TiGER

Tiny bandwidth KEM for easy miGration based on RLWE(R)

Seunghwan Park, Chi-Gon Jung, Aesun Park,
Joongeun Choi, and Honggoo Kang

Contents

01

Introduction

Post-Quantum Cryptography

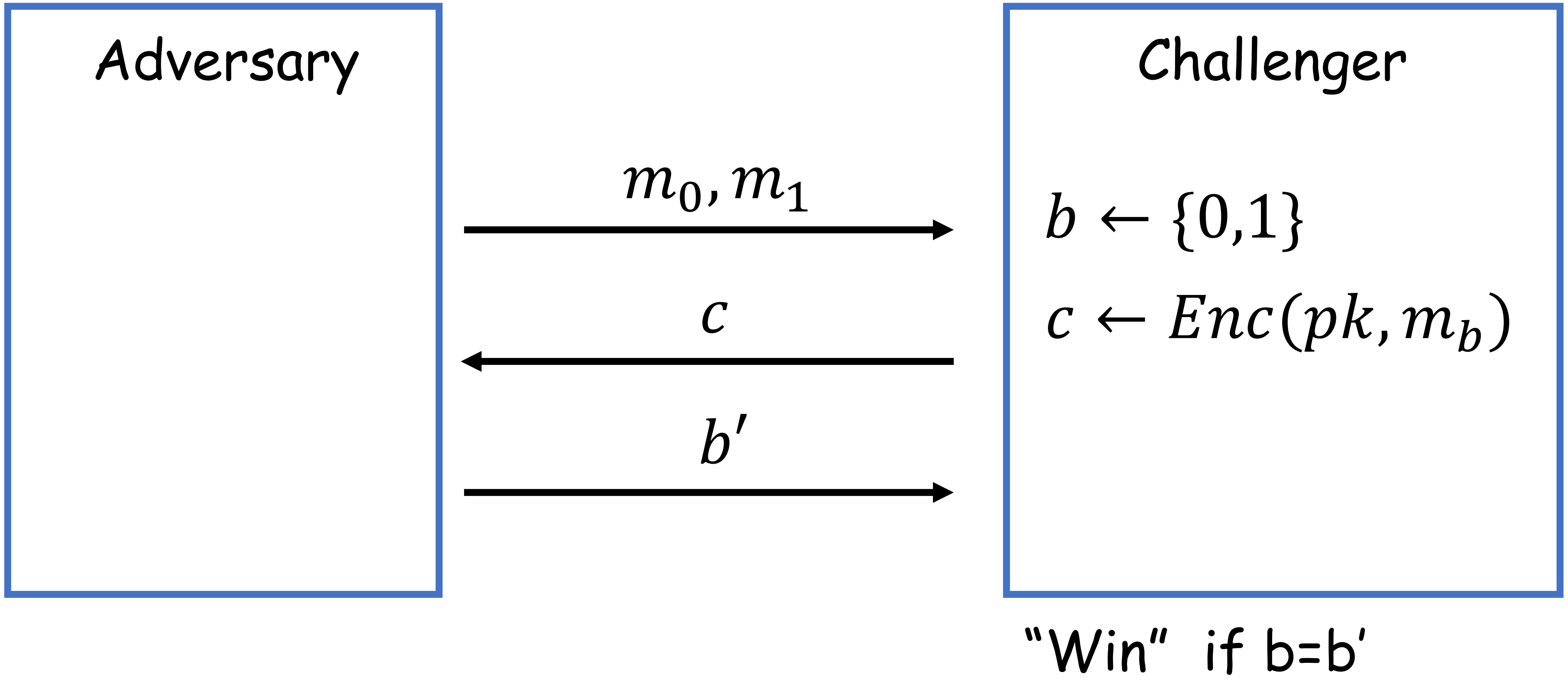
☐ Post-Quantum Cryptography

- **Lattice-based**
- **Code-based**
- Multi-variate
- Hash-based
- **Isogeny-based**

