 4*j]);
            t2 = load32_littleendian(msg[32*i + 4*j + 1]);
            t3 = 0;

            for (int k = 0; k < 2; k++)
            {
                for(int l = 0; l < 16; l++)
                {
                    t3 ^= (((e->coeffs[256*i + 16*l + 2*j + k] + t2)^t1) & 0x1) << (l+16*k);

                    t1 >>= 1;
                    t2 >>= 1;
                }

                msg[32*i + 4*j] = t3;
                msg[32*i + 4*j + 1] = t3 >> 8;
                msg[32*i + 4*j + 2] = t3 >> 16;
            }
        }
    }
}
```

## Algorithm 5 Inv

**Require:** Polynomial  $y \in R_q$

**Require:** Byte array  $B = (b_0, b_1, \dots, b_{n/4-1})$

**Ensure:** Message Byte array  $m = (m_0, m_1, \dots, m_{31})$

1:  $(\beta_0, \dots, \beta_{2n-1}) := \text{BytesToBits}(B)$

2: **for**  $i$  from 0 to  $n-1$  **do**

3:  $m_i := ((f_i + \beta_{i+n} \& 1) \oplus \beta_i)$

$m = \text{BitsToBytes}((m_0, \dots, m_{n-1}))$

4: **return**  $m$

## Gen( $1^\lambda$ )

- 1:  $f', g \leftarrow \psi_1^n$
- 2:  $f = 3f' + 1$
- 3: **if**  $f, g$  are not invertible in  $R_q$
- 4: restart
- 5: **return**  $(pk, sk) = (h = 3gf^{-1}, f)$

## Encap( $pk$ )

- 1:  $m \leftarrow \{0,1\}^n$
- 2:  $(r, K) = H(m)$
- 3:  $m = SOTP(m, G(r))$
- 4:  $c = hr + m$
- 5: **return**  $(c, K)$

## Decap( $sk, c$ )

- 1:  $m = (cf \mod^{\pm q}) \mod^{\pm 3}$
- 2:  $r = (c - m)h^{-1}$
- 3:  $m = Inv(m, G(r))$
- 4:  $(r', K) = H(m)$
- 5: **if**  $r = r'$
- 6: **return**  $K$
- 7: **else**
- 8: **return**  $\perp$

# ATTACK FOR CCA-NTRU+

- **Step 1.** find an example of malicious  $M' \in \{-1,0,1\}^n$  such that an intermediate value  $M' + u_2$  in  $Inv(M', u)$  is non-binary ;

- Example (n=4):

Suppose  $m=(1,0,1,1)$ ,  $G(r) = u=(1,1,0,1,1,0,1,0)$

$$\begin{aligned} SOTP(m, G(r)) &= (m \oplus u_1) - u_2 \\ &= ((1,0,1,1) \oplus (1,1,0,1)) - (1,0,1,0) \\ &= (0,1,1,0) - (1,0,1,0) \\ &= (-1,1,0,0) := M \end{aligned}$$

Let  $M' := M + (2, 0, 0, 0) = (1, 1, 0, 0)$ . Then,

$$\begin{aligned} Inv(M', G(r)) &= (M' + u_2) \oplus u_1 \\ &= ((1,1,0,0) + (1,0,1,0)) \oplus (1,1,0,1) \\ &= (2,1,1,0) \oplus (1,1,0,1) \\ &= (3,0,1,1) \rightarrow (1,0,1,1) = m \end{aligned}$$

&0x1

# ATTACK FOR CCA-NTRU+

- **Step 2.** use Step 1 to construct a malicious ciphertext  $c'$  from a challenge ciphertext  $c^* = h \cdot r^* + M^*$  in the CCA security game ;
  - Assume  $c^* = h \cdot r^* + M^*$   
 where  $m^* = (1, 0, 1, 1)$ ,  $G(r^*) = u = (1, 1, 0, 1, 1, 0, 1, 0)$ ,  $M^* = SOTP(m^*, G(r^*))$
  - We set  $c' := c^* + (2, 0, 0, 0) = h \cdot r^* + M'$   
 then  $Decaps(sk, c')$  successfully produces the secret key  $K^*$  which is a decapsulation result of  $c^*$  ( $\because Inv(M', G(r^*)) = m^*$ ).
    - i.e., we can ask decapsulation oracle to achieve  $K^*$
  - But, in the CCA security game, since we don't know both  $M^*$  and  $G(r^*)$  used in the challenge ciphertext, we need to guess :
    - the 0-th bits of  $M^*$  and  $G(r^*)$  should be one of the four cases.

- When i) happens and we add  $(2, 0, 0, 0)$  to  $c^*$ , decapsulation fails (since it produces different  $r$ )
- ii) happens with probability  $1/4$

	$M^* [0]$	$G(r^*) [0]$	$M^* + G(r^*) [0]$
i)	1	0	1
	0	1	1
	0	0	0
ii)	-1	1	0

# ATTACK FOR CCA-NTRU+

- **Step 2.** use Step 1 to construct a malicious ciphertext  $c'$  from a challenge ciphertext  $c^* = h \cdot r^* + M^*$  in the CCA security game ;
  - Assume  $c^* = h \cdot r^* + M^*$   
where  $m^* = (1, 0, 1, 1)$ ,  $G(r^*) = u = (1, 1, 0, 1, 1, 0, 1, 0)$ ,  $M^* = SOTP(m^*, G(r^*))$
  - We set  $c' := c^* + (2, 0, 0, 0) = h \cdot r^* + M'$   
then  $Decaps(sk, c')$  successfully produces the secret key  $K^*$  which is a decapsulation result of  $c^*$  ( $\because Inv(M', G(r^*)) = m^*$ ).
    - i.e., we can ask decapsulation oracle to achieve  $K^*$
  - But, in the CCA security game, since we don't know both  $M^*$  and  $G(r^*)$  used in the challenge ciphertext, we need to guess with probability 1/4 for each component
  - With 4 decapsulation queries in average, we can achieve  $K^*$ , and **win the CCA game**

# OW-CCA SECURITY GAME & ATTACK ALGORITHM

Game OW-CCA	$O_{dec}(c)$
1: $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$	1: <b>if</b> $c = c^*$
2: $(K^*, c^*) \leftarrow \text{Encaps}(pk)$	2: <b>return</b> $\perp$
3: $K' \leftarrow \mathcal{A}^{O_{dec}(\cdot)}(pk, c^*)$	3: <b>else return</b>
4: <b>return</b> $[K' = K^*]$	4: $K \leftarrow \text{Decaps}(sk, c)$

**\*OW-CCA SECURITY GAME**

---

**Algorithm 1** Pseudocode for our attack algorithm

---

**Require:** a challenge ciphertext  $c^* \in \mathcal{R}_q$

**Ensure:** a secret key  $K \in \{0, 1\}^{2\lambda}$

```
for  $i \in \{1, \dots, n\}$  do
     $c' \leftarrow c^* + 2 \cdot e_i$  (Note that  $c' \neq c^*$ )
    Send  $c'$  to the decapsulation oracle  $O_{dec}$ 
    if  $O_{dec}$  outputs  $K'$  then
        Output  $K'$  as a decapsulation for  $c^*$ 
        break;
    end if
end for
```

---

**\*ATTACK ALGORITHM**

# ABOUT NTRU+ VERSION 1.1

- On 9/16/23, NTRU+ ver. 1.1 has been released
  - We checked that the attack strategy does not work for the updated algorithm.

