

SMAUG-T, the module lattice based KEM algorithm

Jung Hee Cheon^{1, 2}, Hyeongmin Choe¹, **Hyoeun Seong**², Dongyeon Hong,
Jeongdae Hong³, Junbum Shin²,
Joongeun Choi⁴, Chi-gon Jung⁴, **Seunghwan Park**⁴, Honggoo Kang⁴,
Janghyeon Lee⁴, Seonghyuck Lim⁴, Aesun Park⁴

¹Seoul National University, ²CryptoLab Inc., ³Ministry of National Defense,
⁴Defense Counterintelligence Command

KpqC February 27, 2024



SMAUG-T
HEAAN
CRYPTO LAB  Defense Counterintelligence
Command

Contents

- **Part1.** What is SMAUG-T?
 - Preliminaries
 - History
 - Design goal & Key features
 - Schemes
 - Security
 - Benchmark
- **Part2.** Updates in Round2
 - Side-Channel Attack
 - TiMER : 4th parameter for IoT
 - Conclusion

Preliminaries

- LWE (Learning With Errors)

$$\left(\begin{matrix} \text{m} \\ \text{n} \end{matrix} \right) \left(\begin{matrix} \text{A} \\ \text{b} \end{matrix} \right) = \left(\begin{matrix} \text{A} \\ \text{s} \end{matrix} \right) + \left(\begin{matrix} \text{e} \end{matrix} \right)$$

where $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \leftarrow \chi_s^n$, $\mathbf{e} \leftarrow \chi_e^m$ (χ_e : Discrete Gaussian distribution),

- Search-LWE: Given (\mathbf{A}, \mathbf{b}) , where $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$, find \mathbf{s}
- Decision-LWE: Given samples (\mathbf{A}, \mathbf{b}) , decide whether they are LWE samples or uniformly random samples

- LWR (Learning With Rounding)

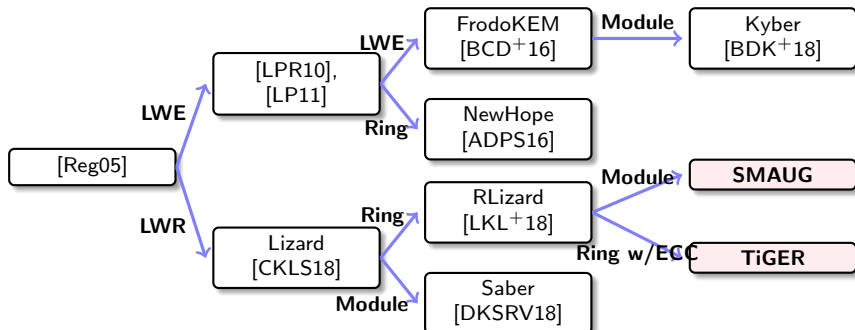
$$\left(\begin{array}{c} \overbrace{\mathbf{A}}^n \\ \underbrace{\phantom{\mathbf{A}}}_m \end{array} , \mathbf{b} \right) = \left\lfloor \frac{p}{q} \cdot \begin{array}{c} \mathbf{A} \quad \mathbf{s} \end{array} \right\rfloor$$

where $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \leftarrow \chi_s^n$, $\mathbf{e} \leftarrow \chi_e^m$,

- Search-LWR: Given (\mathbf{A}, \mathbf{b}) , where $\mathbf{b} = \lfloor \frac{p}{q} \cdot \mathbf{A} \cdot \mathbf{s} \rfloor$, find \mathbf{s}
- Decision-LWR: Given samples (\mathbf{A}, \mathbf{b}) , decide whether they are LWR samples or uniformly random samples

History of Lattice PKE/KEM

Development of PKE and KEM based on LWE and LWR



SMAUG & TiGER are merged in Round2!



Parameter sets

- **SMAUG-T**

- SMAUG-T128
- SMAUG-T192
- SMAUG-T256
- Additional parameter for IoT
 - (Called) **TIMER** : Tiny SMAUG-T using Error Reconciliation

Design goal

Goal: Suitable to lightweight devices

- Better performance: faster than previous works
- Small memory: smaller size in software
- Low communication cost: smaller key and ciphertext size
- Resistant to side-channel attack: constant-time implementation

Key features

- Take both advantages of $\text{LWE} + \text{LWR}$
 - LWE: Conservative security guarantee for keys
 - LWR: Efficient encryption and decryption
- Module structure
 - Flexible security level and compact $pk, ctxt$ size
- Power-of-two moduli
 - Easy implementation of sampling and rounding
 - No arithmetic for modulus reduction
- Sparse secret
 - Small secret key and fast polynomial multiplication
 - Security of LWE with sparse secret is guaranteed [CHK⁺16]

Key features

- Take both advantages of $\text{LWE} + \text{LWR}$
 - LWE: Conservative security guarantee for keys
 - LWR: Efficient encryption and decryption
- Module structure
 - Flexible security level and compact $pk, ctext$ size
- Power-of-two moduli
 - Easy implementation of sampling and rounding
 - No arithmetic for modulus reduction
- Sparse secret
 - Small secret key and fast polynomial multiplication
 - Security of LWE with sparse secret is guaranteed [CHK⁺16]

Key features

- Take both advantages of $\text{LWE} + \text{LWR}$
 - LWE: Conservative security guarantee for keys
 - LWR: Efficient encryption and decryption
- Module structure
 - Flexible security level and compact $pk, ctxt$ size
- Power-of-two moduli
 - Easy implementation of sampling and rounding
 - No arithmetic for modulus reduction
- Sparse secret
 - Small secret key and fast polynomial multiplication
 - Security of LWE with sparse secret is guaranteed [CHK⁺16]

