IEICE Electronics Express, Vol.17, No.17, 1–6

LETTER

A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse

Yiming Huang[1], Miaoqing Huang[2], Zhongkui Lei[1, a)], and Jiaxuan Wu[3]

Pakin Panawattanakul
4 / Aug /2025

Presentation outline

- Objective & Scope
- Accelerator architecture
- Module & Dataflow for encapsulation and decapsulation
- FPGAs used in implementation
- Future work

Objective & Scope

- Optimize resource utilization for Kyber Accelerator
  - Reuse module for encapsulation and decapsulation
  - Use BRAM to store intermediate data
    - Between module
    - Pipeline
- Not include key-generation
- Arithmetic Pipeline in NTT and Hash Module

Math Notations

- Bit stream = $\{0,1\}^{256}$ e.g. 011110101000..00
- Polynomial ring R_q : poly nomial degree 255, with coefficient in [0,3329)
- Polynomial ring P : polynomial degree 255 (intermediate state in calculation before reducing)
- Small polynomial S_η : polynomial degree 255, with coefficient [-η,η]
- Number in NTT form $\hat{x}$

Note : reference for more detail explaination and example page 32-35

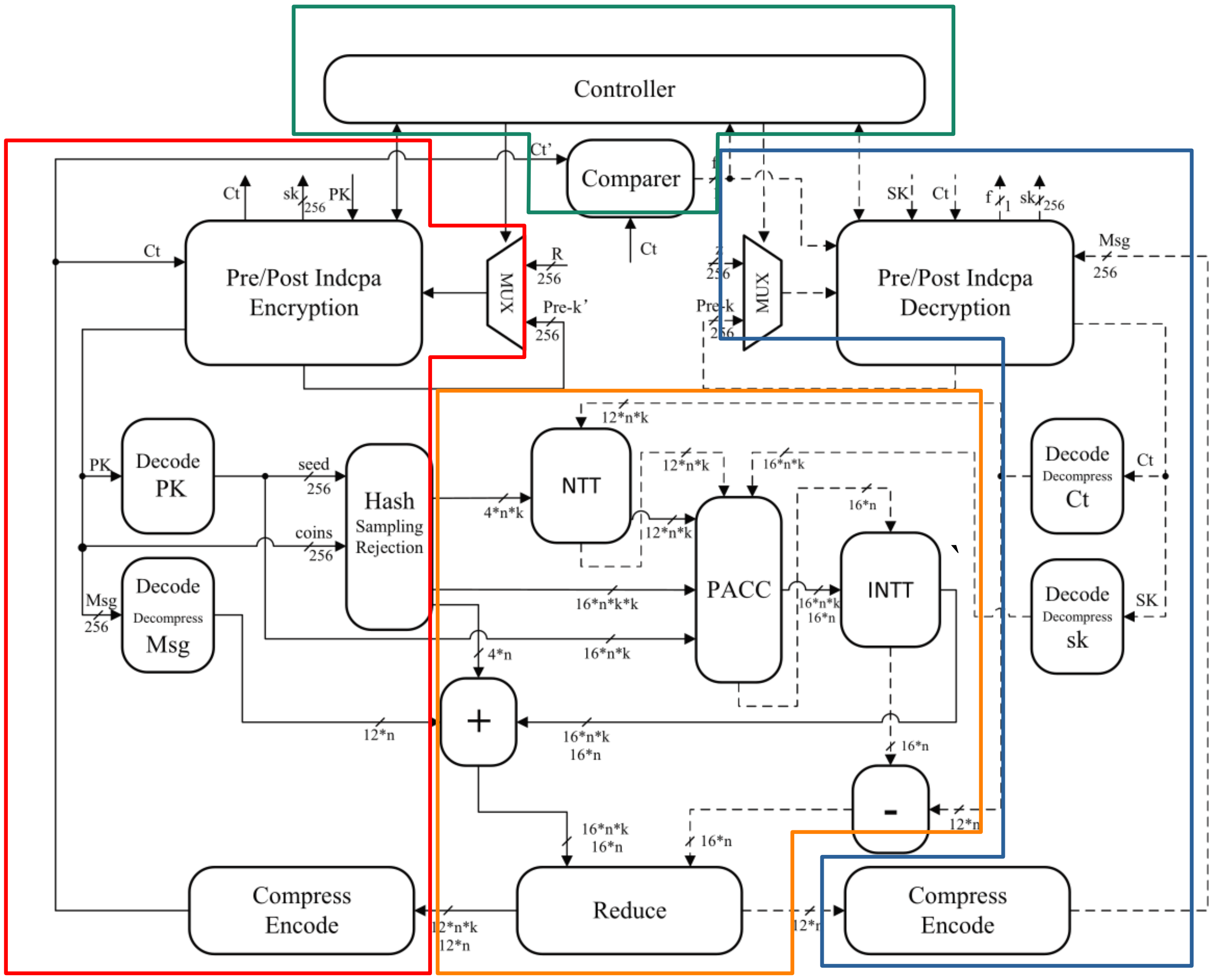
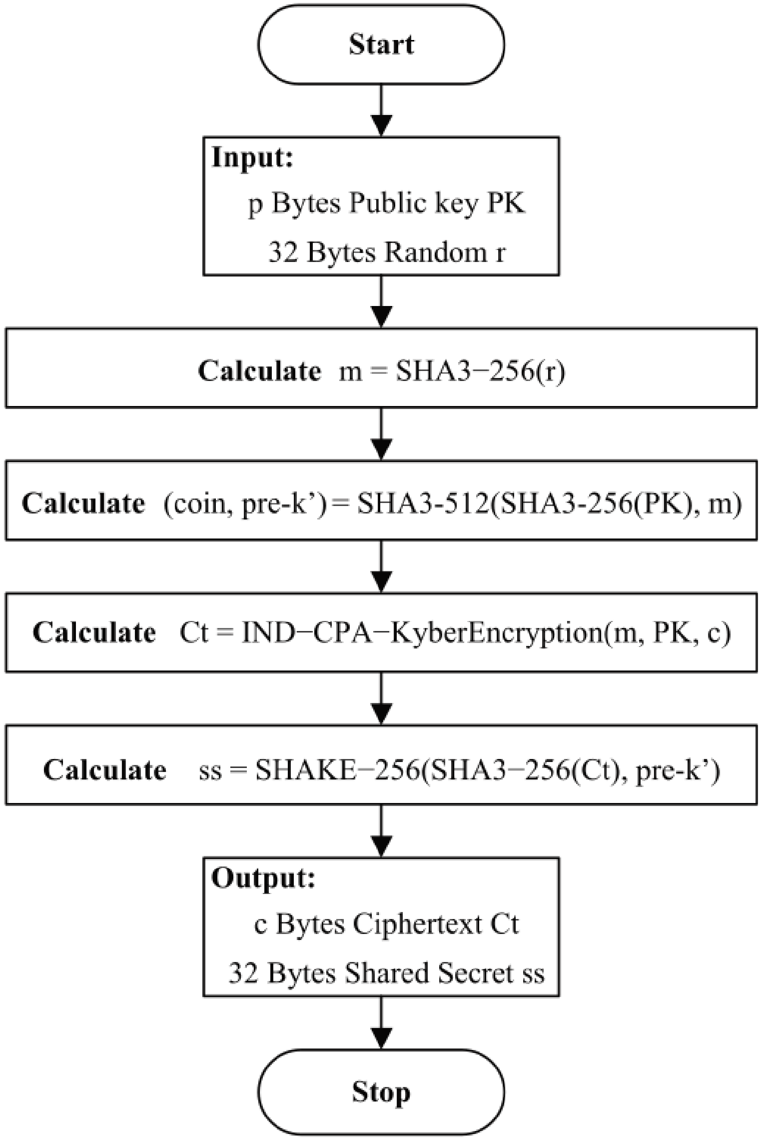


Fig. 3 Overall Kyber encapsulation and decapsulation architecture.

