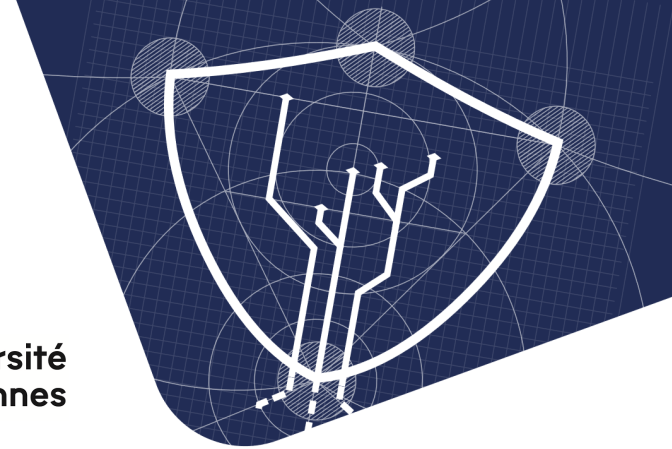


edsi
The Embedded Security Experts

KUDELSKI IOT THINGS

Université de Rennes

April 01, 2025

Post-Quantum Identity-Based Encryption from ML–KEM

Building an IBE scheme from existing blocks

Julien CAM

- **Identity-Based Encryption**
  - Introduction to asymmetric encryption
  - Introduction to Identity-Based Encryption
  - The definition of an IBE scheme
- ID–ML–KEM
- ID–ML–KEM_CKKS19
- Conclusion

Introduction to asymmetric encryption

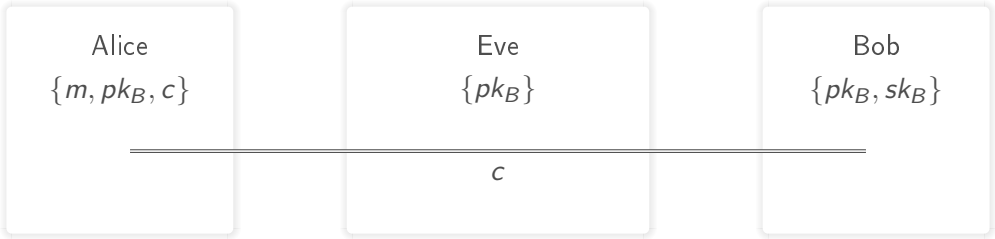
Introduction to asymmetric encryption

Introduction to asymmetric encryption

Introduction to asymmetric encryption

Introduction to asymmetric encryption

Introduction to asymmetric encryption



Introduction to asymmetric encryption

Introduction to asymmetric encryption

Introduction to Identity-Based Encryption

- In a traditional asymmetric encryption scheme, a user chooses her secret key, and computes her public key from it with a one-way function.
- In an IBE scheme, a user chooses her public key, and has to find a secret key associated with it.
- Problem: If Alice can compute a secret key associated with identity `Alice` without prior knowledge, then Eve can do the same and obtain the secret key of Alice.
- Therefore, we need an authority to generate the secret keys for the users.
  This authority, called a PKG (*Private Key Generator*) has to be highly trusted.

Introduction to Identity-Based Encryption

- In a traditional asymmetric encryption scheme, a user chooses her secret key, and computes her public key from it with a one-way function.
- In an IBE scheme, a user chooses her public key, and has to find a secret key associated with it.
- Problem: If Alice can compute a secret key associated with identity `Alice` without prior knowledge, then Eve can do the same and obtain the secret key of Alice.
- Therefore, we need an authority to generate the secret keys for the users.
  This authority, called a PKG (*Private Key Generator*) has to be highly trusted.

Introduction to Identity-Based Encryption

- In a traditional asymmetric encryption scheme, a user chooses her secret key, and computes her public key from it with a one-way function.
- In an IBE scheme, a user chooses her public key, and has to find a secret key associated with it.
- Problem: If Alice can compute a secret key associated with identity `Alice` without prior knowledge, then Eve can do the same and obtain the secret key of Alice.
- Therefore, we need an authority to generate the secret keys for the users.
  This authority, called a PKG (*Private Key Generator*) has to be highly trusted.

Introduction to Identity-Based Encryption

- In a traditional asymmetric encryption scheme, a user chooses her secret key, and computes her public key from it with a one-way function.
- In an IBE scheme, a user chooses her public key, and has to find a secret key associated with it.
- Problem: If Alice can compute a secret key associated with identity `Alice` without prior knowledge, then Eve can do the same and obtain the secret key of Alice.
- Therefore, we need an authority to generate the secret keys for the users.
  This authority, called a PKG (*Private Key Generator*) has to be highly trusted.

The definition of an IBE scheme

Public Recipient's secret PKG's secret

Definition (IBE scheme)

An IBE scheme is made of 4 polynomial-time algorithms:

- $\texttt{Setup}()$: Create the secret key SK and the public key PK of the PKG.
- $\texttt{Extract}(SK, pk)$: Use SK to compute a secret key sk associated with pk.
- $\texttt{Encrypt}(PK, pk, m)$: Encrypt m with the recipient's identity pk.
- $\texttt{Decrypt}(PK, sk, c)$: Use the secret key sk to decrypt the ciphertext c.

such that, for all pk, m and $(PK, SK) = \texttt{Setup}()$:

$$\texttt{Decrypt}(PK, \texttt{Extract}(SK, pk), \texttt{Encrypt}(PK, pk, m)) = m$$

The definition of an IBE scheme

Public Recipient's secret PKG's secret

Definition (IBE scheme)