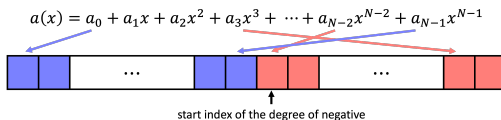
Key features

- Take both advantages of $\text{LWE} + \text{LWR}$
 - LWE: Conservative security guarantee for keys
 - LWR: Efficient encryption and decryption
- Module structure
 - Flexible security level and compact $pk, ctxt$ size
- Power-of-two moduli
 - Easy implementation of sampling and rounding
 - No arithmetic for modulus reduction
- Sparse secret
 - Small secret key and fast polynomial multiplication
 - Security of LWE with sparse secret is guaranteed [CHK⁺16]

Key features

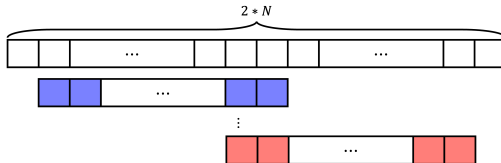
- Sparse secret

- Given a sparse polynomial $a(x)$ of degree N with hamming weight h , store the degrees of non-zero coefficient into the array like below:



- Polynomial multiplication

- Only with add and subtraction in $2 * N$ space



- Parameter sets

security level	TiMER I	SMAUG-T128 I	SMAUG-T192 III	SMAUG-T256 V
n	256	256	256	256
k	2	2	3	5
(q, p)	(1024, 256)	(1024, 256)	(2048, 256)	(2048, 256)
(p', t)	(8, 2)	(32, 2)	(256, 2)	(64, 2)
(h_s, h_r)	(100, 132)	(140, 132)	(198, 151)	(176, 160)
σ	1.453713	1.0625	1.453713	1.0625

- Polynomial Ring

- $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N + 1) = \{a_0 + a_1x^1 \cdots + a_{N-1}x^{N-1} \mid a_i \in \mathbb{Z}_q\}$

- Module Lattice

- (Euclidean) Lattice: vectors whose elements are integers from \mathbb{Z}_q
 - Module Lattice: vectors whose elements are polynomials from \mathcal{R}_q

- Operations

- $\lfloor \mathbf{a} \rfloor$: Round each coefficients of polynomial $a_i(x)$ in a vector \mathbf{a}
 - expandA: Sample uniform random matrix \mathbf{A} from seed
 - HWT_h : Sample a ternary polynomial having hamming weight h
 - dGaussian_σ : Discrete Gaussian sampling with a standard deviation σ

Schemes: IND-CPA PKE

- $\text{KeyGen}(1^\lambda)$:

$$\text{pk} = \left(\overset{k}{\underbrace{\begin{matrix} k & \text{A} & k \end{matrix}}} , \text{b} = - \text{A}^T \text{s} + \text{e} \right) \quad \text{sk} = \text{s}$$

- 1: $\text{seed} \xleftarrow{\$} \{0, 1\}^{256}$
- 2: $(\text{seed}_A, \text{seed}_{sk}, \text{seed}_e) \leftarrow \text{XOF}(\text{seed})$
- 3: $\mathbf{A} \leftarrow \text{expandA}(\text{seed}_A) \in \mathcal{R}_q^{k \times k}$
- 4: $\mathbf{s} \leftarrow \text{HWT}_{hs}(\text{seed}_{sk}) \in \mathcal{S}_\eta^k, \quad \mathbf{e} \leftarrow \text{dGaussian}_\sigma(\text{seed}_e) \in \mathcal{R}^k$
- 5: $\mathbf{b} = -\mathbf{A}^T \cdot \mathbf{s} + \mathbf{e} \in \mathcal{R}_q^k$
- 6: **return** $\text{pk} = (\text{seed}_A, \mathbf{b}), \text{sk} = \mathbf{s}$

Schemes: IND-CPA PKE

- $\text{Encrypt}(\text{pk}, \mu; \text{seed}_{\mathbf{r}})$:

$$\text{ct} = \left(\left\lfloor \frac{p}{q} \cdot \mathbf{A} \cdot \mathbf{r} \right\rfloor, \left\lfloor \frac{p'}{q} \cdot \mathbf{b}^T \cdot \mathbf{r} + \frac{p'}{t} \cdot \mu \right\rfloor \right)$$

- 1: $\mathbf{A} = \text{expandA}(\text{seed}_{\mathbf{A}})$
- 2: $\mathbf{r} \leftarrow \text{HWT}_{hr}(\text{seed}_{\mathbf{r}}) \in \mathcal{S}_{\eta}^k$
- 3: $\mathbf{c}_1 = \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{r} \rfloor \in \mathcal{R}_p^k$
- 4: $c_2 = \lfloor p'/q \cdot \mathbf{b}^T \cdot \mathbf{r} + p'/t \cdot \mu \rfloor \in \mathcal{R}_{p'}$
- 5: **return** $\text{ct} = (\mathbf{c}_1, c_2)$

* $\text{pk} = (\text{seed}_{\mathbf{A}}, \mathbf{b}), \mu \in \mathcal{R}_t$

Schemes: IND-CPA PKE

- Decrypt(sk, ct):

1: $\mu' = \lfloor t/p \cdot \mathbf{c}_1^T \cdot \mathbf{s} + t/p' \cdot c_2 \rfloor \in \mathcal{R}_t$ * sk=s, ct=(c₁, c₂)