---

**Algorithm 7** Inv

---

**Require:** Polynomial  $f \in R_q$

**Require:** Byte array  $B = (b_0, b_1, \dots, b_{n/4-1})$

**Ensure:** Message Byte array  $m = (m_0, m_1, \dots, m_{31})$

1:  $(\beta_0, \dots, \beta_{n-1}) := \text{BytesToBits}((b_0, \dots, b_{n/8-1}))$

2:  $(\beta_n, \dots, \beta_{2n-1}) := \text{BytesToBits}((b_{n/8}, \dots, b_{n/4-1}))$

3: **for**  $i$  from 0 to  $n - 1$  **do**

4:   **if**  $f_i + \beta_{i+n} \notin \{0, 1\}$ , **return**  $\perp$

5:    $m_i := ((f_i + \beta_{i+n}) \& 1) \oplus \beta_i$

$m = \text{BitsToBytes}((m_0, \dots, m_{n-1}))$

6: **return**  $m$

---

- Security proofs should be revised also.

# MEET-LWE ATTACK COSTS FOR LATTICE-BASED KEMS

- [May21] May, Alexander. "How to meet ternary LWE keys." *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II* 41. Springer International Publishing, 2021.
- Some parts of the slides introducing Meet LWE idea are taken from May's slide in Crypto 2021 ([\(40\) How to Meet Ternary LWE Keys – YouTube](#))

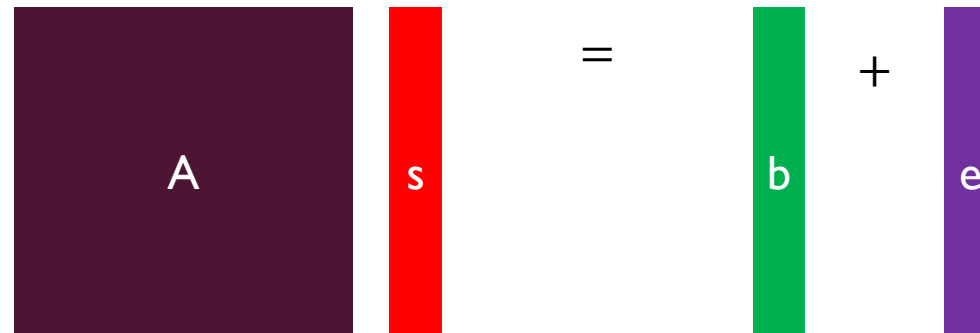


# TERNARY LWE PROBLEM

## [Ternary LWE problem]

Given;  $A \in \mathbb{Z}_q^{n \times n}, b \in \mathbb{Z}_q^n$  such that  $A \cdot s = b + e$  for  $s, e \in \{0, \pm 1\}^n$ ,

Find;  $s \in \{0, \pm 1\}^n$


$$A \cdot s = b + e$$

- Asymptotically, Brute Force < Odlyzko's MitM < Meet LWE
- Meet LWE can be extended to (fixed size of) small errors, so it is applicable to all 3 Lattice-based KEMs
  - SMAUG, TiGER use sparse secrets
  - NTRU+ uses the ternary LWE problem (they use sparse ternary or binary secrets)

# BRUTE FORCE ATTACK FOR TERNARY LWE

- Equation :  $A \cdot s = b + e \text{ mod } q$


$$A \cdot s = b + e$$

## [Brute Force]

- Input :  $A \in Z_q^{n \times n}, b \in Z_q^n$
- For all  $s \in \{0, \pm 1\}^n$ :
  - If  $A \cdot s - b \in \{0, \pm 1\}^n$  then output  $s$

- $S = 3^n$ ; search space size for ternary keys
- Running time is  $T = S$

# ODLYZKO'S MITM

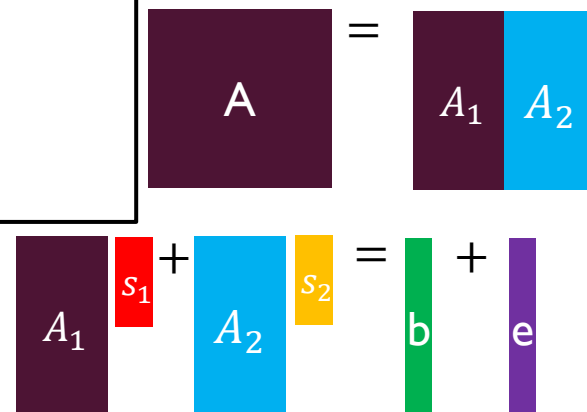
- Equation :  $A_1 \cdot s_1 = -A_2 \cdot s_2 + b + e \mod q$

i.e.  $A_1 \cdot s_1 \approx -A_2 \cdot s_2 + b \mod q$

## [Odlyzko's MitM]

- Input :  $A = (A_1|A_2) \in \mathbb{Z}_q^{n \times n}, b \in \mathbb{Z}_q^n$
- For all  $s_1 \in \{0, \pm 1\}^{n/2}$ :
  - Construct  $L_1$  with entries  $(s_1, h(A_1 s_1))$
- For all  $s_2 \in \{0, \pm 1\}^{n/2}$ :
  - Construct  $L_2$  with entries  $(s_2, h(-A_2 s_2 + b))$
- Output  $(s_1|s_2)$  with  $h(A_1 s_1) = h(-A_2 s_2 + b)$

\*  $h$ : locality sensitive hash



- $S = 3^n$ ; search space size for ternary keys
- Running time is  $T = 3^{n/2} = S^{1/2}$  with same memory

# REPRESENTATIONS (HOWGRAVE-GRAHAM, JOUX '10)

- **Idea** ;  $s := s_1 + s_2$  for  $s_1, s_2 \in \{0, \pm 1\}^n$

- Allows redundancy

$$\begin{array}{|c|c|} \hline A & s_1 \\ \hline \end{array} + \begin{array}{|c|c|} \hline A & s_2 \\ \hline \end{array} = \begin{array}{|c|} \hline b \\ \hline \end{array} + \begin{array}{|c|} \hline e \\ \hline \end{array}$$

- $(1, 0, 1, -1, 0) = (1, 0, 0, -1, 0) + (0, 0, 1, 0, 0)$

$$= (1, 0, 1, 0, 0) + (0, 0, 0, -1, 0)$$

$$= (0, 0, 1, 0, 0) + (1, 0, 0, -1, 0)$$

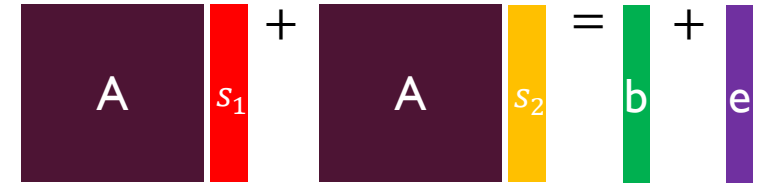
$$= (0, 0, 1, -1, 0) + (1, 0, 0, 0, 0)$$

	1		0		-1	
REP-0	• 1+0		-		• (-1)+0	
	• 0+1				• 0+(-1)	
REP-1	• 1+0		• 1+(-1)		• (-1)+0	
	• 0+1		• (-1)+1		• 0+(-1)	
REP-2	• 1+0	• 2+(-1)	• 1+(-1)	• 2+(-2)	• (-1)+0	• 1+(-2)
	• 0+1	• (-1)+2	• (-1)+1	• (-2)+2	• 0+(-1)	• (-2)+1

# MEET LWE ATTACK [MAY21]

- Equation :  $A \cdot s_1 = -A \cdot s_2 + b + e \mod q$

i.e.  $A \cdot s_1 \approx -A \cdot s_2 + b \mod q$


$$A \cdot s_1 = -A \cdot s_2 + b + e \mod q$$

## [Meet LWE (high-level idea)]

- Input :  $A \in \mathbb{Z}_q^{n \times n}, b \in \mathbb{Z}_q^n$