Modules sections

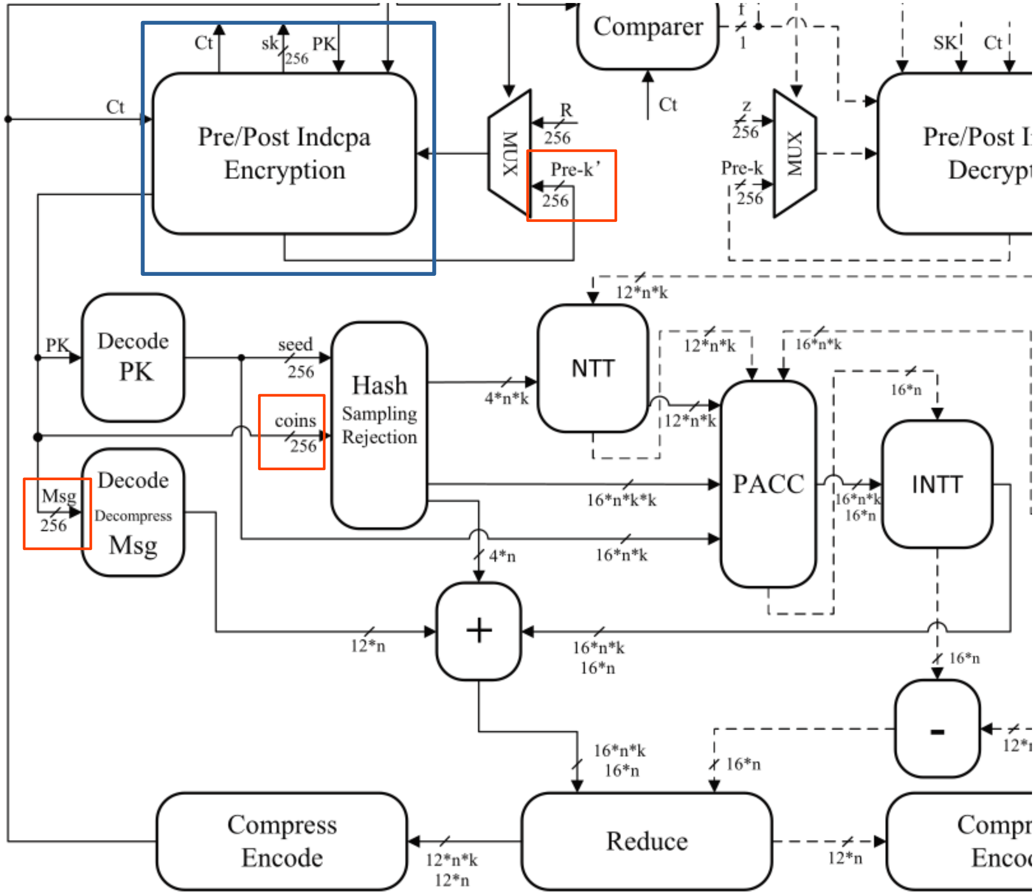
1) Control
2) Encapsulation
3) Decapsulation
4) Main computation

<u>Note</u> Article does not include detail implementation of each module



Encapsulation Module &Dataflow

Note : reference for more detailed explaination in Page 37-42



Pre/Post Indcpa Encryption

Pre-Encryption

Input

1) Public key: $(\rho, \hat{t})$
2) R = Random 256 bits

$$\rho = 0,1^{256}, \hat{t} = \begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix}$$