An IBE scheme is made of 4 polynomial-time algorithms:

- $\texttt{Setup}()$: Create the secret key SK and the public key PK of the PKG.
- $\texttt{Extract}(SK, pk)$: Use SK to compute a secret key sk associated with pk.
- $\texttt{Encrypt}(PK, pk, m)$: Encrypt m with the recipient's identity pk.
- $\texttt{Decrypt}(PK, sk, c)$: Use the secret key sk to decrypt the ciphertext c.

such that, for all pk, m and $(PK, SK) = \texttt{Setup}()$:

$$\texttt{Decrypt}(PK, \texttt{Extract}(SK, pk), \texttt{Encrypt}(PK, pk, m)) = m$$

The definition of an IBE scheme

Public Recipient's secret PKG's secret

Definition (IBE scheme)

An IBE scheme is made of 4 polynomial-time algorithms:

- `Setup`(): Create the secret key SK and the public key PK of the PKG.
- `Extract`(SK, pk): Use SK to compute a secret key sk associated with pk.
- `Encrypt`(PK, pk, m): Encrypt m with the recipient's identity pk.
- `Decrypt`(PK, sk, c): Use the secret key sk to decrypt the ciphertext c.

such that, for all pk, m and $(PK, SK) = \texttt{Setup}()$:

$$\texttt{Decrypt}(PK, \texttt{Extract}(SK, pk), \texttt{Encrypt}(PK, pk, m)) = m$$

The definition of an IBE scheme

Public Recipient's secret PKG's secret

Definition (IBE scheme)

An IBE scheme is made of 4 polynomial-time algorithms:

- $\texttt{Setup}()$: Create the secret key SK and the public key PK of the PKG.
- $\texttt{Extract}(SK, pk)$: Use SK to compute a secret key sk associated with pk.
- $\texttt{Encrypt}(PK, pk, m)$: Encrypt m with the recipient's identity pk.
- $\texttt{Decrypt}(PK, sk, c)$: Use the secret key sk to decrypt the ciphertext c.

such that, for all pk, m and $(PK, SK) = \texttt{Setup}()$:

$$\texttt{Decrypt}(PK, \texttt{Extract}(SK, pk), \texttt{Encrypt}(PK, pk, m)) = m$$

The definition of an IBE scheme

Public Recipient's secret PKG's secret

Definition (IBE scheme)

An IBE scheme is made of 4 polynomial-time algorithms:

- $\texttt{Setup}()$: Create the secret key SK and the public key PK of the PKG.
- $\texttt{Extract}(SK, pk)$: Use SK to compute a secret key sk associated with pk.
- $\texttt{Encrypt}(PK, pk, m)$: Encrypt m with the recipient's identity pk.
- $\texttt{Decrypt}(PK, sk, c)$: Use the secret key sk to decrypt the ciphertext c.

such that, for all pk, m and $(PK, SK) = \texttt{Setup}()$:

$$\texttt{Decrypt}(PK, \texttt{Extract}(SK, pk), \texttt{Encrypt}(PK, pk, m)) = m$$

The definition of an IBE scheme

Public Recipient's secret PKG's secret

Definition (IBE scheme)

An IBE scheme is made of 4 polynomial-time algorithms:

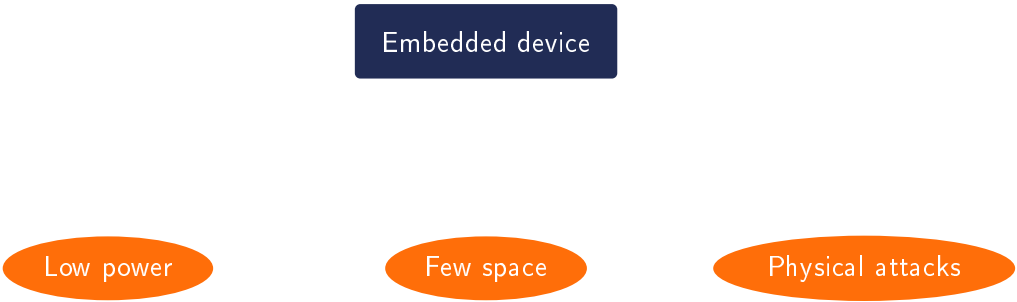
- $\texttt{Setup}()$: Create the secret key SK and the public key PK of the PKG.
- $\texttt{Extract}(SK, pk)$: Use SK to compute a secret key sk associated with pk.
- $\texttt{Encrypt}(PK, pk, m)$: Encrypt m with the recipient's identity pk.
- $\texttt{Decrypt}(PK, sk, c)$: Use the secret key sk to decrypt the ciphertext c.

such that, for all pk, m and $(PK, SK) = \texttt{Setup}()$:

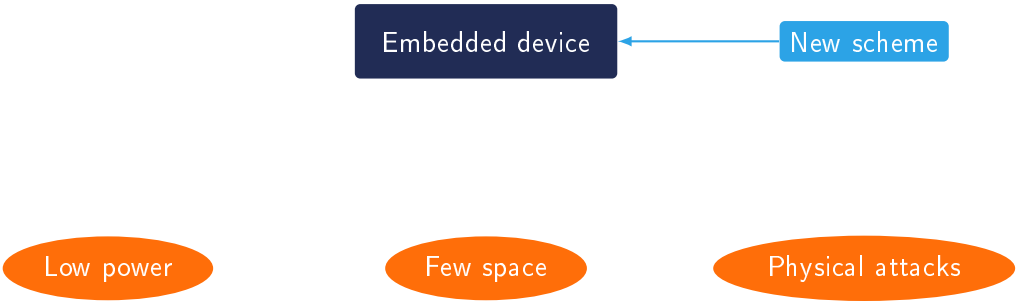
$$\texttt{Decrypt}(PK, \texttt{Extract}(SK, pk), \texttt{Encrypt}(PK, pk, m)) = m$$