- Choose representation REP-0, REP-1, REP-2
- Guess  $r$  coordinates of  $e$  (say  $e_r$ )
- For  $s_1$ , construct  $L_1$  with entries  $(s_1, As_1)$
- For  $s_2$ , construct  $L_2$  with entries  $(s_2, -As_2 + b)$
- Output  $s_1 + s_2$  s.t.
  - $\pi_r(As_1) = \pi_r(-As_2 + b) + e_r$
  - $h(As_1) = h(-As_2 + b)$  for  $n - r$  coordinates

- By using representations  $s = s_1 + s_2$ , **the number of solutions ( $:= R$ ) increases**
- We can **reduce the list  $(L_1, L_2)$  sizes with a factor of  $R$**  (by guessing  $r$  coordinates of  $e$ ), expecting at least one solution exists
  - This strategy can be recursively applied to  $s_1, s_2$ , respectively (lists can be obtained by tree-based construction)
- Run-time  $T = T_g \cdot T_\ell$  where  $T_g$ ; guessing complexity,  $T_\ell$ ; list construction complexity

# ATTACK COMPLEXITIES OF MEET LWE

<pre> ***** Meet-LWE Rep-0 ***** SMAUG128: time= 230.9 = 217.0 + 13.9, memory= 217.0 SMAUG192: time= 294.8 = 278.5 + 16.3, memory= 278.5 SMAUG256: time= 368.2 = 350.8 + 17.4, memory= 350.8 ***** Meet-LWE Rep-1 ***** SMAUG128: time= 183.3 = 152.0 + 31.3, memory= 152.0 w SMAUG192: time= 233.1 = 192.4 + 40.6, memory= 192.4 w SMAUG256: time= 303.3 = 254.5 + 48.8, memory= 254.5 w ***** Meet-LWE Rep-2 ***** SMAUG128: time= 176.4 = 147.4 + 29.0, memory= 147.4 w SMAUG192: time= 229.9 = 199.7 + 30.2, memory= 199.7 w SMAUG256: time= 296.5 = 253.6 + 43.0, memory= 253.6 w </pre>	<pre> ***** Meet-LWE Rep-0 ***** NTRU+576: time= 340.7 = 322.5 + 18.2, memory= 322.5 NTRU+768: time= 454.8 = 430.3 + 24.6, memory= 430.3 NTRU+864: time= 512.7 = 484.2 + 28.5, memory= 484.2 NTRU+1152: time= 683.8 = 645.8 + 38.0, memory= 645.8 ***** Meet-LWE Rep-1 ***** NTRU+576: time= 269.4 = 237.7 + 31.7, memory= 237.7 with (1 NTRU+768: time= 361.7 = 320.5 + 41.2, memory= 320.5 with (2 NTRU+864: time= 411.4 = 366.2 + 45.2, memory= 366.2 with (2 NTRU+1152: time= 569.3 = 513.0 + 56.3, memory= 513.0 with ( ***** Meet-LWE Rep-2 ***** NTRU+576: time= 263.3 = 227.6 + 35.7, memory= 227.6 with (2 NTRU+768: time= 349.1 = 302.3 + 46.8, memory= 302.3 with (2 NTRU+864: time= 391.7 = 338.6 + 53.1, memory= 338.6 with (2 NTRU+1152: time= 518.6 = 448.1 + 70.5, memory= 448.1 with ( </pre>	<pre> ***** Meet-LWE Rep-0 ***** TIGER128: time= 225.3 = 213.4 + 11.9, memory= 213.4 TIGER192: time= 220.3 = 209.9 + 10.4, memory= 209.9 TIGER256: time= 387.7 = 369.5 + 18.2, memory= 369.5 ***** Meet-LWE Rep-1 ***** TIGER128: time= 175.5 = 150.2 + 25.4, memory= 150.2 with (11, 2, 1) TIGER192: time= 194.0 = 164.9 + 29.0, memory= 164.9 with (8, 1, 1) TIGER256: time= 309.5 = 269.1 + 40.4, memory= 269.1 with (16, 1, 1) ***** Meet-LWE Rep-2 ***** TIGER128: time= 167.0 = 141.7 + 25.4, memory= 141.7 with (11, 0, 2, 0) TIGER192: time= 170.1 = 141.1 + 29.0, memory= 141.1 with (8, 0, 1, 0) TIGER256: time= 298.5 = 258.1 + 40.4, memory= 258.1 with (16, 0, 3, 0) </pre>
---	--	--

- We slightly modified Meet-LWE algorithm for non-ternary errors
- **TiGER192** parameter is vulnerable to Meet-LWE attack
  - In their analysis, the claimed log complexity against best (quantum) attack was **192**, but it is dropped to **170.1** (Note. it is a classical attack)
  - (Recommendation) They need to increase  $h_s, h_r$  to fix it
- The other parameter sets of 3 lattice-based KEMs are fine

# OUR EXPERIMENT

```
from math import log, floor, sqrt, log2 #, prod
from scipy.special import gammaln
import numpy as np

def log2_multinom(c):
    return (gammaln(c.sum()+1) - gammaln(c+1).sum()) / log(2)

def meet_lwe_rep0(n, q, w, B):
    n2 = floor(n/2)

    w2 = floor(w/2)
    w4 = floor(w/4)
    w8 = floor(w/8)