Background

☐ Provable Security

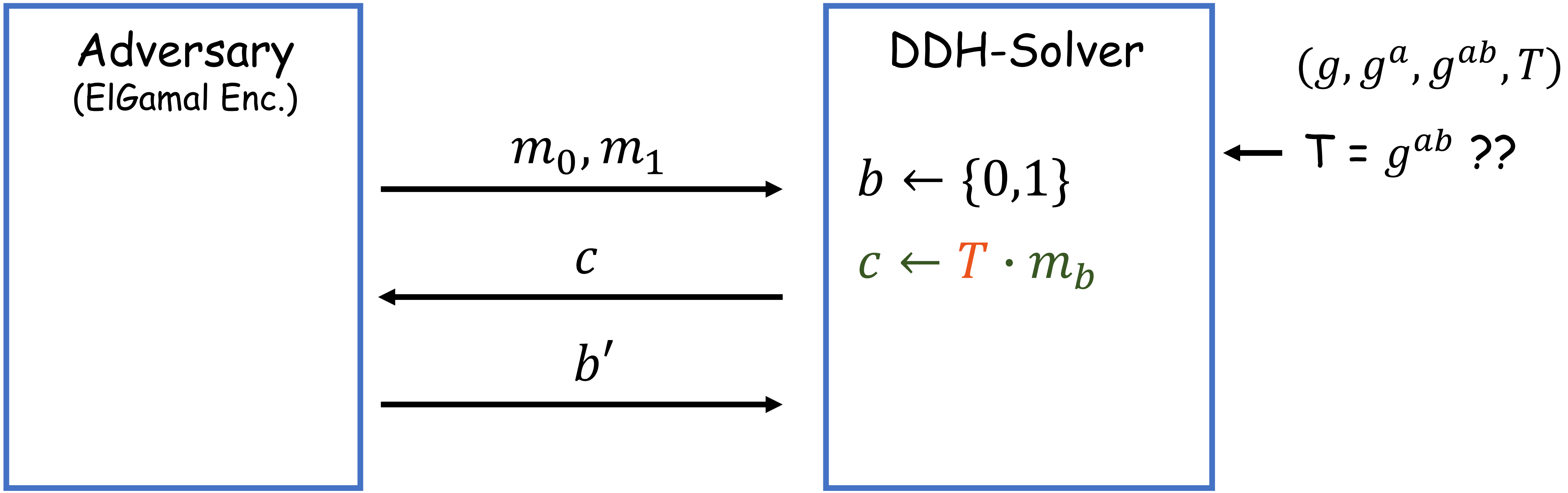
- Hard Problem & Public Key Encryption



Background

☐ Provable Security

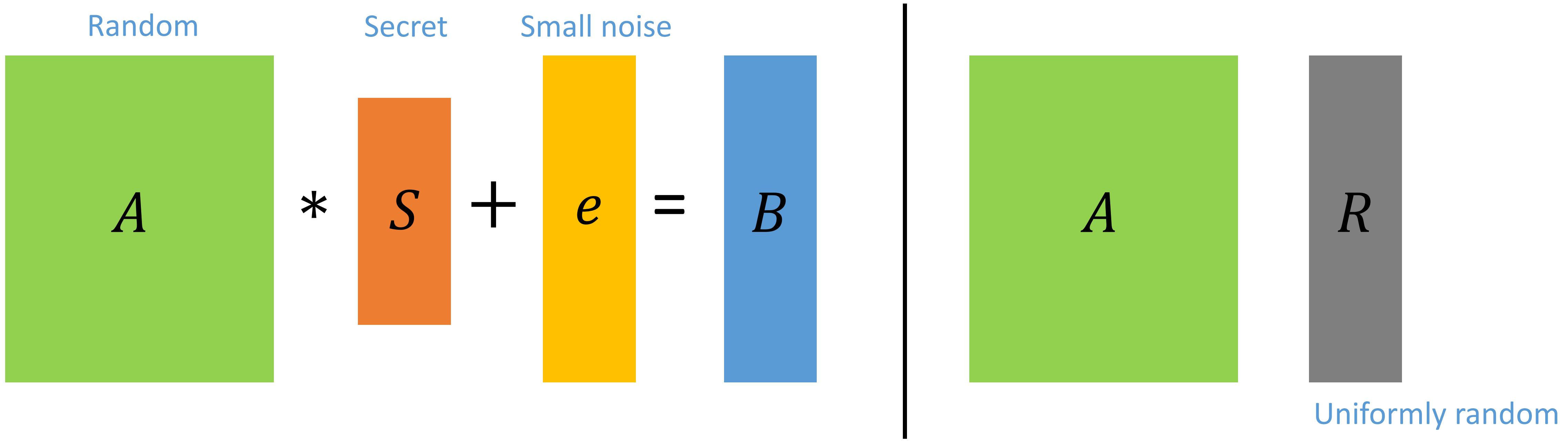
- Hard Problem & Public Key Encryption



Background

Learning-With-Errors(LWE)

- decisional LWE

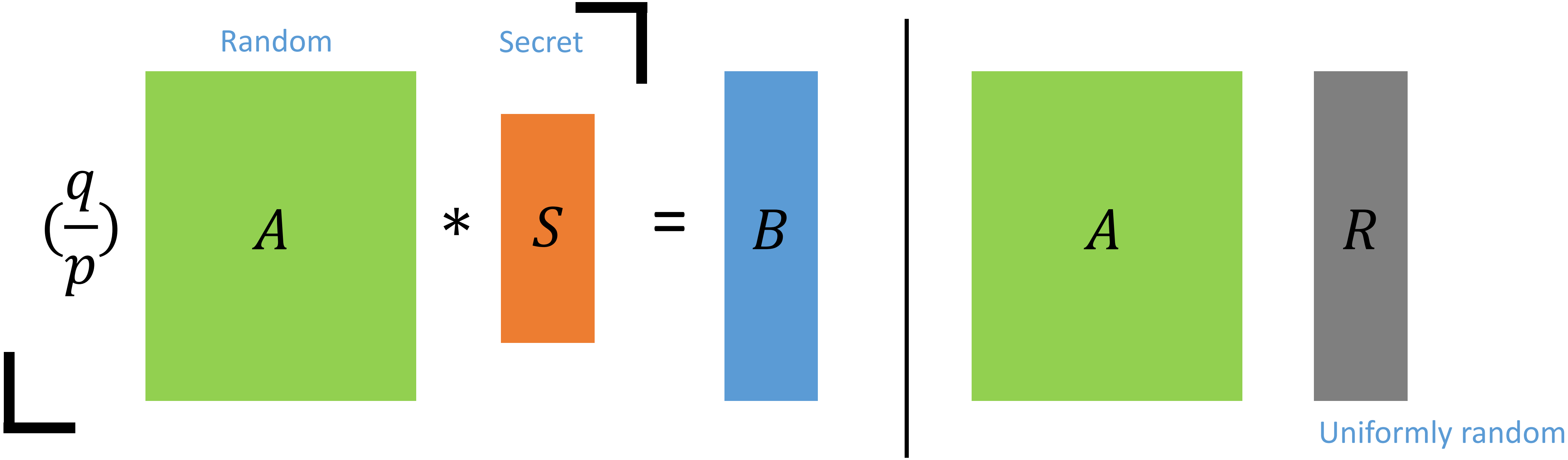


Distinguish (A, B) from (A, R)

Background

☐ Learning-With-Rounding(LWR)

- decisional LWR

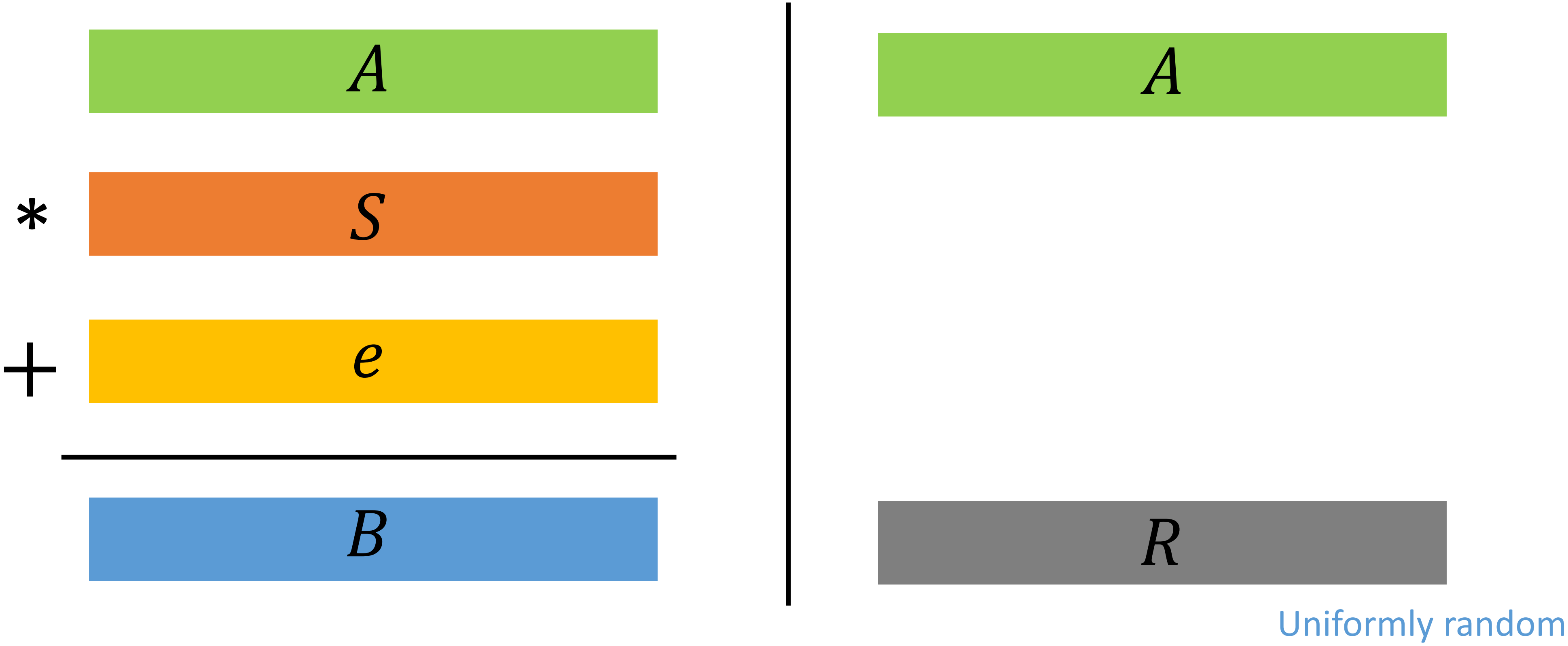


Distinguish (A, B) from (A, R)