- Identity-Based Encryption
- ID–ML–KEM
  - The cost of integrating a new scheme
  - An overview of ML–KEM
  - Inverting the LWE function
  - An overview of an ideal ID–ML–KEM
- ID–ML–KEM_CKKS19
- Conclusion

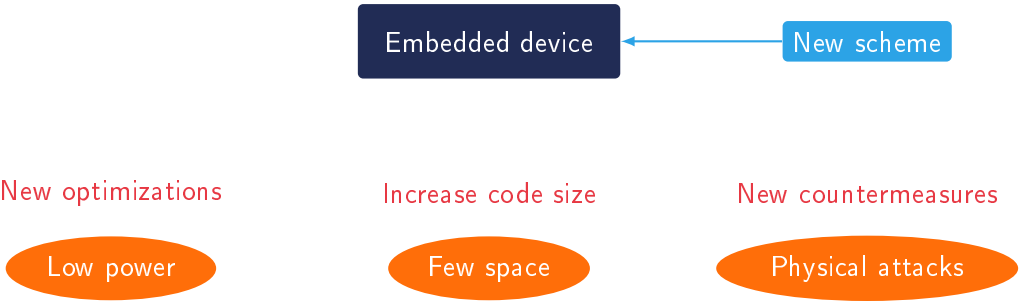
The cost of integrating a new scheme



The cost of integrating a new scheme



The cost of integrating a new scheme



An overview of ML–KEM

KeyGen

Output	PK	$\in R_q^{k \times k}$
Output	pk	$\in R_q^k$
Output	sk	$\in R_q^k$
Such that	$pk - PK \cdot sk$ is small	

Encrypt

Input	PK	$\in R_q^{k \times k}$
Input	pk	$\in R_q^k$
Input	m	$\in R_q$
Output	$u = PK^T r + e_1$	$\in R_q^k$
Output	$v = pk^T r + e_2 + \lfloor q/2 \rceil m$	$\in R_q$
With	$r, e_1 \in R_q^k,\ e_2 \in R_q$ from CBD	

Decrypt

Input	sk	$\in R_q^k$
Input	u	$\in R_q^k$
Input	v	$\in R_q$
Output	$m' = \left\lfloor \frac{v - sk^T u}{q/2} \right\rceil$	$\in R_q$

Inverting the LWE function

- To generate keys with ML–KEM:
  1. Sample a uniform matrix PK
  2. Sample small random sk and e
  3. Compute $pk = PK \times sk + e$

- Finding e and sk from PK and pk is hard (LWE assumption)

- It is however possible knowing some *trapdoor* SK such that:
  - SK has small entries
  - $PK \times SK = 0$

- So, an ID–ML–KEM requires a trapdoor for inverting the LWE function

Inverting the LWE function

- To generate keys with ML–KEM:
  1. Sample a uniform matrix PK
  2. Sample small random sk and e
  3. Compute $pk = PK \times sk + e$

- Finding e and sk from PK and pk is hard (LWE assumption)

- It is however possible knowing some *trapdoor* SK such that:
  - SK has small entries
  - $PK \times SK = 0$

- So, an ID–ML–KEM requires a trapdoor for inverting the LWE function

Inverting the LWE function

- To generate keys with ML–KEM:
  1. Sample a uniform matrix PK
  2. Sample small random sk and e
  3. Compute $pk = PK \times sk + e$

- Finding e and sk from PK and pk is hard (LWE assumption)

- It is however possible knowing some *trapdoor* SK such that:
  - SK has small entries
  - $PK \times SK = 0$

- So, an ID–ML–KEM requires a trapdoor for inverting the LWE function

Inverting the LWE function

- To generate keys with ML–KEM:
  1. Sample a uniform matrix PK
  2. Sample small random sk and e
  3. Compute $pk = PK \times sk + e$

- Finding e and sk from PK and pk is hard (LWE assumption)

- It is however possible knowing some *trapdoor* SK such that:
  - SK has small entries
  - $PK \times SK = 0$

- So, an ID–ML–KEM requires a trapdoor for inverting the LWE function

An overview of ML–KEM

KeyGen

Output	PK	$\in R_q^{k \times k}$
Output	pk	$\in R_q^k$
Output	sk	$\in R_q^k$
Such that	$pk - PK \cdot sk$ is small	

Encrypt

Input	PK	$\in R_q^{k \times k}$
Input	pk	$\in R_q^k$
Input	m	$\in R_q$
Output	$u = PK^T r + e_1$	$\in R_q^k$
Output	$v = pk^T r + e_2 + \lfloor q/2 \rceil m$	$\in R_q$
With	$r, e_1 \in R_q^k,\ e_2 \in R_q$ from CBD	

Decrypt

Input	sk	$\in R_q^k$
Input	u	$\in R_q^k$
Input	v	$\in R_q$
Output	$m' = \left\lfloor \frac{v - sk^T u}{q/2} \right\rceil$	$\in R_q$