2: **return** μ'

- Correctness

- A ciphertext \mathbf{c}_1 and c_2 can be expressed with errors:

$$\mathbf{c}_1 = \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{r} \rfloor = p/q \cdot (\mathbf{A} \cdot \mathbf{r} + \mathbf{e}_1)$$

$$c_2 = \lfloor p'/q \cdot \mathbf{b}^T \cdot \mathbf{r} + p'/t \cdot \mu \rfloor = p'/q \cdot (\mathbf{b}^T \cdot \mathbf{r} + e_2) + p'/t \cdot \mu$$

- Then, in decryption

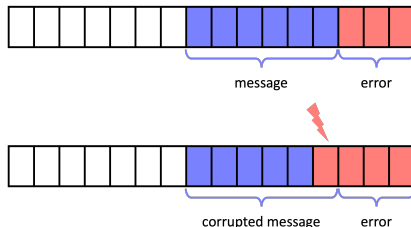
$$\begin{aligned} \left\lfloor \frac{t}{p} \cdot \mathbf{c}_1^T \cdot \mathbf{s} + \frac{t}{p'} c_2 \right\rfloor &= \left\lfloor \frac{t}{q} \cdot \mathbf{r}^T \cdot (\mathbf{A}^T \mathbf{s} + \mathbf{b}) + \mu + \frac{t}{q} \cdot (\mathbf{s}^T \mathbf{e}_1 + e_2) \right\rfloor \\ &= \mu + \underbrace{\left\lfloor \frac{t}{q} \cdot (\mathbf{r}^T \mathbf{e} + \mathbf{s}^T \mathbf{e}_1 + e_2) \right\rfloor}_{\text{error term}}. \end{aligned}$$

Schemes: IND-CPA PKE

- Error bound

- To recover the message μ correctly, the ∞ -norm of error must be smaller than $q/2t$.
- We define a decryption failure probability (DFP) δ :

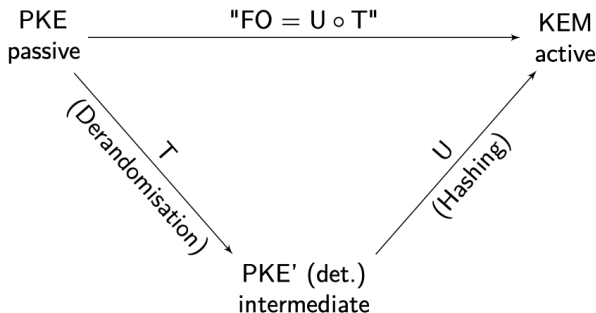
$$\delta = \Pr \left[\left\| \mathbf{r}^T \mathbf{e} + \mathbf{s}^T \mathbf{e}_1 + e_2 \right\|_{\infty} > q/2t \right].$$



Schemes: FO transformation

Basic idea: FO (Fujisaki-Okamoto) transformation [FO99]

- From IND-CPA PKE to IND-CCA KEM scheme
- Security reduction in ROM and QROM [HHK17, SXY18]



Schemes: FO transformation

Two steps of FO transformation

- Derandomization T

- Make an IND-CPA PKE scheme deterministic

$$\text{Encrypt}_1(pk, \mu) = \text{Encrypt}(pk, \mu; H(\mu))$$

where $H(\mu)$ is as the random coin [HHK17].

- Hashing U

- U^\perp : turn an IND-CPA PKE into an IND-CCA KEM

Encapsulation

- Choose a uniformly random message μ
- $c = \text{Encrypt}_1(pk, \mu)$
- $K = \text{KDF}(\mu, c)$

(a)

Decapsulation

- $\mu' = \text{Decrypt}_1(sk, c)$
- if $\mu' = \perp$ return \perp
- else return $K = \text{KDF}(\mu', c)$

(b)

Schemes: FO transformation

Many variants of U^\perp

- Implicit rejection

Encapsulation

1. Choose a uniformly random message μ
2. $c = \text{Encrypt}_1(pk, \mu)$
3. $K = \text{KDF}(\mu, c)$

(c)

Decapsulation

1. $\mu' = \text{Decrypt}_1(sk, c)$
2. if $\mu' = \perp$ return $\text{KDF}(t, c)$ (a.k.a implicit rejection)
3. else return $K = \text{KDF}(\mu', c)$

where t is a random value in a secret key

(d)

Schemes: FO transformation

Many variants of U^\perp

- Re-encryption

Encapsulation

1. Choose a uniformly random message μ
2. $c = \text{Encrypt}_1(pk, \mu)$
3. $K = \text{KDF}(\mu, c)$

(e)

Decapsulation

1. $\mu' = \text{Decrypt}_1(sk, c)$
2. if $c \neq \text{Encrypt}_1(pk, \mu')$ return $\text{KDF}(t, c)$ (a.k.a re-encryption)
3. else return $K = \text{KDF}(\mu', c)$

(f)

Schemes: IND-CCA KEM

Apply variant of FO transform FO_m^{\perp}

- use hash of public key to prevent multi-target attack
- generate shared key without ciphertext contribution

KeyGen(1^λ):

- 1: $(pk, sk') \leftarrow \text{PKE.KeyGen}(1^\lambda)$
- 2: $d \leftarrow \{0, 1\}^{256}$
- 3: **return** $pk, sk = (sk', t)$

Encap(pk):

- 1: $\mu \leftarrow \{0, 1\}^{256}$
- 2: $(K, \text{seed}) \leftarrow G(\mu, H(pk))$ * Hashing U & Avoid multi-target attack
- 3: $ct \leftarrow \text{PKE.Encrypt}(pk, \mu; \text{seed})$ * Derandomization T ($G(\mu)$)
- 4: **return** ct, K