Background

☐ Learning-With-Errors(LWE)

- decisional Ring-LWE(RLWE)



Distinguish (A, B) from (A, R)

Background

☐ Learning-With-Errors(LWE)

- decisional Ring-LWE(RLWE)

$q \in \mathbb{Z}, R_q := R/qR = \mathbb{Z}_q[X]/\langle X^4 + 1 \rangle$

	$10x^3 + 11x^2 + 11x + 4$	$10x^3 + 11x^2 + 11x + 4$
$*$	$11x^3 + 11x^2 + 9x + 6$	
$+$	$1x^3 + 1x^2 - 1x + 0$	
	$7x^3 + 10x^2 + 5x + 10$	$r_3x^3 + r_2x^2 + r_1x + r_0$

Uniformly random

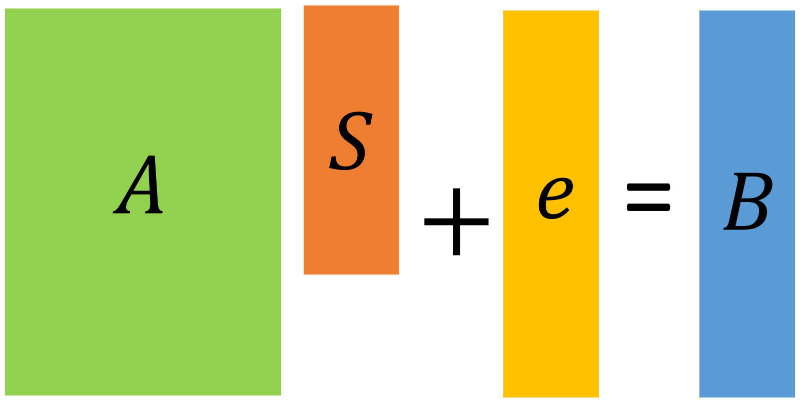
Distinguish (A, B) from (A, R)

Background

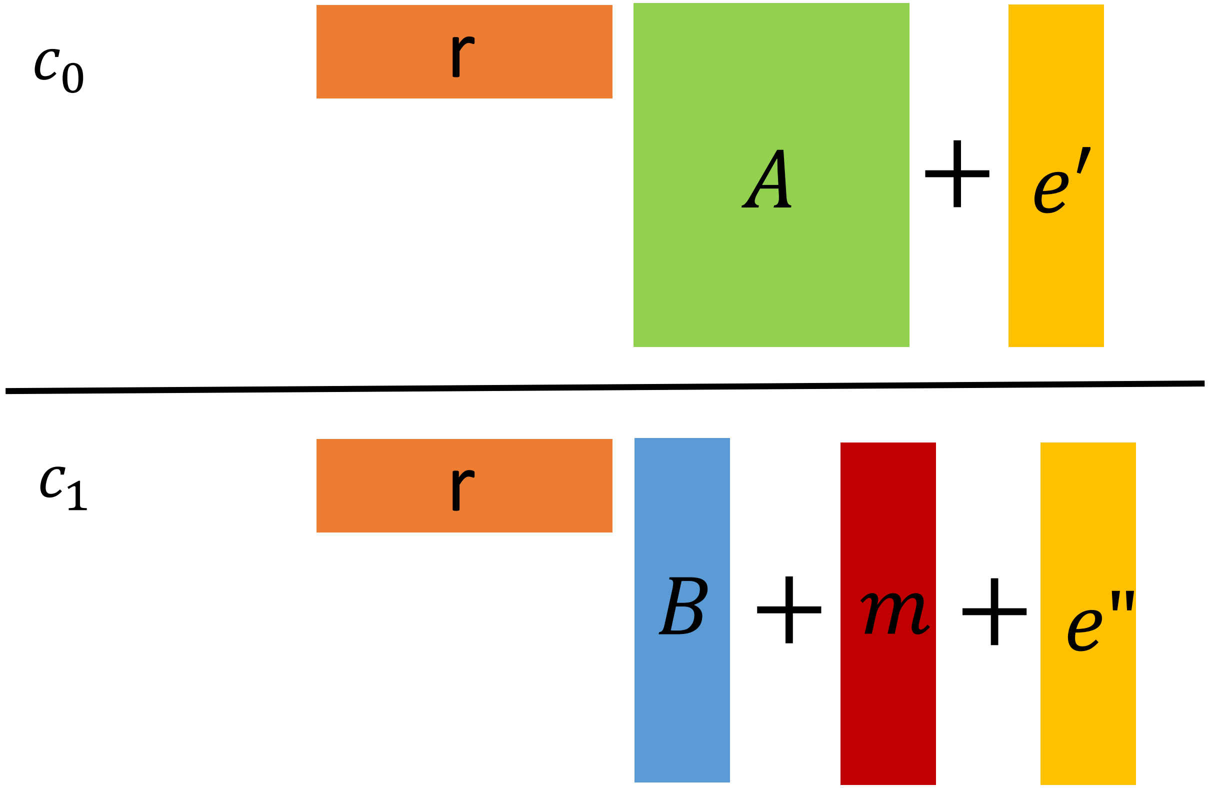
CPA-secure Public-Key Encryption (PKE)

- [LP11]

$$KeyGen(1^{\lambda}) \rightarrow pk =< A, B > \quad sk =< S >$$



$$Encryption(pk, m) \rightarrow \ c =< c_0, c_1 >$$



Background

- CCA-secure Key encapsulation mechanism (KEM)
  - **Fujisaki-Okamoto (FO) transform**.