An overview of an ideal ID–ML–KEM

Setup

Output $SK \in ?$

Output $PK \in R_q^{k \times k}$

Such that SK is a LWE-trapdoor for PK

Extract

Input $SK \in ?$

Input $pk \in R_q^k$

Output $sk \in R_q^k$

Such that $pk - PK \cdot sk$ is small

Encrypt

Input $PK \in R_q^{k \times k}$

Input $pk \in R_q^k$

Input $m \in R_q$

Output $u = PK^T r + e_1 \in R_q^k$

Output $v = pk^T r + e_2 + \lfloor q/2 \rceil m \in R_q$

With $r, e_1 \in R_q^k,\ e_2 \in R_q$ from CBD

Decrypt

Input $sk \in R_q^k$

Input $u \in R_q^k$

Input $v \in R_q$

Output $m' = \left\lfloor \frac{v - sk^T u}{q/2} \right\rceil \in R_q$

- Identity-Based Encryption
- ID–ML–KEM
- **ID–ML–KEM_CKKS19**
  - The CKKS19 scheme
  - From CKKS19 to ID–ML–KEM_CKKS19
  - The probability of a decryption failure
- Conclusion

An overview of an ideal ID–ML–KEM

Setup

Output $SK \in ?$

Output $PK \in R_q^{k \times k}$

Such that SK is a LWE-trapdoor for PK

Extract

Input $SK \in ?$

Input $pk \in R_q^k$

Output $sk \in R_q^k$

Such that $pk - PK \cdot sk$ is small

Encrypt

Input $PK \in R_q^{k \times k}$

Input $pk \in R_q^k$

Input $m \in R_q$

Output $u = PK^T r + e_1 \in R_q^k$

Output $v = pk^T r + e_2 + \lfloor q/2 \rceil m \in R_q$

With $r, e_1 \in R_q^k,\ e_2 \in R_q$ from CBD

Decrypt

Input $sk \in R_q^k$

Input $u \in R_q^k$

Input $v \in R_q$

Output $m' = \left\lfloor \frac{v - sk^T u}{q/2} \right\rceil \in R_q$

The CKKS19 scheme

Setup

Output	SK	$\in \mathbb{Z}^{(k+1)n \times (k+1)n}$
Output	PK	$\in R_q^k$
Such that	SK is a LWE-trapdoor for PK	

Extract

Input	SK	$\in \mathbb{Z}^{(k+1)n \times (k+1)n}$
Input	pk	$\in R_q$
Output	sk	$\in R_q^k$
Such that	$pk - PK \cdot sk$ is small	

Encrypt

Input	PK	$\in R_q^k$
Input	pk	$\in R_q$
Input	m	$\in R_q$
Output	$u = r \cdot PK + e_1$	$\in R_q^k$
Output	$v = r \cdot pk + e_2 + \lfloor q/2 \rceil m$	$\in R_q$
With	$e_1 \in R_q^k,\ r, e_2 \in R_q$ from $\mathcal{U}_{[\![-1,1]\!]}$	

Decrypt

Input	sk	$\in R_q^k$
Input	u	$\in R_q^k$
Input	v	$\in R_q$
Output	$m' = \left\lfloor \frac{v - sk^T u}{q/2} \right\rceil$	$\in R_q$

From CKKS19 to ID–ML–KEM_CKKS19

- The encryption and decryption follow a structure similar to ML–KEM
- The public keys are not of the form we wanted for ID–ML–KEM
- The parameters are not the ones used by ML–KEM

From CKKS19 to ID–ML–KEM_CKKS19

- The encryption and decryption follow a structure similar to ML–KEM
- The public keys are not of the form we wanted for ID–ML–KEM
- The parameters are not the ones used by ML–KEM

	ID–ML–KEM	CKKS19
Master public key PK	Matrix in $R_q^{2\times 2}$	Vector (h_1, h_2) in R_q^2
User public key pk	Vector in R_q^2	Polynomial $H(\texttt{ID})$ in R_q

From CKKS19 to ID–ML–KEM_CKKS19

- The encryption and decryption follow a structure similar to ML–KEM
- The public keys are not of the form we wanted for ID–ML–KEM
- The parameters are not the ones used by ML–KEM

	ID–ML–KEM	CKKS19
Master public key PK	Matrix in $R_q^{2\times 2}$	Vector (h_1, h_2) in R_q^2
User public key pk	Vector in R_q^2	Polynomial $H(\texttt{ID})$ in R_q

Solution: define $PK = \begin{pmatrix} h_1 & h_2 \\ 0 & 0 \end{pmatrix}$ and $pk = \begin{pmatrix} H(\texttt{ID}) \\ 0 \end{pmatrix}$

From CKKS19 to ID–ML–KEM_CKKS19

- The encryption and decryption follow a structure similar to ML–KEM
- The public keys can be adapted to match our expectations for ID–ML–KEM
- The parameters are not the ones used by ML–KEM

From CKKS19 to ID–ML–KEM_CKKS19

- The encryption and decryption follow a structure similar to ML–KEM
- The public keys can be adapted to match our expectations for ID–ML–KEM
- The parameters are not the ones used by ML–KEM

n	q	$\mathbb{Z}_q$ arithmetic	NTT	`BaseCaseMultiply`	Security
512	524 288	New but easy	???	???	82 bits
256	3 329	ML–KEM	$R_q \simeq (\mathbb{Z}_q^2)^{128}$	ML–KEM	70 bits
256	8 380 417	ML–DSA	$R_q \simeq (\mathbb{Z}_q)^{256}$	ML–DSA	41 bits
512	3 329	ML–KEM	$R_q \simeq (\mathbb{Z}_q^4)^{128}$	New	132 bits
1 024	8 380 417	ML–DSA	$R_q \simeq (\mathbb{Z}_q)^{1024}$	ML–DSA	141 bits