Schemes: IND-CCA KEM

Apply variant of FO transform FO_m^\perp

- use hash of public key to prevent mult-target attack
- generate shared key without ciphertext contribution

Decap(sk, ct):

▷ $\text{sk} = (\text{sk}', t)$

- 1: $\mu' = \text{PKE.Decrypt}(\text{sk}', \text{ct})$
- 2: $(K', \text{seed}') \leftarrow G(\mu', H(\text{pk}))$
- 3: $\text{ct}' = \text{PKE.Encrypt}(\text{pk}, \mu'; \text{seed}')$ * re-encryption
- 4: $(\hat{K}, \cdot) \leftarrow G(d, H(\text{ct}))$
- 5: **if** $\text{ct} \neq \text{ct}'$ **then**
- 6: $K' \leftarrow \hat{K}$ * implicit rejection \nless
- 7: **end if**
- 8: **return** K'

Security estimation and DFP of SMAUG-T parameter sets.

- Core-SVP and DFP

Parameters sets Target security	TiMER I	SMAUG-T128 I	SMAUG-T192 III	SMAUG-T256 V
Classical core-SVP	120.0	120.0	181.7	264.5
Quantum core-SVP	105.6	105.6	160.9	245.2
Dec. fail. prob.*	-132.9	-119.6	-136.1	-167.2

*Decryption failure probability δ is in \log_2 scale.

- Beyond the core-SVP

Parameters sets Target security	TiMER I	SMAUG-T128 I	SMAUG-T192 III	SMAUG-T256 V
Arora-Ge	653.8(317.2)	741.3(598.0)	964.4(636.5)	-
BKW	135.3(123.2)	144.7(133.7)	202.0(190.7)	274.6(256.9)
Meet-LWE	144.3(123.7)	164.3(143.7)	213.8(192.4)	283.2(254.7)

Benchmark

Performance and size (cpu cycles, bytes)

- Reference & Additional implementation

	TiMER	SMAUG-T128	SMAUG-T192	SMAUG-T256
Keygen	65839	66491	123522	220780
Encap	69070	72122	119897	216542
Decap	88509	94115	150358	255035
Secret key	136 (808)	176 (848)	236 (1324)	218 (2010)
Public key	672	672	1088	1792
Ciphertext	608	672	1024	1472

Intel(R) Core i7-10700K (3.80GHz), gcc 11.4.0

- The most speed critical component is the symmetric primitive.
 - We chose SHA3 (Keccak) as the symmetric variant
 - It accounts for approximately 30-40% of the total cycles

Benchmark

Performance (cpu cycles)

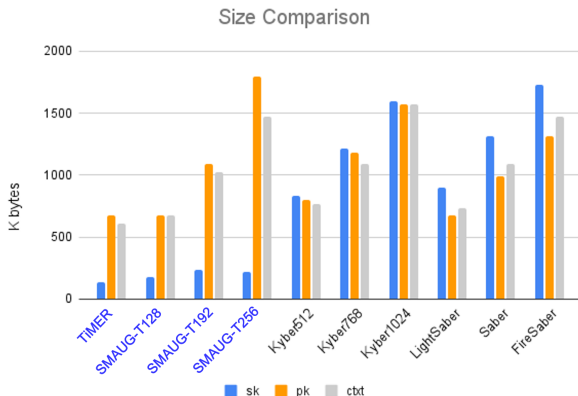
- Optimized implementation
 - Support 90s symmetric variant for fair benchmarking

SMAUG-T	Reference	AVX2		AVX2 90s	
Keygen I	66491	40179	x1.6	23178	x2.8
Keygen III	123522	71442	x1.7	41963	x2.9
Keygen V	220780	123166	x1.8	63610	x3.4
Encap I	72122	42167	x1.7	30673	x2.3
Encap III	119897	71114	x1.7	49692	x2.4
Encap V	216542	129281	x1.6	76658	x2.8
Decap I	94115	57374	x1.6	46439	x2
Decap III	150358	86914	x1.7	64380	x2.3
Decap V	255035	150244	x1.7	97721	x2.6

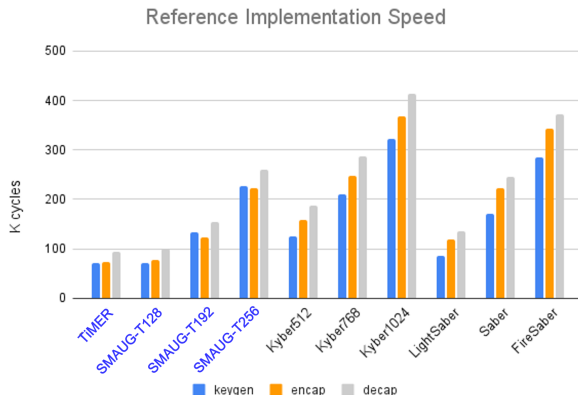
Intel(R) Core i7-10700K (3.80GHz), gcc 11.4.0

*We still in progress

SMAUG-T is **efficient & fast!**



SMAUG-T is **efficient & fast!**



Side Channel Attack

Enhanced Security Against Side-Channel Attacks

- Gaussian sampler
 - **Issue** : An error e sampled from a Gaussian distribution can be distinguished into two sets using side channel information
 - **Improvement** : A shuffling algorithm was applied to the implementation code
- Hamming weight sampler
 - **Issue** : Constant time implementation
 - **Improvement** : HWT sampling updated to avoid rejection sampling
 - We applied CWW sampling(SMAUG v2.0)
 - To eliminate the bias, the implementation code was modified