    # Compute log_2 of  $L^{(1)} = S^{(1)} / R^{(1)}$ ,
    # where  $S^{(1)} = (n \text{ choose } w/4, w/4, n-w/2)$  and
    #  $R^{(1)} = (w/2 \text{ choose } w/4, w/4)$ .
    logS1 = log2_multinom(np.array([w4, w2-w4, n-w2]))
    logR1 = 2*log2_multinom(np.array([w4, w2-w4]))
    logL1 = logS1 - logR1

    # Compute log_2 of  $L^{(2)} = S^{(2)}$ ,
    # where  $S^{(2)} = n/2 \text{ choose } w/8, w/9, n/2-w/4$ 
    logL2 = log2_multinom(np.array([w8, w4-w8, n-n2-w4]))
```

- Use **Python code** to compute the Meet LWE attack costs for Rep-0, Rep-1, and Rep-2, respectively
  - We utilized the python code modified from an open source “Meet\_LWE.py” in SMAUG v1.0 helper scripts, by extending it into the non-ternary error cases
- **B**: error parameter for LWE

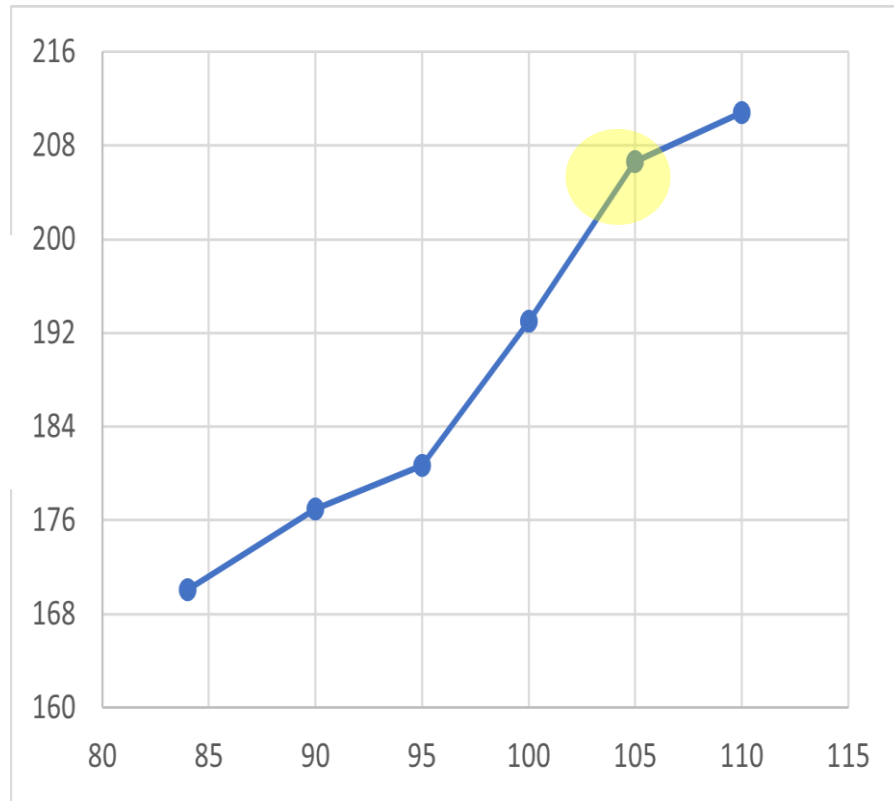
```
# Compute  $T_l$  for the list construction,
# where  $T_l = \max(L^{(1)}, L^{(2)})$ 
logT_l = max(logL1, logL2)

# Compute  $T_g$  for the guessing,
# where  $T_g = 3^{(r/2)} = 3^{(\log_q(R^{(1)}))/2}$ 
# which leads to  $\log_2(T_g) = 0.5 * \log_2(R^{(1)}) * \log_2(3) / \log_2(q)$ 
logT_g = 0.5*floor(logR1/log2(q))*log2(B)

return (logT_l+logT_g, logT_l, logT_g, logT_l)
```

# ATTACK COMPLEXITIES FOR VARIOUS PARAMETERS

↑  
 $Y =$   
Meet-LWE  
Time  
Complexity



—  $X = h_s$  (Hamming weight parameter  
of LWR's secret key  $sk$ .) →

- Meet LWE Complexity for the LWR instance in TiGER when increasing  $h_s$  (hamming weight of LWR's secret key  $sk$ .)
- (Recommendation) They need to increase  $h_s$  to be over **104** to achieve **200-bit classical security as claimed in TiGER** against the Meet-LWE attack. ( $104 \leq h_s$ )



# SECURITY EVALUATION OF {LWE, LWR}-BASED SCHEMES USING LATTICE ESTIMATOR

- [Lattice Estimator — Lattice Estimator 0.1 documentation \(lattice-estimator.readthedocs.io\)](https://lattice-estimator.readthedocs.io)
- Albrecht, Martin R., Rachel Player, and Sam Scott. "On the concrete hardness of learning with errors." *Journal of Mathematical Cryptology* 9.3 (2015): 169-203.

# GOAL

- Better understanding for the security estimation of KpqC Round I candidates
  - Analysis reports for the respective attacks
- Estimate the security for all the LWE/LWR based schemes {NTRU+, SMAUG, TiGER, HAETAE, NCC-Sign}

# METHODS

- **Lattice estimator**

- For LWE/LWR security analysis, M. Albrecht's Lattice Estimator ([Lattice Estimator — Lattice Estimator 0.1 documentation \(lattice-estimator.readthedocs.io\)](https://lattice-estimator.readthedocs.io)) is used. Lattice Estimator is a Sage open source that calculates the attack complexities and additional parameters required for attack by taking LWE/LWR parameters as input values.