CPA-PKE

$KeyGen(1^{\lambda}) \rightarrow pk, sk$

$Enc(pk, M) \rightarrow C$

$Dec(sk, C) \rightarrow M$

+

Hash Function

=

CCA-KEM

$KeyGen(1^{\lambda}) \rightarrow pk, sk$

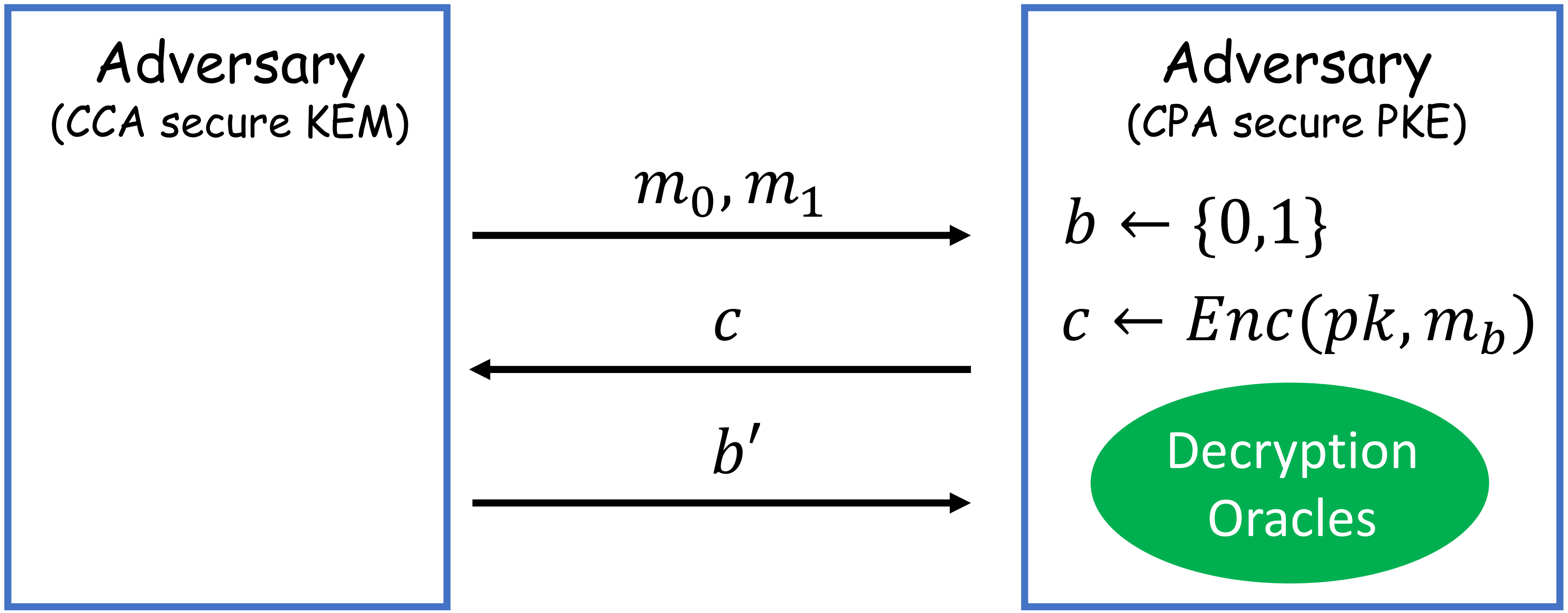
$Enc(pk, M; G(M)) \rightarrow C$

$H(M, C) \rightarrow K$

$Dec(sk, C) \rightarrow K$

Background

- CCA-secure Key encapsulation mechanism (KEM)
  - **Fujisaki-Okamoto (FO) transform**.



Background

☐ Lattice based PKE(or KEM)

- Related works

NewHope
RLWE

KYBER
MLWE

SABER
MLWR

RLizard
RLWE+ RLWR

LAC
RLWE

Round5
RLWR

ThreeBears
I-MLWE

Our Goal

☐ Application of PQC-KEM

- TLS Protocol
- IKEv2 Protocol

☐ Performance of lattice based KEM

Algorithm	Time(s)
RLIZARD-ECDSA-WITH-ARIA-128-CBC-SHA256 (RLizard.KEM)	0.012
RLIZARD-ECDSA-WITH-AES-128-CBC-SHA256 (RLizard.KEM)	0.011
NEWHOPE-ECDSA-WITH-AES-128-CBC-SHA256 (NEWHOPE 12289)	0.012
ECDH-ECDSA-WITH-AES-128-CBC-SHA256 (ECDH25519)	0.009

『Development of lattice-based post-quantum public-key cryptographic schemes』
('20.2.14. / Ewha Womans Univ.)

※ Our goal is to construct lattice based KEM with <u>short-ciphertext</u>.

02

Design Rationale

Design Rationale

Q. How to construct lattice-based KEM with short-ciphertext?

Design Rationale

☐ LWE(R) vs. Module-LWE(R) vs. Ring-LWE(R)

- Size of the public key and the ciphertext (byte)

	Public Key	Ciphertext	Ref.
LWE	$n \times \bar{n} \times \log q/8 + Seed_A$	$\bar{m} \times n \times \log q/8 + \bar{m} \times \bar{n} \times \log q/8$	$\bar{n} = \bar{m}$ =8 (FrodoKEM)
MLWE	$n \times k \times \log q/8 + Seed_A$	$k \times n \times \log q/8 + n \times \log q/8$	k=2,3,4 (Kyber)
RLWE	$n \times \log q/8 + Seed_A$	$n \times \log q/8 + n \times \log q/8$	-

- **Size of pk & ctx :** LWE >= MLWE >= RLWE (depend on parameters)
- **Speed of implementation :** RLWE >= MLWE>=LWE (depend on multiplication)

Design Rationale

☐ LWE(R) vs. Module-LWE(R) vs. **Ring-LWE(R)**

- Size of the public key and the ciphertext (byte)

	Public Key	Ciphertext
RLWE	$n \times \log q/8 + Seed_A$	$n \times \log q/8 + n \times \log q/8$

$$\mathbb{Z}_q[X]/\langle X^4 + 1\rangle$$

$$a_3x^3 + a_2x^2 + a_1x + a_0$$

$$n = 4 \quad \overbrace{\boxed{\quad a_3 \quad}}^{\log q}\ \boxed{\quad a_2 \quad}\ \boxed{\quad a_1 \quad}\ \boxed{\quad a_0 \quad}$$

Design Rationale

☐ LWE(R) vs. Module-LWE(R) vs. Ring-LWE(R)

- Size of the public key and the ciphertext (byte)

	Public Key	Ciphertext
RLWE	$n \times \log q/8 + Seed_A$	$n \times \log q/8 + n \times \log q/8$

$\mathbb{Z}_q[X]/\langle X^4 + 1\rangle$

$a_3x^3 + a_2x^2 + a_1x + a_0$

$\log q$ $q\downarrow \rightarrow$ **ciphertext size ↓**

$n = 4$

a_3	a_2	a_1	a_0

Design Rationale

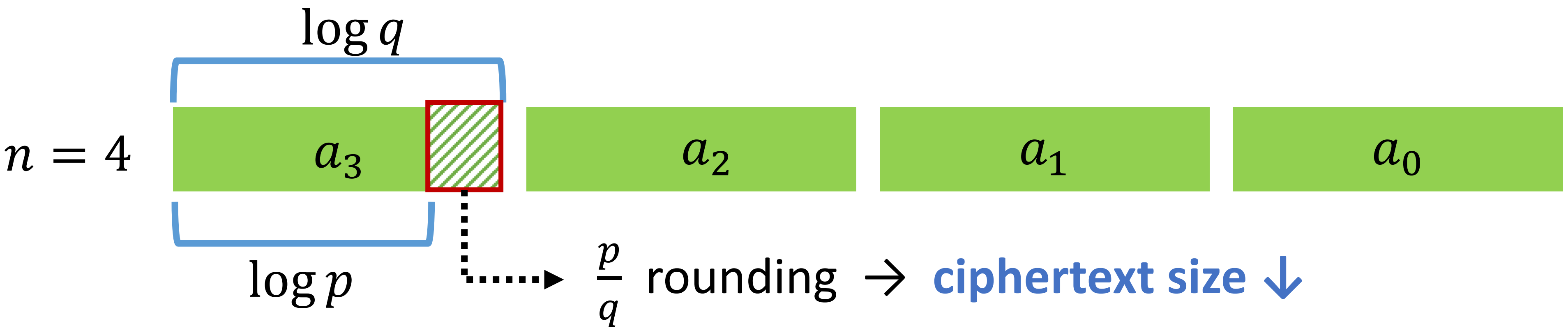
☐ LWE(R) vs. Module-LWE(R) vs. Ring-LWE(R)