From CKKS19 to ID–ML–KEM_CKKS19

- The encryption and decryption follow a structure similar to ML–KEM
- The public keys can be adapted to match our expectations for ID–ML–KEM
- Some standard parameters can be found: $n = 1\,024$ and $q = 8\,380\,417$

The probability of a decryption failure

Definition

A decryption failure is a case where, for a legitimate master key pair (SK, PK), a message m and a legitimate user public key pk:

$$\texttt{Decrypt}(PK,\ \texttt{Extract}(SK, pk),\ \texttt{Encrypt}(PK, pk, m)) \neq m$$

The probability of a decryption failure

Definition

A decryption failure is a case where, for a legitimate master key pair (SK, PK), a message m and a legitimate user public key pk:

$$\texttt{Decrypt}(PK,\ \texttt{Extract}(SK, pk),\ \texttt{Encrypt}(PK, pk, m)) \neq m$$

$$\texttt{Decrypt}(PK,\ \texttt{Extract}(SK, pk),\ \texttt{Encrypt}(PK, pk, m))$$
$$= m + \left\lfloor \frac{(pk - PK \cdot \texttt{Extract}(SK, pk))^T r + e_2 - \texttt{Extract}(SK, pk)^T e_1}{q/2} \right\rceil$$

The probability of a decryption failure

Definition

A decryption failure is a case where, for a legitimate master key pair (SK, PK), a message m and a legitimate user public key pk:

$$\texttt{Decrypt}(PK,\ \texttt{Extract}(SK, pk),\ \texttt{Encrypt}(PK, pk, m)) \neq m$$

$$\texttt{Decrypt}(PK,\ \texttt{Extract}(SK, pk),\ \texttt{Encrypt}(PK, pk, m))$$
$$= m + \left\lfloor \frac{(pk - PK \cdot \texttt{Extract}(SK, pk))^T r + e_2 - \texttt{Extract}(SK, pk)^T e_1}{q/2} \right\rceil$$

$$\text{Failure} \iff \left\| e_k^T r + e_2 - sk^T e_1 \right\|_\infty \geqslant q/4$$

The probability of a decryption failure

Definition

A decryption failure is a case where, for a legitimate master key pair (SK, PK), a message m and a legitimate user public key pk:

$$\texttt{Decrypt}(PK,\ \texttt{Extract}(SK, pk),\ \texttt{Encrypt}(PK, pk, m)) \neq m$$

$$\texttt{Decrypt}(PK,\ \texttt{Extract}(SK, pk),\ \texttt{Encrypt}(PK, pk, m))$$
$$= m + \left\lfloor \frac{(pk - PK \cdot \texttt{Extract}(SK, pk))^T r + e_2 - \texttt{Extract}(SK, pk)^T e_1}{q/2} \right\rceil$$

$$\text{Failure} \iff \left\| e_k^T r + e_2 - sk^T (e_1 + e_u) + e_v \right\|_\infty \geqslant q/4$$

The probability of a decryption failure

Definition

A decryption failure is a case where, for a legitimate master key pair (SK, PK), a message m and a legitimate user public key pk:

$$\texttt{Decrypt}(PK,\ \texttt{Extract}(SK, pk),\ \texttt{Encrypt}(PK, pk, m)) \neq m$$

$$\text{Failure} \iff \left\| e_k^T r + e_2 - sk^T (e_1 + e_u) + e_v \right\|_\infty \geqslant q/4$$

- e_k and sk are sampled from $\mathcal{N}(0, \sigma^2)$
- e_1, e_2 and r are sampled from Centered Binomial Distributions
- e_u and e_v are errors caused by compression and decompression

Conclusion

- With Identity-Based Encryption, fewer messages are required
- IBE avoids the need to verify certificates (no signature scheme required)
- A scheme based on ML–KEM allows:
  - Relying on some of the most trusted lattice-based assumptions
  - Using code that is already optimized, implemented and protected against physical attacks
- The CKKS19 scheme can be modified to reuse ML–KEM's `Encrypt` and `Decrypt`
- We can found parameters that allows to achieve:
  - Standard level of security (141 bits)
  - Negligible failure probability (2^{-228})

Conclusion

- With Identity-Based Encryption, fewer messages are required
- IBE avoids the need to verify certificates (no signature scheme required)
- A scheme based on ML–KEM allows:
  - Relying on some of the most trusted lattice-based assumptions
  - Using code that is already optimized, implemented and protected against physical attacks
- The CKKS19 scheme can be modified to reuse ML–KEM's `Encrypt` and `Decrypt`
- We can found parameters that allows to achieve:
  - Standard level of security (141 bits)
  - Negligible failure probability (2^{-228})

Conclusion

- With Identity-Based Encryption, fewer messages are required
- IBE avoids the need to verify certificates (no signature scheme required)
- A scheme based on ML–KEM allows:
  - Relying on some of the most trusted lattice-based assumptions
  - Using code that is already optimized, implemented and protected against physical attacks
- The CKKS19 scheme can be modified to reuse ML–KEM's `Encrypt` and `Decrypt`
- We can found parameters that allows to achieve:
  - Standard level of security (141 bits)
  - Negligible failure probability (2^{-228})

Conclusion

- With Identity-Based Encryption, fewer messages are required
- IBE avoids the need to verify certificates (no signature scheme required)
- A scheme based on ML–KEM allows:
  - Relying on some of the most trusted lattice-based assumptions
  - Using code that is already optimized, implemented and protected against physical attacks
- The CKKS19 scheme can be modified to reuse ML–KEM's `Encrypt` and `Decrypt`
- We can found parameters that allows to achieve:
  - Standard level of security (141 bits)
  - Negligible failure probability (2^{-228})