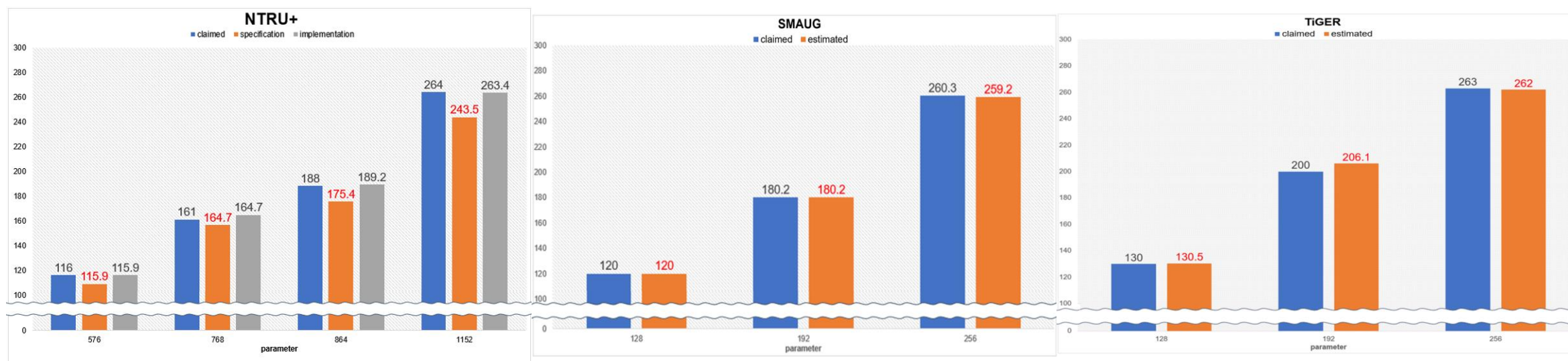
- **The BKZ Algorithm Complexity – Core-SVP model**

- The principle of the BKZ algorithm is to repeatedly apply the SVP solver, an algorithm that finds the shortest vector, for a sub-lattice of dimension ( $\beta$ ) smaller than that of a given lattice.
- The Core-SVP model from the NewHope paper (USENIX'16) is a model for estimating the time complexity of the BKZ algorithm. The classical security in bits is estimated as  $2^{c \cdot \beta}$  using  $c = 0.292$ , and the quantum security (bit) can be also estimated by calculating the classical security (bit)  $\times c_q/0.292$  in the Core-SVP model.

	Classical	Quantum[1]
$c$	0.292	0.257
$T$	$2^{0.292\beta}$	$2^{0.257\beta}$

[1] Chailloux, A., Loyer, J. Lattice Sieving via Quantum Random Walks. ASIACRYPT 2021

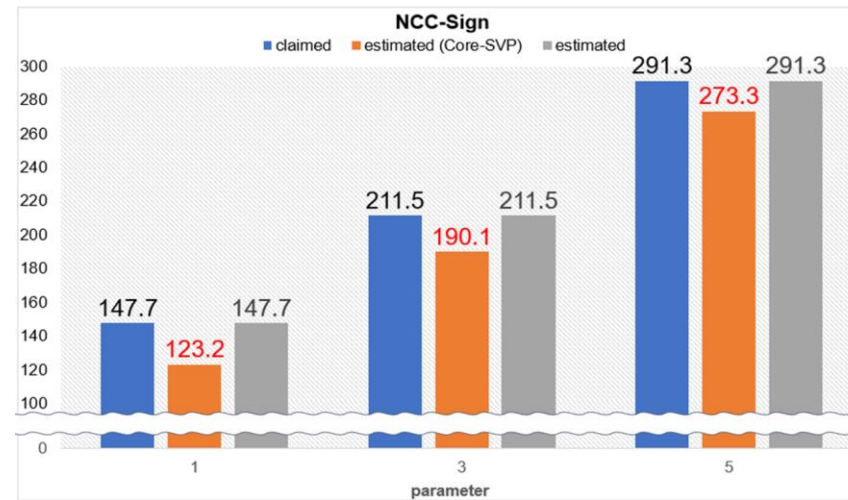
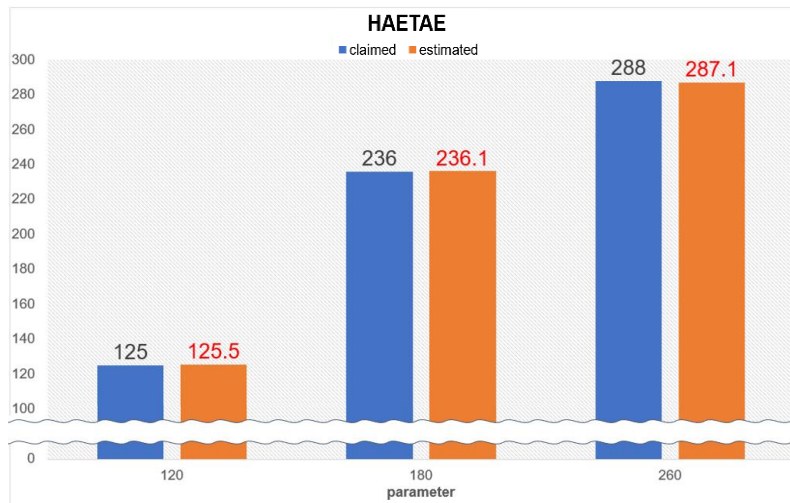
# RESULTS - KEMS



## Notes.

- NTRU+ in its specification uses the binary secrets for LWE (Algorithm 6, 9 in the specification), while it uses the centered binomial distribution for the LWE secrets in the implementation. So, we present evaluations for both.
- Estimated security for SMAUG-256, TiGER-256 ; 1-bit lower than the proposed security

# RESULTS - SIGNATURES



## Notes.

- NCC-Sign proposed the security without core-SVP model, so we presented the security evaluation with and without the Core-SVP model.

# SUMMARY

- CCA-NTRU+ can be attacked since their decoding method(the *Inv* algorithm) does not check if the intermediate value is binary
  - Can be fixed if they check if the intermediate value is binary, and abort otherwise.