Side Channel Attack

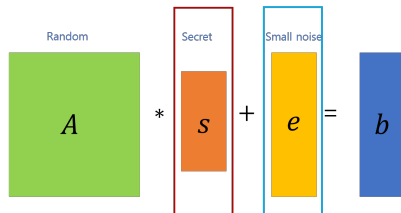
Enhanced Security Against Side-Channel Attacks

- Gaussian sampler
 - **Issue** : An error e sampled from a Gaussian distribution can be distinguished into two sets using side channel information
 - **Improvement** : A shuffling algorithm was applied to the implementation code
- Hamming weight sampler
 - **Issue** : Constant time implementation
 - **Improvement** : HWT sampling updated to avoid rejection sampling
 - We applied CWW sampling(SMAUG v2.0)
 - To eliminate the bias, the implementation code was modified

Side Channel Attack

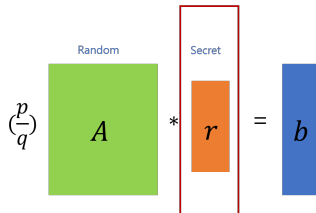
KeyGen(1^λ):

- 1: $\text{seed} \leftarrow \{0, 1\}^{256}$
- 2: $(\text{seed}_A, \text{seed}_{sk}, \text{seed}_e) \leftarrow \text{XOF}(\text{seed})$
- 3: $\mathbf{A} \leftarrow \text{expandA}(\text{seed}_A) \in \mathcal{R}_q^{k \times k}$
- 4: $\mathbf{s} \leftarrow \text{HWT}_{h_s}(\text{seed}_{sk}) \in S_\eta^k$
- 5: $\mathbf{e} \leftarrow \text{dGaussian}_\sigma(\text{seed}_e) \in \mathcal{R}^k$
- 6: $\mathbf{b} = -\mathbf{A}^\top \cdot \mathbf{s} + \mathbf{e} \in \mathcal{R}_q^k$
- 7: **return** $\text{pk} = (\text{seed}_A, \mathbf{b}), \text{sk} = \mathbf{s}$



Enc(pk, μ ; seed_r):

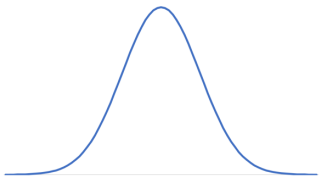
- 1: $\mathbf{A} = \text{expandA}(\text{seed}_A)$
- 2: **if** seed_r is not given **then** seed_r $\leftarrow \{0, 1\}^{256}$
- 3: $\mathbf{r} \leftarrow \text{HWT}_{h_r}(\text{seed}_r) \in S_\eta^k$
- 4: $\mathbf{c}_1 = \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{r} \rfloor \in \mathcal{R}_p^k$
- 5: $c_2 = \lfloor p'/q \cdot \langle \mathbf{b}, \mathbf{r} \rangle + p'/t \cdot \mu \rfloor \in \mathcal{R}_{p'}$
- 6: **return** $\text{ct} = (\mathbf{c}_1, c_2)$



Side Channel Attack

Discrete Gaussian sampler with $\sigma = 1.0625$

```
int addGaussianError(uint16_t *op, const size_t length, const uint8_t *seed) {  
    ...  
    for (size_t i = 0; i < length; i += 64) {  
        unsigned64_t s[2];  
  
        s[0] = (난수 배열 x의 비트 연산)  
        s[1] = (난수 배열 x의 비트 연산)  
  
        for (size_t k = 0; k < 64; ++k) {  
            op[i + k] = ((s[0] >> k) & 0x01) | (((s[1] >> k) & 0x01) << 1);  
            uint16_t sign = (x[9] >> k) & 0x01;  
            op[i + k] = (((-sign) ^ op[i + k]) + sign) << _16_LOG_Q;  
        }  
    }  
    return 0;  
}
```