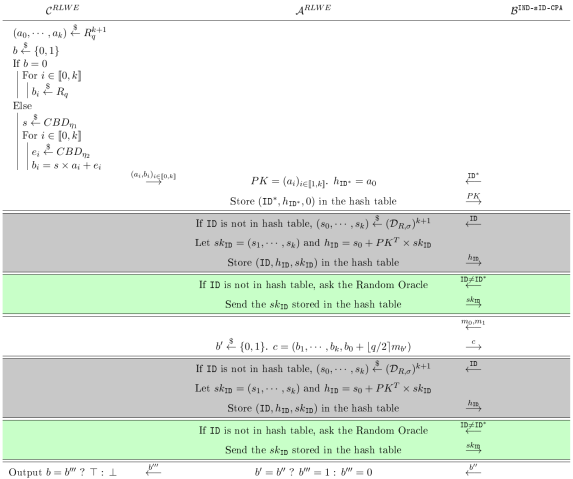
Conclusion

- With Identity-Based Encryption, fewer messages are required
- IBE avoids the need to verify certificates (no signature scheme required)
- A scheme based on ML–KEM allows:
  - Relying on some of the most trusted lattice-based assumptions
  - Using code that is already optimized, implemented and protected against physical attacks
- The CKKS19 scheme can be modified to reuse ML–KEM's `Encrypt` and `Decrypt`
- We can found parameters that allows to achieve:
  - Standard level of security (141 bits)
  - Negligible failure probability (2^{-228})

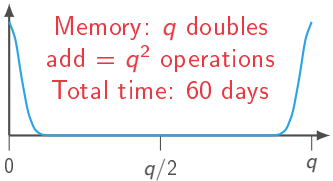
KUDELSKI IOT THINGS

Université de Rennes

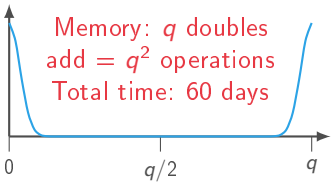
ID–ML–KEM_CKKS19 - Proof of security

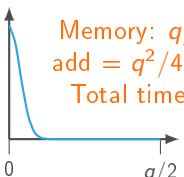


Optimizing the computation of the failure probability

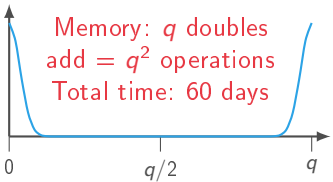


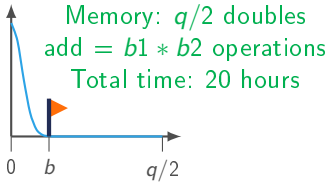
Optimizing the computation of the failure probability

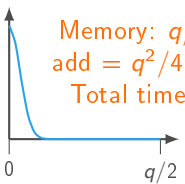




Optimizing the computation of the failure probability







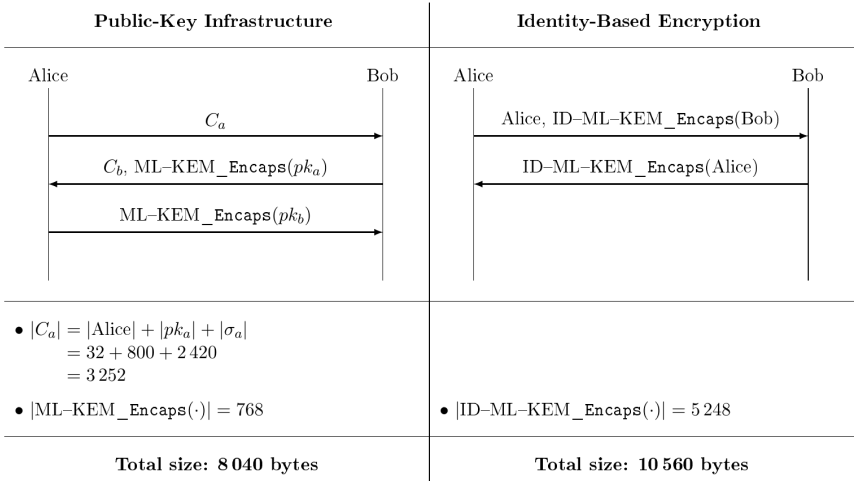
Compression parameters

d_u	23	20	19	18	17	19					18
d_v	4	23				10	5	3	2	1	3
$-\log_2(\mathbb{P}(\text{fail}))$	∞	∞	388	103	20	388	348	228	104	0	61
Size (in Bytes)	6 400	8 064	7 808	7 552	7 296	6 144	5 504	5 248	5 120	4 992	4 992

ID–ML–KEM_CKKS19 - Sizes

Element	Nature	Size (bytes)		
Master secret key	$\mathbb{Z}^{(k+1)n\times(k+1)n}$	$\texttt{sizeof}(\mathbb{Z})\times(k+1)^2\times n^2$	$=$	18 874 368
Master SK seed	$\mathbb{B}^{\lambda/8}$	$\lambda\,/\,8$	$=$	32
Master SK GSO	$\mathbb{R}^{(k+1)n\times(k+1)n}$	$\texttt{sizeof}(\mathbb{R})\times(k+1)^2\times n^2$	$=$	75 497 472
Master public key	T_q^k	$\lceil\log_2(q)\rceil\times n\times k\,/\,8$	$=$	5 888
User secret key	T_q^k	$\lceil\log_2(q)\rceil\times n\times k\,/\,8$	$=$	5 888
User public key	R_q	$\lceil\log_2(q)\rceil\times n\,/\,8$	$=$	2 944
User identity	$\mathbb{B}^{\lambda/8}$	$\lambda\,/\,8$	$=$	32
Ciphertext	$\mathbb{B}^{n(d_uk+d_v)/8}$	$n\times(d_u\times k+d_v)\,/\,8$	$=$	5 248