- We evaluate the concrete security of 3 lattice-based KEMs against Meet LWE attack
  - TiGER needs to take into account Meet LWE attack for their TiGER192 parameter set
  - Can be fixed by increasing  $h_s, h_r$
  - TiGER updated the parameter sets : now secure against Meet-LWE attack

Table 1: The detail parameters for each security level

<i>parameters</i>	security level	$n$	$q$	$p$	$k_1$	$k_2$	$h_s$	$h_r$	$h_e$	$d$	$f$
TiGER128	AES128	512	256	128	64	64	160	128	32	128	3
TiGER192	AES192	1024	256	64	64	4	84	84	84	256	5
TiGER256	AES256	1024	256	128	128	4	198	198	32	256	5

Table 1: The detail parameters for each security level

<i>parameters</i>	security level	$n$	$q$	$p$	$k_1$	$k_2$	$h_s$	$h_r$	$h_e$	$d$	$f$
TiGER128	AES128	512	256	128	64	16	142	110	32	128	3
TiGER192	AES192	1024	256	128	64	4	132	132	32	256	5
TiGER256	AES256	1024	256	128	128	4	196	196	32	256	5

- We estimated the security of all the {LWE, LWR}-based schemes using Lattice estimator and verified the (most of) claims in the proposals of {NTRU+, SMAUG, TiGER, HAETAE, NCC-Sign}

---

# THANK YOU!



ANY QUESTIONS OR COMMENTS?

[JOOHEELEE@SUNGSHIN.AC.KR](mailto:JOOHEELEE@SUNGSHIN.AC.KR)

# COMMENTS FOR THE SECURITY PROOF

[FO transform without Re-Encryption (Lemma 5 in NTRU+ paper)]

<u>Gen(<math>1^\lambda</math>)</u> 1: $(pk, sk) := \text{Gen}'(1^\lambda)$ 2: <b>return</b> $(pk, sk)$	<u>Encap(<math>pk</math>)</u> 1: $m \leftarrow \mathcal{M}$ 2: $(r, K) := H(m)$ 3: $c := \text{Enc}'(pk, m; r)$ - $M := \text{SOTP}(m, G(r))$ - $c := \text{Enc}(pk, M; r)$ 4: <b>return</b> $(K, c)$
<u>Decap(<math>sk, c</math>)</u> 1: $m' := \text{Dec}'(sk, c)$ - $M' = \text{Dec}(sk, c)$ - $r' = \text{RRec}(pk, M', c)$ - $m' = \text{Inv}(M', G(r'))$ 2: $(r'', K') := H(m')$ 3: <b>if</b> $m' = \perp$ or $c \neq \text{Enc}'(pk, m'; r'')$ 4: <b>return</b> $\perp$ 5: <b>else</b> 6: <b>return</b> $K'$	

Fig. 9.  $\text{KEM} = \text{FO}^\perp[\text{PKE}', H]$ .

<u>Gen(<math>1^\lambda</math>)</u> 1: $(pk, sk) := \text{Gen}'(1^\lambda)$ 2: <b>return</b> $(pk, sk)$	<u>Encap(<math>pk</math>)</u> 1: $m \leftarrow \mathcal{M}$ 2: $(r, K) := H(m)$ 3: $c := \text{Enc}'(pk, m; r)$ - $M := \text{SOTP}(m, G(r))$ - $c := \text{Enc}(pk, M; r)$ 4: <b>return</b> $(K, c)$	<u>Decap(<math>sk, c</math>)</u> 1: $m' := \text{Dec}'(sk, c)$ - $M' = \text{Dec}(sk, c)$ - $r' = \text{RRec}(pk, M', c)$ - $m' = \text{Inv}(M', G(r'))$ 2: $(r'', K') := H(m')$ 3: <b>if</b> $m' = \perp$ or $r' \neq r''$ 4: <b>return</b> $\perp$ 5: <b>else</b> 6: <b>return</b> $K'$
--	---	--

Fig. 10.  $\text{KEM} = \overline{\text{FO}}^\perp[\text{PKE}', H]$ .



# COMMENTS FOR THE SECURITY PROOF

[FO transform without Re-Encryption (Lemma 5 in NTRU+ paper)]

- To show: For input ciphertext  $c$ ,

$$c = \text{Enc}'(pk, m'; r'') \text{ if and only if } r' = r''$$

<u>Gen(<math>1^\lambda</math>)</u> 1: $(pk, sk) := \text{Gen}'(1^\lambda)$ 2: <b>return</b> $(pk, sk)$	<u>Encap(<math>pk</math>)</u> 1: $m \leftarrow \mathcal{M}$ 2: $(r, K) := H(m)$ 3: $c := \text{Enc}'(pk, m; r)$ $- M := \text{SOTP}(m, G(r))$ $- c := \text{Enc}(pk, M; r)$ 4: <b>return</b> $(K, c)$
<u>Decap(<math>sk, c</math>)</u> 1: $m' := \text{Dec}'(sk, c)$ $- M' = \text{Dec}(sk, c)$ $- r' = \text{RRec}(pk, M', c)$ $- m' = \text{Inv}(M', G(r'))$ 2: $(r'', K') := H(m')$ 3: <b>if</b> $m' = \perp$ or $c \neq \text{Enc}'(pk, m'; r'')$ 4: <b>return</b> $\perp$ 5: <b>else</b> 6: <b>return</b> $K'$	

Fig. 9.  $\text{KEM} = \text{FO}^\perp[\text{PKE}', H]$ .

<u>Gen(<math>1^\lambda</math>)</u> 1: $(pk, sk) := \text{Gen}'(1^\lambda)$ 2: <b>return</b> $(pk, sk)$	<u>Encap(<math>pk</math>)</u> 1: $m \leftarrow \mathcal{M}$ 2: $(r, K) := H(m)$ 3: $c := \text{Enc}'(pk, m; r)$ $- M := \text{SOTP}(m, G(r))$ $- c := \text{Enc}(pk, M; r)$ 4: <b>return</b> $(K, c)$	<u>Decap(<math>sk, c</math>)</u> 1: $m' := \text{Dec}'(sk, c)$ $- M' = \text{Dec}(sk, c)$ $- r' = \text{RRec}(pk, M', c)$ $- m' = \text{Inv}(M', G(r'))$ 2: $(r'', K') := H(m')$ 3: <b>if</b> $m' = \perp$ or $r' \neq r''$ 4: <b>return</b> $\perp$ 5: <b>else</b> 6: <b>return</b> $K'$
--	---	--

Fig. 10.  $\text{KEM} = \overline{\text{FO}}^\perp[\text{PKE}', H]$ .

# COMMENTS FOR THE SECURITY PROOF

[FO transform without Re-Encryption (Lemma 5 in NTRU+ paper)]

- To show: For input ciphertext  $c$ ,

$$c = \text{Enc}'(pk, m'; r'') \text{ if and only if } r' = r''$$

- $(\rightarrow)$  Assume  $c = \text{Enc}'(pk, m'; r'')$  in Decaps.