- Size of the public key and the ciphertext (byte)

	Public Key	Ciphertext
RLWE	$n \times \log q/8 + Seed_A$	$n \times \log q/8 + n \times \log q/8$

$\mathbb{Z}_q[X]/\langle X^4 + 1\rangle$

$$a_3x^3 + a_2x^2 + a_1x + a_0$$



Design Rationale

☐ only RLWE vs. only RLWR vs. **RLWR+RLWE**

	pk		ctx	
●	RLWE	+	RLWE	: To reduce the size of the ciphertext, the compression function is needed
●	RLWR	+	RLWR	: Parameters setting is difficult & Decryption failure rate ↑
●	**RLWR**	**+**	**RLWE**	: The size of the ciphertext is similar to only RLWR

(Using the compression function)

➢ By adjusting the standard deviation of the noise distribution, difficulties in parameter setting are solved. (& Decryption failure rate ↑)

Design Rationale

☐ Decryption Failure Rate(DFR)

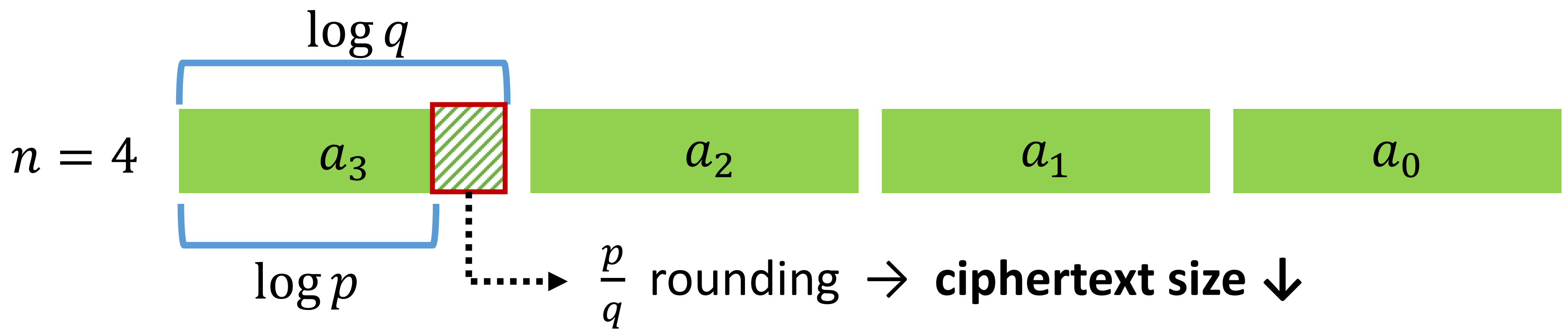
Random | Secret | Small noise

LWE $$A \times S + \boxed{e} = B$$

RLWE $$(a'_3 + \boxed{e_3})x^3 + (a'_2 + \boxed{e_2})x^2 + (a'_1 + \boxed{e_1})x + (a'_0 + \boxed{e_0})$$

Design Rationale

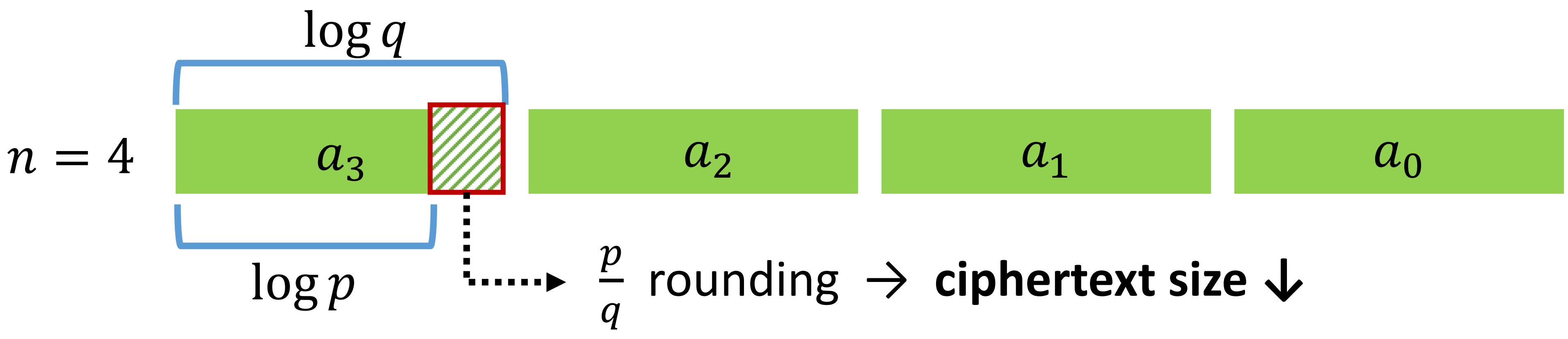
☐ Decryption Failure Rate(DFR)



- q ↓ → ciphertext size ↓ → **Decryption Failure Rate ↑**

Design Rationale

☐ Decryption Failure Rate(DFR)



- q ↓ → ciphertext size ↓ → **Decryption Failure Rate ↑**
- To solve this problem, we use error correction codes **XEf** and **D2**.

Design Rationale

□ Error correction code : XEf + D2

- To solve DFR ↑, we use error correction codes **Xef** and **D2**.
- **XEf** [Round5] : Efficient implementation (600cycles, 5bits correction)

  ➢ TiGER 128 : XE3

  128 bits Msg | 91 bits Red. | 37bit padding

  256bit codeword

  ➢ TiGER 192, 256 : XE5

  256 bits Msg | 234 bits Red. | 22bit padding

  512bit codeword

[Round5] Baan, H., Bhattacharya, S., Fluhrer, S.R., Garcia-Morchon, O., Laarhoven, T., Rietman, R., Saarinen, M.J.O., Tolhuizen, L., Zhang, Z.: Round5: Compact and fast post-quantum public-key encryption. IACR Cryptology ePrint Archive 2019, 90 (2019)

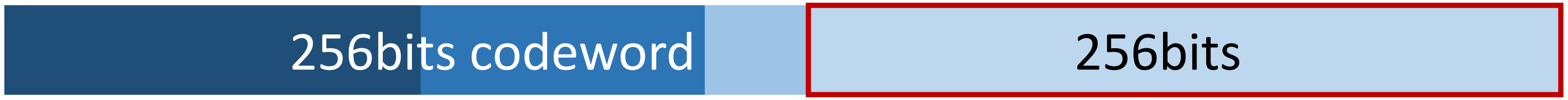
Design Rationale

□ Error correction code : XEf + D2

- To solve DFR ↑, we use error correction codes **Xef** and **D2**.
- **XEf** [Round5] : Efficient implementation (600cycles, 5bits correction)