The communications



APPENDIX

All algorithms

Setup()

Output: Master secret key $SK \in R_q^{3\times 3}$
Output: Master public key $PK \in R_q^2$
1: Sample from $[\mathcal{D}_{\mathbb{Z},\sigma_j}]^n$ the coefficients $f_{i,j}$ of $S = \begin{pmatrix} f_{1,1} & f_{1,2} \\ f_{2,1} & f_{2,2} \\ f_{3,1} & f_{3,2} \end{pmatrix}$ ▷ S_i is S without its i^{th} row
2: $d = (\det(S_1), -\det(S_2), \det(S_3))$
3: Find small F_1, F_2 and F_3 in R such that: $\sum_{i=1}^{3} d_i \cdot F_i = q$ ▷ The MNTRU equation
4: **return** $SK = \begin{pmatrix} f_{1,1} & f_{1,2} & F_1 \\ f_{2,1} & f_{2,2} & F_2 \\ f_{3,1} & f_{3,2} & F_3 \end{pmatrix}$ and $PK = (d_2 d_1^{-1}, d_3 d_1^{-1})$

Extract(SK, pk)

Input: Master secret key $SK \in R_q^{3\times 3}$
Input: User's public key $pk \in R_q$
Output: User's secret key $sk \in R_q^2$
1: **if** sk has already been computed **then**
2: **return** sk ▷ Outputting two sk for the same pk breaks security
3: **else**
4: $\begin{pmatrix} s_0 \\ sk_1 \\ sk_2 \end{pmatrix} = \begin{pmatrix} pk \\ 0 \\ 0 \end{pmatrix} - \texttt{GaussianSampler}\left(SK, \sigma, \begin{pmatrix} pk \\ 0 \\ 0 \end{pmatrix}\right)$
5: **return** $sk = (sk_1, sk_2)$
6: **end if**

Encrypt(PK, pk, m)

Input: Master public key $PK \in R_q^2$
Input: User's public key $pk \in R_q$
Input: Message $m \in R_q$
Output: Ciphertext $c \in R_q^2 \times R_q$
1: Define $A = \begin{pmatrix} PK_1 & PK_2 \\ 0 & 0 \end{pmatrix}$ and $t = \begin{pmatrix} pk \\ 0 \end{pmatrix}$
2: Sample $r \in R_q^2$ from CBD_{η_1}
3: Sample $e_1 \in R_q^2$ from CBD_{η_2}
4: Sample $e_2 \in R_q$ from CBD_{η_2}
5: **return** $(A^T r + e_1, t^T r + e_2 + \lfloor q/2 \rceil m)$

Decrypt(sk, c)

Input: User's secret key $sk \in R_q^2$
Input: Ciphertext $c \in R_q^2 \times R_q$
Output: Message $m' \in R_q$
1: $w = c_2 - sk^T c_1$
2: **return** $m' = \lfloor 2w/q \rceil$

APPENDIX

The BKZ basis reduction algorithm and the security of CKKS19

- The most efficient way to solve lattice problems is to "reduce" a basis with BKZ.
- To do that, we reduce the basis by blocks of size β.
  To summarize, BKZ solves SVP in each sublattice generated by β vectors.
- Increasing β means:
  - More information at each iteration $\Rightarrow$ better output
  - Solving SVP in higher dimension $\Rightarrow$ higher running time.

- Breaking CKKS19 requires to solve a lattice problem with a certain quality.
- Reaching this quality requires at least a block size β.
- Using such a block size requires a running-time of at least 2^{λ}.
- So CKKS19 provides a λ-bit security.

APPENDIX

The BKZ basis reduction algorithm and the security of CKKS19

- The most efficient way to solve lattice problems is to "reduce" a basis with BKZ.
- To do that, we reduce the basis by blocks of size β.
  To summarize, BKZ solves SVP in each sublattice generated by β vectors.
- Increasing β means:
  - More information at each iteration $\Rightarrow$ better output
  - Solving SVP in higher dimension $\Rightarrow$ higher running time.

- Breaking CKKS19 requires to solve a lattice problem with a certain quality.
- Reaching this quality requires at least a block size β.
- Using such a block size requires a running-time of at least 2^{λ}.
- So CKKS19 provides a λ-bit security.

APPENDIX

The BKZ basis reduction algorithm and the security of CKKS19

- The most efficient way to solve lattice problems is to "reduce" a basis with BKZ.
- To do that, we reduce the basis by blocks of size β.
  To summarize, BKZ solves SVP in each sublattice generated by β vectors.
- Increasing β means:
  - More information at each iteration $\Rightarrow$ better output
  - Solving SVP in higher dimension $\Rightarrow$ higher running time.

- Breaking CKKS19 requires to solve a lattice problem with a certain quality.
- Reaching this quality requires at least a block size β.
- Using such a block size requires a running-time of at least 2^{λ}.
- So CKKS19 provides a λ-bit security.

APPENDIX

The BKZ basis reduction algorithm and the security of CKKS19

- The most efficient way to solve lattice problems is to "reduce" a basis with BKZ.
- To do that, we reduce the basis by blocks of size β.
  To summarize, BKZ solves SVP in each sublattice generated by β vectors.
- Increasing β means:
  - More information at each iteration $\Rightarrow$ better output
  - Solving SVP in higher dimension $\Rightarrow$ higher running time.