SMAUG-128 : intermediate value {-3,-2,-1,0,1,2,3}

Side Channel Attack

Discrete Gaussian sampler with $\sigma = 1.0625$

$(-sign)^{op \rightarrow coeffs[i+k]+sign}$

$0xFFFD = \{111 \dots 11101\}_2$

$0xFFFE = \{111 \dots 11110\}_2$

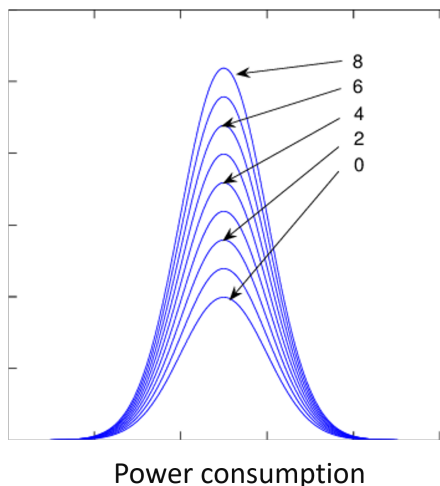
$0xFFFF = \{111 \dots 11111\}_2$

$0x0000 = \{000 \dots 00000\}_2$

$0x0001 = \{000 \dots 00001\}_2$

$0x0002 = \{000 \dots 00010\}_2$

$0x0003 = \{000 \dots 00011\}_2$



Side Channel Attack

Discrete Gaussian sampler with $\sigma = 1.0625$

$(-sign)^{op} \rightarrow coeffs[i+k] + sign$

$0xFFFF = \{111 \dots 11101\}_2$

$0xFFFE = \{111 \dots 11110\}_2$

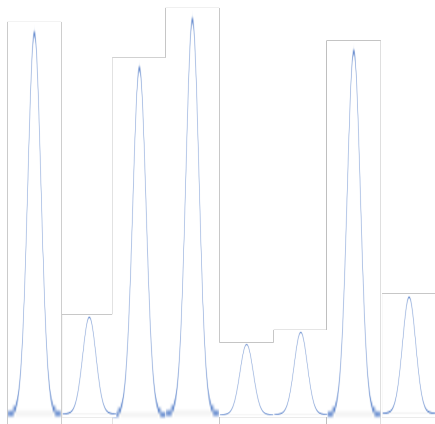
$0xFFFF = \{111 \dots 11111\}_2$

$0x0000 = \{000 \dots 00000\}_2$

$0x0001 = \{000 \dots 00001\}_2$

$0x0002 = \{000 \dots 00010\}_2$

$0x0003 = \{000 \dots 00011\}_2$



Side Channel Attack

Discrete Gaussian sampler with $\sigma = 1.0625$

$(-sign)^{op \rightarrow coeffs[i+k] + sign}$

$0xFFFFD = \{111 \dots 11101\}_2$

$0xFFFFE = \{111 \dots 11110\}_2$

$0xFFFFF = \{111 \dots 11111\}_2$



**Leftshift
OR
Addition**

$0x0004 = \{000 \dots 00100\}_2$

$0x0005 = \{000 \dots 00101\}_2$

$0x0006 = \{000 \dots 00110\}_2$

$0x0000 = \{000 \dots 00000\}_2$

$0x0001 = \{000 \dots 00001\}_2$

$0x0002 = \{000 \dots 00010\}_2$

$0x0003 = \{000 \dots 00011\}_2$

$0x0000 = \{000 \dots 00000\}_2$

$0x0001 = \{000 \dots 00001\}_2$

$0x0002 = \{000 \dots 00010\}_2$

$0x0003 = \{000 \dots 00011\}_2$

Side Channel Attack

Discrete Gaussian sampler with $\sigma = 1.0625$

- Applying Fisher-Yates Shuffle

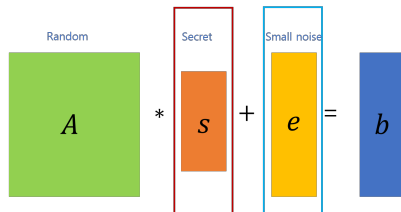
Schemes	TiMER	SMAUG-T128	SMAUG-T192	SMAUG-T256
Original	65839	66491	123522	220780
Countermeasure	116546	117152	226058	472838
Overhead	77.0%	76.2%	83.0%	114.2%

Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz

Side Channel Attack

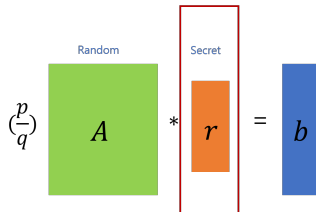
KeyGen(1^λ):

- 1: $\text{seed} \leftarrow \{0, 1\}^{256}$
- 2: $(\text{seed}_A, \text{seed}_{sk}, \text{seed}_e) \leftarrow \text{XOF}(\text{seed})$
- 3: $\mathbf{A} \leftarrow \text{expandA}(\text{seed}_A) \in \mathcal{R}_q^{k \times k}$
- 4: $\mathbf{s} \leftarrow \text{HWT}_{h_s}(\text{seed}_{sk}) \in S_\eta^k$
- 5: $\mathbf{e} \leftarrow \text{dGaussian}_\sigma(\text{seed}_e) \in \mathcal{R}^k$
- 6: $\mathbf{b} = -\mathbf{A}^\top \cdot \mathbf{s} + \mathbf{e} \in \mathcal{R}_q^k$
- 7: **return** $\text{pk} = (\text{seed}_A, \mathbf{b})$, $\text{sk} = \mathbf{s}$



Enc(pk, μ ; seed_r):

- 1: $\mathbf{A} = \text{expandA}(\text{seed}_A)$
- 2: **if** seed_r is not given **then** seed_r $\leftarrow \{0, 1\}^{256}$
- 3: $\mathbf{r} \leftarrow \text{HWT}_{h_r}(\text{seed}_r) \in S_\eta^k$
- 4: $\mathbf{c}_1 = \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{r} \rfloor \in \mathcal{R}_p^k$
- 5: $c_2 = \lfloor p'/q \cdot \langle \mathbf{b}, \mathbf{r} \rangle + p'/t \cdot \mu \rfloor \in \mathcal{R}_{p'}$
- 6: **return** $\text{ct} = (\mathbf{c}_1, c_2)$



Side Channel Attack

Form of sparse polynomial

- Sparse polynomial space is S_η which means that coefficient belongs to $-1, 0, 1$
- Sparse polynomial $s(x)$ and $r(x)$ have h_s and h_r of non-zero coefficients

Parameters sets Security level	TiMER 1	SMAUG-T128 1	SMAUG-T192 3	SMAUG-T256 5
n	256	256	256	256
k	2	2	3	5
(q, p)	(1024, 256)	(1024, 256)	(2048, 256)	(2048, 256)
(p', t)	(8, 2)	(32, 2)	(256, 2)	(64, 2)
(h_s, h_r)	(100, 132)	(140, 132)	(198, 151)	(176, 160)
σ	1.453713	1.0625	1.453713	1.0625