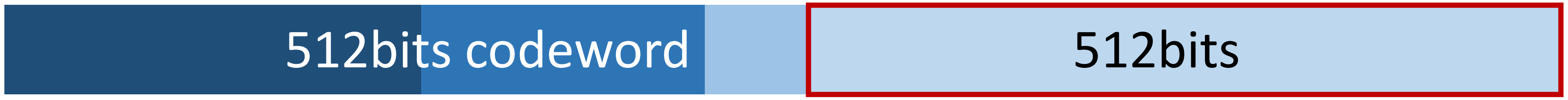
➢ TiGER 128 = XE3 (128bits Msg, 91bit Red, 37bits padding = 256bits codeword)

$n = 512$



➢ TiGER 192, 256 = XE5 (256bits Msg, 234bits Red, 22bit padding = 512bits codeword)

$n = 1024$



[Round5] Baan, H., Bhattacharya, S., Fluhrer, S.R., Garcia-Morchon, O., Laarhoven, T., Rietman, R., Saarinen, M.J.O., Tolhuizen, L., Zhang, Z.: Round5: Compact and fast post-quantum public-key encryption. IACR Cryptology ePrint Archive 2019, 90 (2019)

Design Rationale

□ Error correction code : XEf + D2

- To solve DFR ↑, we use error correction codes **Xef** and **D2**.
- **D2** [Newhope] : Encoding from one message bit to two coefficients
  - ❖ Reduce decryption failure bound from q/4 to q/2

➢ TiGER 128 : XE3 + D2

$n = 512$

➢ TiGER 192, 256 : XE5 + D2

$n = 1024$

[Newhope] Alkim, E., Ducas, L., P¨oppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 327–343 (2016)

Design Rationale

☐ Description

Public Key : **RLWR**

$$SHAKE256(Seed_a, n/8) \rightarrow a$$

$$\lfloor (\frac{q}{p}) \, a * s \rceil = b$$

Ciphertext : **RLWE** + Compression

$$\lfloor (\frac{k_1}{q}) \left(a * r + e_1 \right) \rceil \rightarrow c_1$$

XEf & D2

$$\lfloor (\frac{k_2}{q}) \left(\boldsymbol{eccENC}(M) + b * r + e_1 \right) \rceil \rightarrow c_2$$

Design Rationale

☐ Description

Public Key : RLWR

$SHAKE256(Seed_a, n/8) \rightarrow a$

$$\left\lfloor \left(\frac{q}{p}\right) a * s \right\rceil = b$$

Parameters

$R_q \coloneqq \mathbb{Z}_q[X]/\langle X^n + 1\rangle$, where n is power of 2.

$\boldsymbol{n = 512, 1024}$ $\quad q = ??\ ,\ \ p = ??$

$k_1 = ??\ ,\ \ k_2 = ??$

Ciphertext : RLWE + Compression

$$\left\lfloor \left(\frac{k_1}{q}\right) \begin{pmatrix} a \\ * \; r \\ + \; e_1 \end{pmatrix} \right\rceil \Rightarrow c_1$$

XEf & D2

$$\left\lfloor \left(\frac{k_2}{q}\right) \begin{pmatrix} \boldsymbol{ecc}(M) + b \\ * \; r \\ + \; e_1 \end{pmatrix} \right\rceil \Rightarrow c_2$$

Design Rationale

☐ All integer modulus are power of 2

- rounding & ctx compress → ADD & AND operation
- Fixed $q = 256$ is a byte size.
  - ➢ Efficient modulo operation and memory usage

	Security	n	q	p	k_1	k_2
TiGER128	AES128	512	256	128	64	16
TiGER192	AES192	1024	256	128	64	4
TiGER256	AES256	1024	256	128	128	4

03

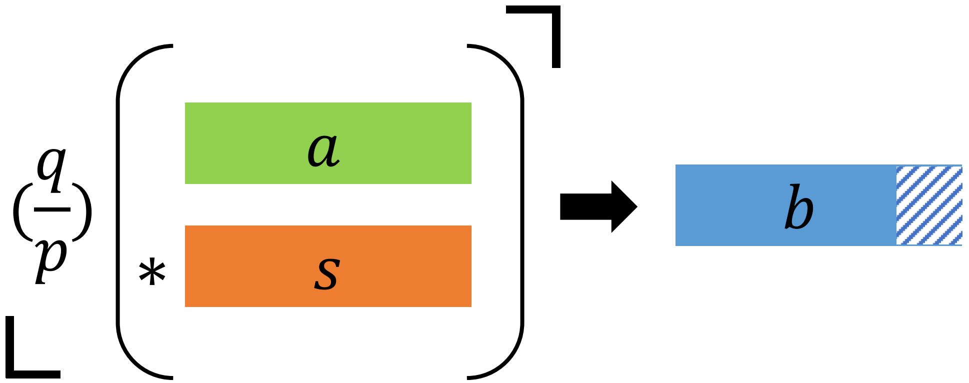
Proposed Scheme

Proposed Scheme

□ PKE_KeyGen

- Input : Security Parameters 1^{λ}
- Output : $pk,\ sk$
  - ➢ $\boldsymbol{a} \leftarrow SHAKE256(Seed_a, n/8)$
  - ➢ $\boldsymbol{s} \leftarrow HWT(h_s, Seed_s)$
  - ➢ $\boldsymbol{b} \leftarrow \lfloor (p/q) \cdot \boldsymbol{a} * \boldsymbol{s} \rceil$

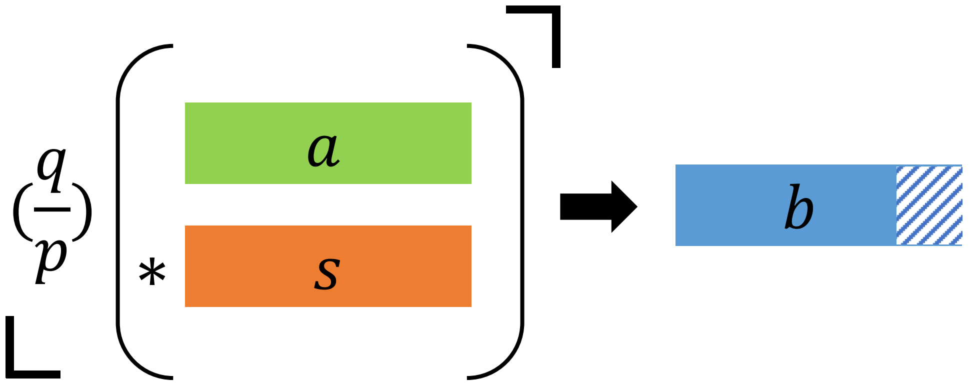
  - ➢ $pk = \langle Seed_a || \boldsymbol{b} \rangle$
  - ➢ $sk = \langle \boldsymbol{s} \rangle$



Proposed Scheme

☐ PKE_KeyGen

- Input : Security Parameters 1^λ
- Output : $pk,\ sk$
  - ➢ $\boldsymbol{a} \leftarrow SHAKE256(Seed_a, n/8)$
  - ➢ $\boldsymbol{s} \leftarrow HWT(h_s, Seed_s)$
  - ➢ $\boldsymbol{b} \leftarrow \lfloor (p/q) \cdot \boldsymbol{a} * \boldsymbol{s} \rceil$
  - ➢ $pk = \langle Seed_a || \boldsymbol{b} \rangle$
  - ➢ $sk = \langle \boldsymbol{s} \rangle$