- Breaking CKKS19 requires to solve a lattice problem with a certain quality.
- Reaching this quality requires at least a block size β.
- Using such a block size requires a running-time of at least 2^{λ}.
- So CKKS19 provides a λ-bit security.

APPENDIX

The BKZ basis reduction algorithm and the security of CKKS19

- The most efficient way to solve lattice problems is to "reduce" a basis with BKZ.
- To do that, we reduce the basis by blocks of size β.
  To summarize, BKZ solves SVP in each sublattice generated by β vectors.
- Increasing β means:
  - More information at each iteration $\Rightarrow$ better output
  - Solving SVP in higher dimension $\Rightarrow$ higher running time.

- Breaking CKKS19 requires to solve a lattice problem with a certain quality.
- Reaching this quality requires at least a block size β.
- Using such a block size requires a running-time of at least 2^{λ}.
- So CKKS19 provides a λ-bit security.

APPENDIX

The BKZ basis reduction algorithm and the security of CKKS19

- The most efficient way to solve lattice problems is to "reduce" a basis with BKZ.
- To do that, we reduce the basis by blocks of size β.
  To summarize, BKZ solves SVP in each sublattice generated by β vectors.
- Increasing β means:
  - More information at each iteration $\Rightarrow$ better output
  - Solving SVP in higher dimension $\Rightarrow$ higher running time.

- Breaking CKKS19 requires to solve a lattice problem with a certain quality.
- Reaching this quality requires at least a block size β.
- Using such a block size requires a running-time of at least 2^{λ}.
- So CKKS19 provides a λ-bit security.

APPENDIX

The BKZ basis reduction algorithm and the security of CKKS19

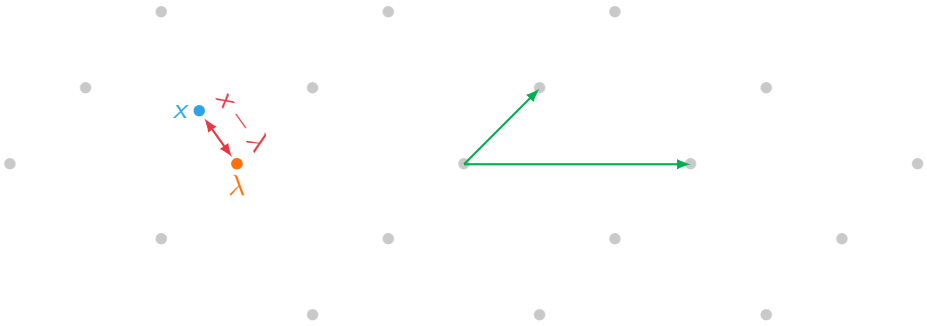
- The most efficient way to solve lattice problems is to "reduce" a basis with BKZ.
- To do that, we reduce the basis by blocks of size β.
  To summarize, BKZ solves SVP in each sublattice generated by β vectors.
- Increasing β means:
  - More information at each iteration $\Rightarrow$ better output
  - Solving SVP in higher dimension $\Rightarrow$ higher running time.

- Breaking CKKS19 requires to solve a lattice problem with a certain quality.
- Reaching this quality requires at least a block size β.
- Using such a block size requires a running-time of at least 2^{λ}.
- So CKKS19 provides a λ-bit security.

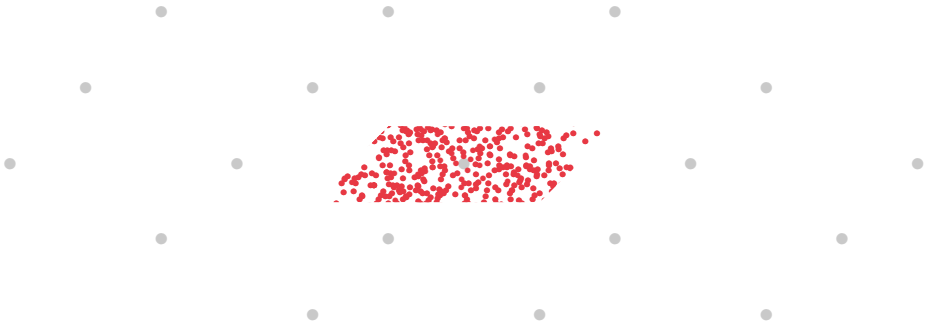
Structured variants of LWE

Plain	Module	Ring
$\begin{pmatrix} 8 & 6 & -1 & 3 \\ -1 & 5 & 7 & -4 \\ 4 & -1 & 9 & 1 \\ -1 & -2 & 6 & 0 \end{pmatrix}$	$\begin{pmatrix} 8 & 6 & -1 & 3 \\ -6 & 8 & -3 & -1 \\ 4 & -1 & 9 & 1 \\ 1 & 4 & -1 & 9 \end{pmatrix}$	$\begin{pmatrix} 8 & 6 & -1 & 3 \\ -3 & 8 & 6 & -1 \\ 1 & -3 & 8 & 6 \\ -6 & 1 & -3 & 8 \end{pmatrix}$
—	$\begin{pmatrix} \times_{6X+8} & \times_{3X-1} \\ \times_{-X+4} & \times_{X+9} \end{pmatrix}$	$\times_{3X^3-X^2+6X+8}$

Deterministic - Output the closest lattice element



Deterministic - Output the closest lattice element



Randomized - Output a relatively close lattice element