Because  $\text{PKE}'$  is injective, the pair  $(m, r)$  used in Encaps is the same as  $(m', r'')$

⋮

<u>Gen(<math>1^\lambda</math>)</u> 1: $(pk, sk) := \text{Gen}'(1^\lambda)$ 2: <b>return</b> $(pk, sk)$	<u>Encap(<math>pk</math>)</u> 1: $m \leftarrow \mathcal{M}$ 2: $(r, K) := H(m)$ 3: $c := \text{Enc}'(pk, m; r)$ $- M := \text{SOTP}(m, G(r))$ $- c := \text{Enc}(pk, M; r)$ 4: <b>return</b> $(K, c)$
<u>Decap(<math>sk, c</math>)</u> 1: $m' := \text{Dec}'(sk, c)$ $- M' = \text{Dec}(sk, c)$ $- r' = \text{RRec}(pk, M', c)$ $- m' = \text{Inv}(M', G(r'))$ 2: $(r'', K') := H(m')$ 3: <b>if</b> $m' = \perp$ or $c \neq \text{Enc}'(pk, m'; r'')$ 4: <b>return</b> $\perp$ 5: <b>else</b> 6: <b>return</b> $K'$	

Fig. 9.  $\text{KEM} = \text{FO}^\perp[\text{PKE}', H]$ .

<u>Gen(<math>1^\lambda</math>)</u> 1: $(pk, sk) := \text{Gen}'(1^\lambda)$ 2: <b>return</b> $(pk, sk)$	<u>Encap(<math>pk</math>)</u> 1: $m \leftarrow \mathcal{M}$ 2: $(r, K) := H(m)$ 3: $c := \text{Enc}'(pk, m; r)$ $- M := \text{SOTP}(m, G(r))$ $- c := \text{Enc}(pk, M; r)$ 4: <b>return</b> $(K, c)$	<u>Decap(<math>sk, c</math>)</u> 1: $m' := \text{Dec}'(sk, c)$ $- M' = \text{Dec}(sk, c)$ $- r' = \text{RRec}(pk, M', c)$ $- m' = \text{Inv}(M', G(r'))$ 2: $(r'', K') := H(m')$ 3: <b>if</b> $m' = \perp$ or $r' \neq r''$ 4: <b>return</b> $\perp$ 5: <b>else</b> 6: <b>return</b> $K'$
--	---	--

Fig. 10.  $\text{KEM} = \overline{\text{FO}}^\perp[\text{PKE}', H]$ .

# COMMENTS FOR THE SECURITY PROOF

[FO transform without Re-Encryption (Lemma 5 in NTRU+ paper)]

- To show: For input ciphertext  $c$ ,

$$c = \text{Enc}'(pk, m'; r'') \text{ if and only if } r' = r''$$

- $(\rightarrow)$  Assume  $c = \text{Enc}'(pk, m'; r'')$  in Decaps.

Because  $\text{PKE}'$  is injective, the pair  $(m, r)$  used in Encaps is the same as  $(m', r'')$

⋮

They assumed that  $c$  (input of Decaps) is an output of Encaps,  
i.e.,  $c = \text{Encaps}(m, r)$  for some  $(m, r)$ .

**But, there is no guarantee that  $c = \text{Encaps}(m, r)$  for some  $m \in M, r \in R$**

<p><u>Gen(<math>1^\lambda</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>(pk, sk) := \text{Gen}'(1^\lambda)</math></li> <li>2: <b>return</b> <math>(pk, sk)</math></li> </ol>	<p><u>Encap(<math>pk</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>m \leftarrow \mathcal{M}</math></li> <li>2: <math>(r, K) := \text{H}(m)</math></li> <li>3: <math>c := \text{Enc}'(pk, m; r)</math>  <math>- M := \text{SOTP}(m, G(r))</math>  <math>- c := \text{Enc}(pk, M; r)</math></li> <li>4: <b>return</b> <math>(K, c)</math></li> </ol>
<p><u>Decap(<math>sk, c</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>m' := \text{Dec}'(sk, c)</math>  <math>- M' = \text{Dec}(sk, c)</math>  <math>- r' = \text{RRec}(pk, M', c)</math>  <math>- m' = \text{Inv}(M', G(r'))</math></li> <li>2: <math>(r'', K') := \text{H}(m')</math></li> <li>3: <b>if</b> <math>m' = \perp</math> or <math>c \neq \text{Enc}'(pk, m'; r'')</math></li> <li>4:   <b>return</b> <math>\perp</math></li> <li>5: <b>else</b></li> <li>6:   <b>return</b> <math>K'</math></li> </ol>	

Fig. 9.  $\text{KEM} = \text{FO}^\perp[\text{PKE}', \text{H}]$ .

<p><u>Gen(<math>1^\lambda</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>(pk, sk) := \text{Gen}'(1^\lambda)</math></li> <li>2: <b>return</b> <math>(pk, sk)</math></li> </ol>	<p><u>Encap(<math>pk</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>m \leftarrow \mathcal{M}</math></li> <li>2: <math>(r, K) := \text{H}(m)</math></li> <li>3: <math>c := \text{Enc}'(pk, m; r)</math>  <math>- M := \text{SOTP}(m, G(r))</math>  <math>- c := \text{Enc}(pk, M; r)</math></li> <li>4: <b>return</b> <math>(K, c)</math></li> </ol>	<p><u>Decap(<math>sk, c</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>m' := \text{Dec}'(sk, c)</math>  <math>- M' = \text{Dec}(sk, c)</math>  <math>- r' = \text{RRec}(pk, M', c)</math>  <math>- m' = \text{Inv}(M', G(r'))</math></li> <li>2: <math>(r'', K') := \text{H}(m')</math></li> <li>3: <b>if</b> <math>m' = \perp</math> or <math>r' \neq r''</math></li> <li>4:   <b>return</b> <math>\perp</math></li> <li>5: <b>else</b></li> <li>6:   <b>return</b> <math>K'</math></li> </ol>
--	--	--

Fig. 10.  $\text{KEM} = \overline{\text{FO}}^\perp[\text{PKE}', \text{H}]$ .

# COMMENTS FOR THE SECURITY PROOF

[FO transform without Re-Encryption (Lemma 5 in NTRU+ paper)]

- To show: For input ciphertext  $c$ ,

$$c = \text{Enc}'(pk, m'; r'') \text{ if and only if } r' = r''$$

- ( $\leftarrow$ ) Assume  $r' = r''$  in Decaps.