Size of pk **(TiGER128)**

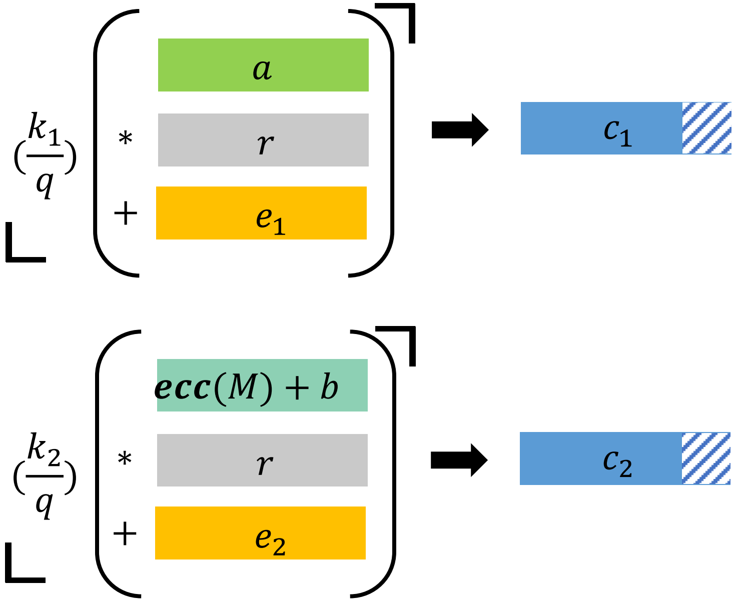
$n = 512,\ q = 256,\ p = 128$

$32 + n \cdot \frac{\log(p)}{\log(q)} = 480$ bytes

Proposed Scheme

☐ PKE_ Encryption

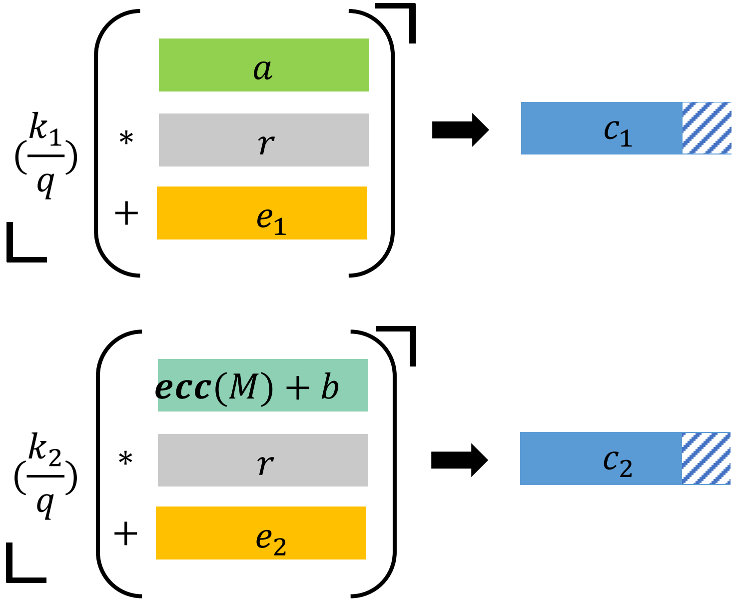
- Input : $pk,\ \boldsymbol{M} \in \{0,1\}^d$
- Output : c
  - ➢ $\boldsymbol{a} \leftarrow SHAKE256(Seed_a, 8/n)$
  - ➢ $\boldsymbol{r} \leftarrow HWT(h_r, w)$
  - ➢ $\boldsymbol{c_1} \leftarrow \lfloor (k_1/q) \cdot (\boldsymbol{a} * \boldsymbol{r}) + \boldsymbol{e_1} \rceil$
  - ➢ $\boldsymbol{c_2} \leftarrow \lfloor (k_2/q) \cdot (\left(\frac{q}{2}\right) \cdot eccENC(\boldsymbol{M}) + ((\frac{q}{p}) \cdot \boldsymbol{b}) * \boldsymbol{r} + \boldsymbol{e_2}) \rceil$
  - ➢ $c = \langle \boldsymbol{c_1} || \boldsymbol{c_2} \rangle$



Proposed Scheme

☐ PKE_ Encryption

- Input : $pk,\ \boldsymbol{M} \in \{0,1\}^d$
- Output : c
  - ➢ $\boldsymbol{a} \leftarrow SHAKE256(Seed_a, 8/n)$
  - ➢ $\boldsymbol{r} \leftarrow HWT(h_r, w)$
  - ➢ $\boldsymbol{c_1} \leftarrow \lfloor (k_1/q) \cdot (\boldsymbol{a} * \boldsymbol{r}) + \boldsymbol{e_1} \rceil$
  - ➢ $\boldsymbol{c_2} \leftarrow \lfloor (k_2/q) \cdot (\left(\frac{q}{2}\right) \cdot eccENC(\boldsymbol{M}) + ((\frac{q}{p}) \cdot \boldsymbol{b}) * \boldsymbol{r} + \boldsymbol{e_2}) \rceil$
  - ➢ $c = \langle \boldsymbol{c_1} || \boldsymbol{c_2} \rangle$



Size of ctx (TiGER128)

$n = 512,\ q = 256,\ p = 128,\ k_1 = 64,\ k_2 = 16$

$$n \cdot \frac{\log(k_1)}{\log(q)} + n \cdot \frac{\log(k_2)}{\log(q)} = 384 + 256 = 640 \text{ bytes}$$

Proposed Scheme

□ PKE_ Decryption

- Input : sk, c
- Output : $\boldsymbol{M}$
  - ➢ $\widehat{\boldsymbol{M}} \leftarrow \lfloor (2/q) \cdot ((\frac{q}{k_2}) \cdot \boldsymbol{c_2} - ((\frac{q}{k_1}) \cdot \boldsymbol{c_1}) * \boldsymbol{s} \rceil$
  - ➢ $\boldsymbol{M} = eccDec(\widehat{\boldsymbol{M}})$

Proposed Scheme

☐ KEM_KeyGen

- Input : Security Parameters 1^{λ}
- Output : $pk,\ sk$
  - $pk, sk_{PKE} \leftarrow \mathbf{PKE\_KeyGen}(1^{\lambda})$
  - $\boldsymbol{u} \leftarrow R_2$
  - $pk = \langle Seed_a || \boldsymbol{b} \rangle$
  - $sk = \langle sk_{PKE} || \boldsymbol{u} \rangle$

Proposed Scheme

☐ KEM_ Encryption

- Input : pk
- Output : $c, \boldsymbol{K}$
  - ➢ $\delta \in \{0,1\}^d$
  - ➢ $c \leftarrow$ **PKE_Encryption**$(pk, \delta; H(\delta, H(pk)))$
  - ➢
  - ➢ $\boldsymbol{K} = G(H(c), \delta)$