Side Channel Attack

Hybrid combination of SampleInBall & CWW sampling [Sen21]

(a) rejection sampling used

HWT_h:

```
1: Initialize  $\mathbf{c} = c_0c_1\dots c_{n-1} = 00\dots 0$ 
2: for  $i$  from  $n - h$  to  $n - 1$  do
3:   while ( $j > i$ ) do ▷  $j \leq i$ 
4:      $j \xleftarrow{\$} \{0, 1, \dots, n - 1\}$ 
5:   end while
6:    $b \leftarrow \{0, 1\}$ 
7:    $c_i := c_j$ 
8:    $c_j := (-1)^b$ 
9: end for
10: return  $\mathbf{c}$ 
```

- Re-seeding is required in some cases
- Number of XOF calls depends on input seed

(b) CWW sampling applied

HWT_h:

```
1: Initialize  $\mathbf{c} = c_0c_1\dots c_{n-1} = 00\dots 0$ 
2: for  $i$  from  $n - h$  to  $n - 1$  do
3:    $s_i \xleftarrow{\$} \{0, 1, \dots, 2^{32} - 1\}$ 
4:    $j \leftarrow \lfloor (i + 1) \cdot s_i / 2^{32} \rfloor$  ▷  $j \leq i$ 
5:    $b \leftarrow \{0, 1\}$ 
6:    $c_i := c_j$ 
7:    $c_j := (-1)^b$ 
8: end for
9: return  $\mathbf{c}$ 
```

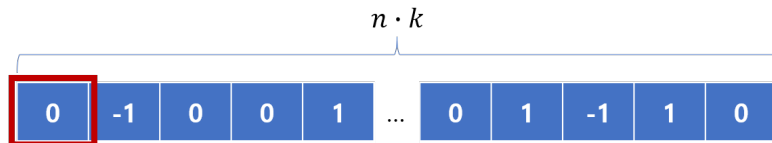
- No additional XOF call required
- Running time doesn't depend on input seed

Side Channel Attack

CWW sampling [Sen21]

SMAUG-T128

- $n = 256$, $k = 2$, $h_r = 132$



- $n_k - h_r$ is number of zeros

Side Channel Attack

CWW sampling [Sen21]

- The probability of not selecting a zero during sampling as $(i+1)$ increases from 381 to 512 is as follows:

$$i + 1 - (2^{32} \bmod (i + 1)) = t, \frac{t}{2^{32}} = e_t,$$

$$\prod_{i=380}^{511} 1 - \frac{1}{i+1} (1 + e_t) = \prod_{i=380}^{511} \left(\frac{i}{i+1} - \frac{e_t}{i+1} \right) \approx$$
$$\prod_{i=380}^{511} \left(\frac{i}{i+1} \right) \left(1 - \sum_{i=380}^{511} \frac{e_t}{i+1} \right)$$

Side Channel Attack

CWW sampling [Sen21]

- The probability of not selecting a zero during sampling as $(i+1)$ increases from 381 to 512 is as follows:

Since $t/(i+1)$ averages to $1/2$,

$$\prod_{i=380}^{511} \left(\frac{i}{i+1}\right) \left(1 - \sum_{i=380}^{511} \frac{e_t}{i+1}\right) \approx \frac{380}{512} \times \left(1 - \frac{2 \times 132}{2^{32}}\right)$$

Consequently, the probability of a number being filled in array 0 (the probability of being sampled as -1 or 1) is approximately as follows:

$$\frac{132}{512} + 2^{-28}$$

Side Channel Attack

CWW sampling [Sen21]

- Countermeasure

- It was shown that this bias does not affect security(BIKE)
- Modify the sampling algorithm(implementation code) to eliminate bias

```
HWTh(seed):▷ seed ∈ {0, 1}256  
1: idx = 0  
2: (buf, rand) ← XOF(seed)  
3: for i from n - h to n - 1 do  
4:   div = 0xffffffff / i  
5:   remain = 0xffffffff - div * i  
6:   remain = remain + 1▷ buf[idx] ∈ {0, 1}32  
7:   if then : 0xffffffff - buf[idx] ≥ remain  
8:     degree = buf[idx] / i  
9:     res[i] = res[degree]  
10:    res[degree] = (-1)rand[idx]▷ rand[idx] ∈ {0, 1}  
11:    idx = idx + 1  
12: return convToIdx(res)▷ Storing the indexes
```

TiMER : 4th parameter for IoT

SMAUG + TiGER??

- SMAUG

- **PK** : Module-LWE / **CT** : Module-LWR
- Tightly set parameters

Parameters sets Security level	SMAUG-128	SMAUG-192	SMAUG-256
n	256	256	256
k	2	3	5
(q, p, p', t)	(1024, 256, 32, 2)	(2048, 256, 256, 2)	(2048, 256, 64, 2)
(h_s, h_r)	(140, 132)	(198, 151)	(176, 160)
σ	1.0625	1.453713	1.0625

- TiGER

- **PK** : Ring-LWR / **CT** : Ring-LWE
- Setting q to a small value, and using ECC(Xef) and D2 code
 - Need an extra message space