Because SOTP is rigid,  $m' = \text{Inv}(M', G(r'))$  implies  $M' = \text{SOTP}(m', G(r'))$ , and thus  $M' = \text{SOTP}(m', G(r''))$

⋮

<u>Gen(<math>1^\lambda</math>)</u> 1: $(pk, sk) := \text{Gen}'(1^\lambda)$ 2: <b>return</b> $(pk, sk)$	<u>Encap(<math>pk</math>)</u> 1: $m \leftarrow \mathcal{M}$ 2: $(r, K) := H(m)$ 3: $c := \text{Enc}'(pk, m; r)$ - $M := \text{SOTP}(m, G(r))$ - $c := \text{Enc}(pk, M; r)$ 4: <b>return</b> $(K, c)$
<u>Decap(<math>sk, c</math>)</u> 1: $m' := \text{Dec}'(sk, c)$ - $M' = \text{Dec}(sk, c)$ - $r' = \text{RRec}(pk, M', c)$ - $m' = \text{Inv}(M', G(r'))$ 2: $(r'', K') := H(m')$ 3: <b>if</b> $m' = \perp$ or $c \neq \text{Enc}'(pk, m'; r'')$ 4: <b>return</b> $\perp$ 5: <b>else</b> 6: <b>return</b> $K'$	

Fig. 9.  $\text{KEM} = \text{FO}^\perp[\text{PKE}', H]$ .

<u>Gen(<math>1^\lambda</math>)</u> 1: $(pk, sk) := \text{Gen}'(1^\lambda)$ 2: <b>return</b> $(pk, sk)$	<u>Encap(<math>pk</math>)</u> 1: $m \leftarrow \mathcal{M}$ 2: $(r, K) := H(m)$ 3: $c := \text{Enc}'(pk, m; r)$ - $M := \text{SOTP}(m, G(r))$ - $c := \text{Enc}(pk, M; r)$ 4: <b>return</b> $(K, c)$	<u>Decap(<math>sk, c</math>)</u> 1: $m' := \text{Dec}'(sk, c)$ - $M' = \text{Dec}(sk, c)$ - $r' = \text{RRec}(pk, M', c)$ - $m' = \text{Inv}(M', G(r'))$ 2: $(r'', K') := H(m')$ 3: <b>if</b> $m' = \perp$ or $r' \neq r''$ 4: <b>return</b> $\perp$ 5: <b>else</b> 6: <b>return</b> $K'$
--	---	--

Fig. 10.  $\text{KEM} = \overline{\text{FO}}^\perp[\text{PKE}', H]$ .

# COMMENTS FOR THE SECURITY PROOF

[FO transform without Re-Encryption (Lemma 5 in NTRU+ paper)]

- To show: For input ciphertext  $c$ ,

$$c = \text{Enc}'(pk, m'; r'') \text{ if and only if } r' = r''$$

- ( $\leftarrow$ ) Assume  $r' = r''$  in Decaps.

Because SOTP is rigid,  $m' = \text{Inv}(M', G(r'))$  implies  $M' = \text{SOTP}(m', G(r'))$ , and thus  $M' = \text{SOTP}(m', G(r''))$

⋮

They assumed that  $M' = \text{Dec}(sk, c)$  is an output of SOTP w.r.t.  $u = G(r')$ , i.e.,  $M' = \text{SOTP}(m, G(r'))$  for some  $m$ .

**But, there is no guarantee that  $M' \in \{\text{SOTP}(m, G(r')) \mid m \in M\}$  as shown in our attack.**

(Recall) Rigidity of SOTP ;

For all  $u \in U$ , and  $y \in Y$  encoded with respect to  $u$ , it holds that  $\text{SOTP}(\text{Inv}(y, u), u) = y$

<p><u>Gen(<math>1^\lambda</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>(pk, sk) := \text{Gen}'(1^\lambda)</math></li> <li>2: <b>return</b> <math>(pk, sk)</math></li> </ol>	<p><u>Encap(<math>pk</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>m \leftarrow \mathcal{M}</math></li> <li>2: <math>(r, K) := H(m)</math></li> <li>3: <math>c := \text{Enc}'(pk, m; r)</math></li> <li style="padding-left: 20px;"><math>- M := \text{SOTP}(m, G(r))</math></li> <li style="padding-left: 20px;"><math>- c := \text{Enc}(pk, M; r)</math></li> <li>4: <b>return</b> <math>(K, c)</math></li> </ol>
<p><u>Decap(<math>sk, c</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>m' := \text{Dec}'(sk, c)</math></li> <li style="padding-left: 20px;"><math>- M' = \text{Dec}(sk, c)</math></li> <li style="padding-left: 20px;"><math>- r' = \text{RRec}(pk, M', c)</math></li> <li style="padding-left: 20px;"><math>- m' = \text{Inv}(M', G(r'))</math></li> <li>2: <math>(r'', K') := H(m')</math></li> <li>3: <b>if</b> <math>m' = \perp</math> or <math>c \neq \text{Enc}'(pk, m'; r'')</math></li> <li>4:     <b>return</b> <math>\perp</math></li> <li>5: <b>else</b></li> <li>6:     <b>return</b> <math>K'</math></li> </ol>	

Fig. 9.  $\text{KEM} = \text{FO}^\perp[\text{PKE}', H]$ .

<p><u>Gen(<math>1^\lambda</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>(pk, sk) := \text{Gen}'(1^\lambda)</math></li> <li>2: <b>return</b> <math>(pk, sk)</math></li> </ol>	<p><u>Encap(<math>pk</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>m \leftarrow \mathcal{M}</math></li> <li>2: <math>(r, K) := H(m)</math></li> <li>3: <math>c := \text{Enc}'(pk, m; r)</math></li> <li style="padding-left: 20px;"><math>- M := \text{SOTP}(m, G(r))</math></li> <li style="padding-left: 20px;"><math>- c := \text{Enc}(pk, M; r)</math></li> <li>4: <b>return</b> <math>(K, c)</math></li> </ol>	<p><u>Decap(<math>sk, c</math>)</u></p> <ol style="list-style-type: none"> <li>1: <math>m' := \text{Dec}'(sk, c)</math></li> <li style="padding-left: 20px;"><math>- M' = \text{Dec}(sk, c)</math></li> <li style="padding-left: 20px;"><math>- r' = \text{RRec}(pk, M', c)</math></li> <li style="padding-left: 20px;"><math>- m' = \text{Inv}(M', G(r'))</math></li> <li>2: <math>(r'', K') := H(m')</math></li> <li>3: <b>if</b> <math>m' = \perp</math> or <math>r' \neq r''</math></li> <li>4:     <b>return</b> <math>\perp</math></li> <li>5: <b>else</b></li> <li>6:     <b>return</b> <math>K'</math></li> </ol>
--	---	---

Fig. 10.  $\text{KEM} = \text{FO}^\perp[\text{PKE}', H]$ .