Proposed Scheme

☐ KEM_ Decryption

- Input : pk, sk, c
- Output : $\boldsymbol{K}$
  - ➢ $\hat{\delta} \leftarrow \textbf{PKE\_Decryption}(sk_{PKE}, c)$
  - ➢ $\hat{\boldsymbol{c}} \leftarrow \textbf{PKE\_Encryption}(pk, \hat{\delta}; H(\hat{\delta}, H(pk)))$
  - ➢ **if** $c = \hat{c}$ **then** $\boldsymbol{K} \leftarrow G(H(c), \delta)$ **else** $\boldsymbol{K} \leftarrow G(H(c), u)$

Proposed Scheme

☐ Parameters and Size of pk, sk, and ctx

Table 1: The detail parameters for each security level

parameters	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

Table 2: Size of pk, sk, and ciphertext (bytes)

parameters	Ciphertext	Public key	Secret key*
TiGER128	640	480	528
TiGER192	1,024	928	1,056
TiGER256	1,152	928	1,056

Proposed Scheme

☐ Parameters and Size of pk, sk, and ctx

Table 1: The detail parameters for each security level

parameters	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

Table 2: Size of pk, sk, and ciphertext (bytes)

parameters	Ciphertext	Public key	Secret key*
TiGER128	640	480	
TiGER192	1,024	928	
TiGER256	1,152	928	

Scheme	ctx	pk
Kyber1024	1568B	1568B
FireSaber	1472B	1312B
LizarMong	1280B	1056B

Proposed Scheme

□ Decryption Failure Rate

- error rate $\hat{\varepsilon} = 1 - \Pr[-\frac{q}{2} < \{(e'_b r + e'_2 + e_{c2}) - (e'_1 s + e'_{c1})\} < \frac{q}{2}]$
- The error rate of each message bit is $2^{-44.28}$ on TiGER128
- Using XEf to correct 3-bits error, decryption failure rate is

$$\epsilon = 1 - (\sum_{f=0}^{3} \binom{512}{f} \cdot ((2^{-44.28})^{f}) \cdot (1 - 2^{-44.28})^{512-f}) \approx 2^{-145.75}$$

	Bit error rate	DFR	f
TiGER128	$2^{-44.28}$	$2^{-145.75}$	3
TiGER192	$2^{-33.48}$	$2^{-150.41}$	5
TiGER256	$2^{-41.96}$	$2^{-201.29}$	5

04

Security

Security

- **Theorem 1 (IND-CPA PKE).** *The above PKE scheme is secure under chosen plaintext attacks if the the RLWE assumption and the RLWR assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that* $\mathbf{Adv}_{PKE}^{IND-CPA}(\mathcal{A}) \leq \mathbf{Adv}_{n,q}^{RLWE}(\mathcal{B}) + \mathbf{Adv}_{n,q,p}^{RLWR}(\mathcal{B})$.

$$pk = \langle Seed_a || \boldsymbol{b} = \lfloor (p/q) \cdot \boldsymbol{a} * \boldsymbol{s} \rceil \rangle$$

$\Updownarrow$ Decisional RLWR problem

$$pk = \langle Seed_a || \boldsymbol{b} \leftarrow \boldsymbol{R_p} \rangle$$

In the random oracle model, $\boldsymbol{a} \leftarrow H(Seed_a)$.

Security

- **Theorem 1 (IND-CPA PKE).** *The above PKE scheme is secure under chosen plaintext attacks if the the RLWE assumption and the RLWR assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that* $\boldsymbol{Adv}_{PKE}^{IND-CPA}(\mathcal{A}) \leq \boldsymbol{Adv}_{n,q}^{RLWE}(\mathcal{B}) + \boldsymbol{Adv}_{n,q,p}^{RLWR}(\mathcal{B})$.

$$\boldsymbol{c_1} \leftarrow \lfloor (k_1/q) \cdot (\boldsymbol{a} * \boldsymbol{r}) + \boldsymbol{e_1} \rceil$$

$$\boldsymbol{c_2} \leftarrow \lfloor (k_2/q) \cdot \left(\left(\frac{q}{2}\right) \cdot eccENC(M_b) + \left(\left(\frac{q}{p}\right) \cdot \boldsymbol{b}\right) * \boldsymbol{r} + \boldsymbol{e_2} \right\rceil$$

$\Updownarrow$ Decisional **RLWE** problem

$$\boldsymbol{c_1} \leftarrow \lfloor (k_1/q) \cdot \boldsymbol{u} \rceil$$

$$\boldsymbol{c_2} \leftarrow \lfloor (k_2/q) \cdot \left(\left(\frac{q}{2}\right) \cdot eccENC(M_b) + \boldsymbol{v} \right\rceil$$

Security

- **Theorem 2 (IND-CCA KEM in QROM).** *We define a public key encryption scheme PKE = (KeyGen, Encrypt, Decrypt) with message space $\mathcal{M}$ and which is (1-ϵ)-correct. For any IND-CCA quantum adversary $\mathcal{A}$ that makes at most q_D queries to the decryption oracle, at most q_G queries to the random oracle G and at most q_H queries to the random oracle H, we have that*

$$\boldsymbol{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) \leq 2q_H \frac{1}{\mathcal{M}} + 4q_G\sqrt{1-\epsilon} + 2(q_G + q_H)\sqrt{\boldsymbol{Adv}_{PKE}^{IND-CPA}(\mathcal{B})}.$$

Security

☐ **Analysis of known attacks** (using LATTICE-ESTIMATOR[ASP15])

- Core-SVP [ADPS16] & Meet-Attack [MAY21]

	TiGER Core-SVP	Kyber Core-SVP	NIST req.
AES128	129	118	143
AES192	231	183	207
AES256	261	256	272

- MATZOV [MAT22]

	TiGER MATZOV	Kyber MATZOV	NIST req.
AES128	147	140	143
AES192	246	201	207
AES256	277	270	272

Performance

☐ Performance(CPU cycles)

- 2~2.4x faster than Kyber(ref), 2.6~4.0x faster than LAC(opt)

Algorithm	Key generate	Encapsulation	Decapsulation
TiGER128 (ref)	58,531	74,312	97,258
TiGER192 (ref)	72,661	128,854	151,382
TiGER256 (ref)	88,441	159,665	193,663
Kyber512 (ref[3])	121,721	153,724	189,515
Kyber768 (ref)	217,175	261,818	304,349
Kyber1024 (ref)	308,615	353,579	411,223
LAC128 (opt[4])	138,841	219,415	253,301
LAC192 (opt)	308,557	414,122	638,422
LAC256 (opt)	368,792	595,165	806,561
Kyber512 (AVX2[5])	34,672	47,670	41,675
Kyber768 (AVX2)	59,150	73,523	64,653
Kyber1024 (AVX2)	92,268	121,576	106,296

✓ **Implementation**
AMD Ryzen3 2200G@3.5GHz,
Ubuntu 22.04.1,
GCC 11.3.0 with -O3
Keygen : 100,000
Enc/Dec : 100,000

Conclude

☐ **TiGER : Tiny bandwidth KEM for easy miGration based on RLWE(R)**

- **Keygen**: RLWR, **Enc • Dec**: RLWE / q=256, using ECC (XEf + D2)
- Short Public Key and Ciphertext
- **Achieve the security level** AES128, AES192, and AES256
- **Fast and suitable for SIMD.**

An Optimal KEM for Quantum Resistant Security Protocols

Q&A

THANK YOU!