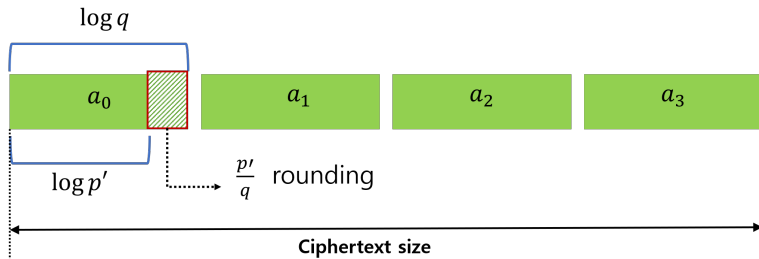
TiMER : 4th parameter for IoT

TiMER

- SMAUG-T with an even smaller ciphertext

- $\text{ECC}(\text{Xef})$: Small q
- D2 encoding : Small p'

$$n = 4 \quad a_3x^3 + a_2x^2 + a_1x + a_0$$



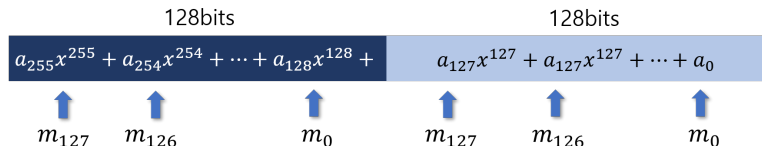
TiMER : 4th parameter for IoT

TiMER

- SMAUG-T with an even smaller ciphertext
 - D2 encoding : Encoding from one message bit to two coefficients
 - Changed the error bound from $q/4$ to $q/2$
 - Reduced randomness space and entropy for μ , $\{0, 1\}^{256} \rightarrow \{0, 1\}^{128}$

SMAUG-T128 with D2

$n = 256$



TiMER : 4th parameter for IoT

TiMER

- TiMER has slightly different features compared to SMAUG-T128 parameter set:
 - Reduced message space: from $\{0, 1\}^{256}$ to $\{0, 1\}^{128}$ for D2 encoding
 - After decryption, the message adjustment process changed from rounding to D2 decoding

Parameters	MSG-BYTE	$p'(\log_{p'})$	σ	h_s
SMAUG-T128	32	32(5)	1.0625	140
TiMER	16	8(3)	1.453713	100

Conclusion

Reminder:

- It has the advantages of both LWE's security and LWR's efficiency
- Small bandwidth & memory consumption which is suitable to IoT environment

Updated:

- SMAUG-T (with TiMER) is now more fast and securely implemented
- Updated to have resistance against **Side-Channel Attacks**
- SMAUG-T offers **superior performance and bandwidth** compared to other lattice schemes

Conclusion

Reminder:

- It has the advantages of both LWE's security and LWR's efficiency
- Small bandwidth & memory consumption which is suitable to IoT environment

Updated:

- SMAUG-T (with TiMER) is now more fast and securely implemented
- Updated to have resistance against **Side-Channel Attacks**
- SMAUG-T offers **superior performance and bandwidth** compared to other lattice schemes

Thanks 👻

<https://kpgc.cryptolab.co.kr/smaug-t>

References I

- [ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe.
Post-quantum key {Exchange—A} new hope.
In [25th USENIX Security Symposium \(USENIX Security 16\)](#), pages 327–343, 2016.
- [BCD⁺16] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila.
Frodo: Take off the ring! practical, quantum-secure key exchange from lwe.
In [Proceedings of the 2016 ACM SIGSAC conference on computer and communications security](#), pages 1006–1018, 2016.
- [BDK⁺18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé.
Crystals-kyber: a cca-secure module-lattice-based kem.
In [2018 IEEE European Symposium on Security and Privacy \(EuroS&P\)](#), pages 353–367. IEEE, 2018.
- [CHK⁺16] Jung Hee Cheon, Kyoohyung Han, Jinsu Kim, Changmin Lee, and Yongha Son.
A practical post-quantum public-key cryptosystem based on spLWE.
In [International Conference on Information Security and Cryptology](#), pages 51–74. Springer, 2016.

References II

- [CKLS18] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song.
Lizard: Cut off the tail! a practical post-quantum public-key encryption from lwe and lwr.
In International Conference on Security and Cryptography for Networks, pages 160–177. Springer, 2018.
- [DKSRV18] Jan-Pieter D’Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren.
Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem.
In International Conference on Cryptology in Africa, pages 282–305. Springer, 2018.
- [FO99] Eiichiro Fujisaki and Tatsuaki Okamoto.
Secure integration of asymmetric and symmetric encryption schemes.
In Michael Wiener, editor, Advances in Cryptology — CRYPTO’ 99, pages 537–554, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
- [HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz.
A modular analysis of the fujisaki-okamoto transformation.
In Yael Kalai and Leonid Reyzin, editors, Theory of Cryptography, pages 341–371, Cham, 2017. Springer International Publishing.

References III

- [LKL⁺18] Joohee Lee, Duhyeong Kim, Hyungkyu Lee, Younho Lee, and Jung Hee Cheon. Rlizard: Post-quantum key encapsulation mechanism for iot devices. [IEEE Access](#), 7:2080–2091, 2018.
- [LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In [Cryptographers' Track at the RSA Conference](#), pages 319–339. Springer, 2011.
- [LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In [Annual international conference on the theory and applications of cryptographic techniques](#), pages 1–23. Springer, 2010.
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In [Proceedings of the thirty-seventh annual ACM symposium on Theory of computing](#), pages 84–93, 2005.
- [Sen21] Nicolas Sendrier. Secure sampling of constant-weight words – application to bike. [Cryptology ePrint Archive](#), Paper 2021/1631, 2021. <https://eprint.iacr.org/2021/1631>.

References IV

- [SXY18] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, Advances in Cryptology – EUROCRYPT 2018, pages 520–551, Cham, 2018. Springer International Publishing.