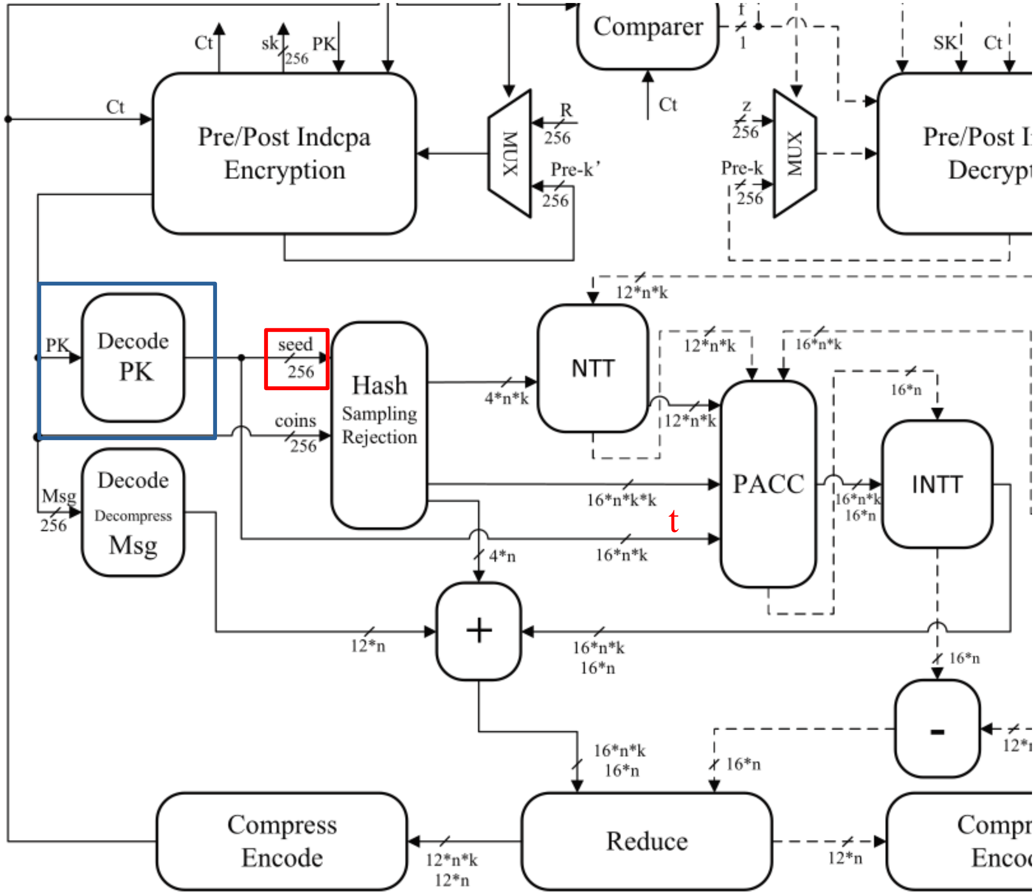
$$R = \{0,1\}^{256}$$

Output

m = SHA3-256(R) , Plain text message

$$(coins, pre-k') = SHA3-512(SHA3-256(PK), m)$$

$$m, coins, pre-k' = 0,1^{256}$$



Decode PK

Split public key into two part

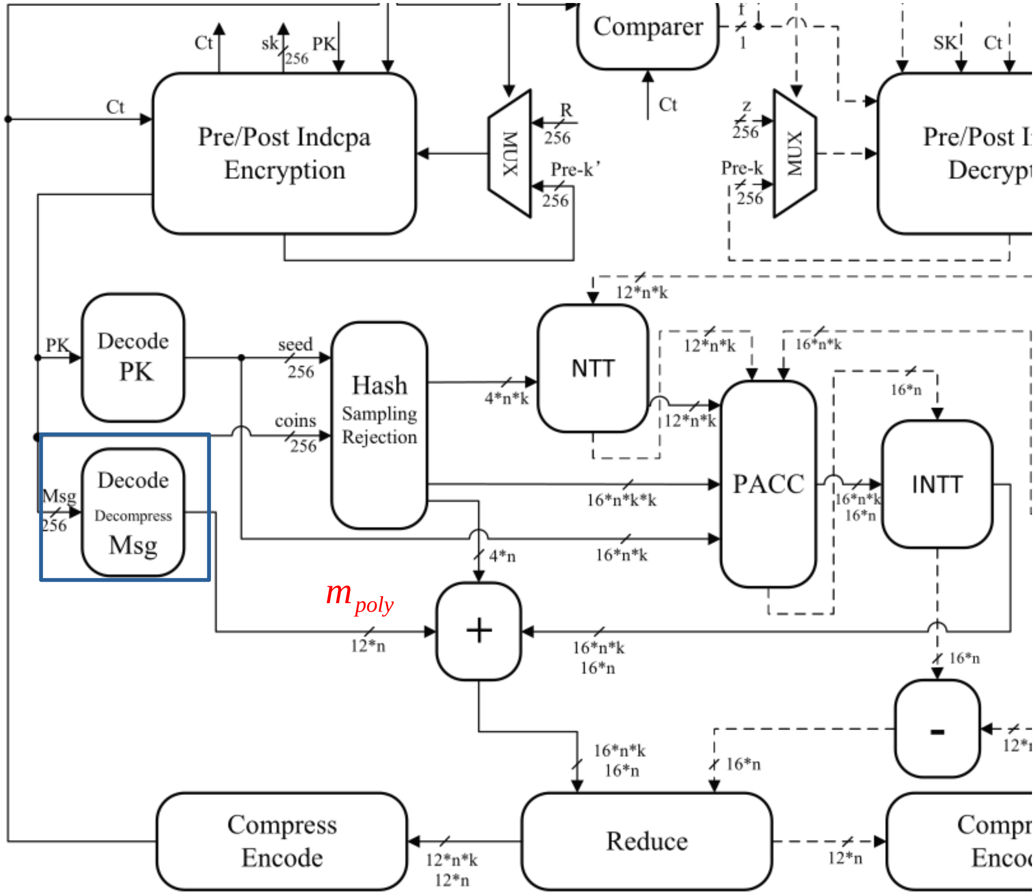
Input : public keys $PK = (\rho, \hat{t})$

$$\rho = \{0,1\}^{256}, \hat{t} = \begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix}$$

Output

1) Seed = ρ
2) $\hat{t}$ Increase coef size 12→16bits

$$\rho = \{0,1\}^{256}, \hat{t} = \begin{pmatrix} P \\ P \\ P \end{pmatrix}$$



Decode Msg

Input : plain text message

$$m = \{0,1\}^{256}$$

Calculate

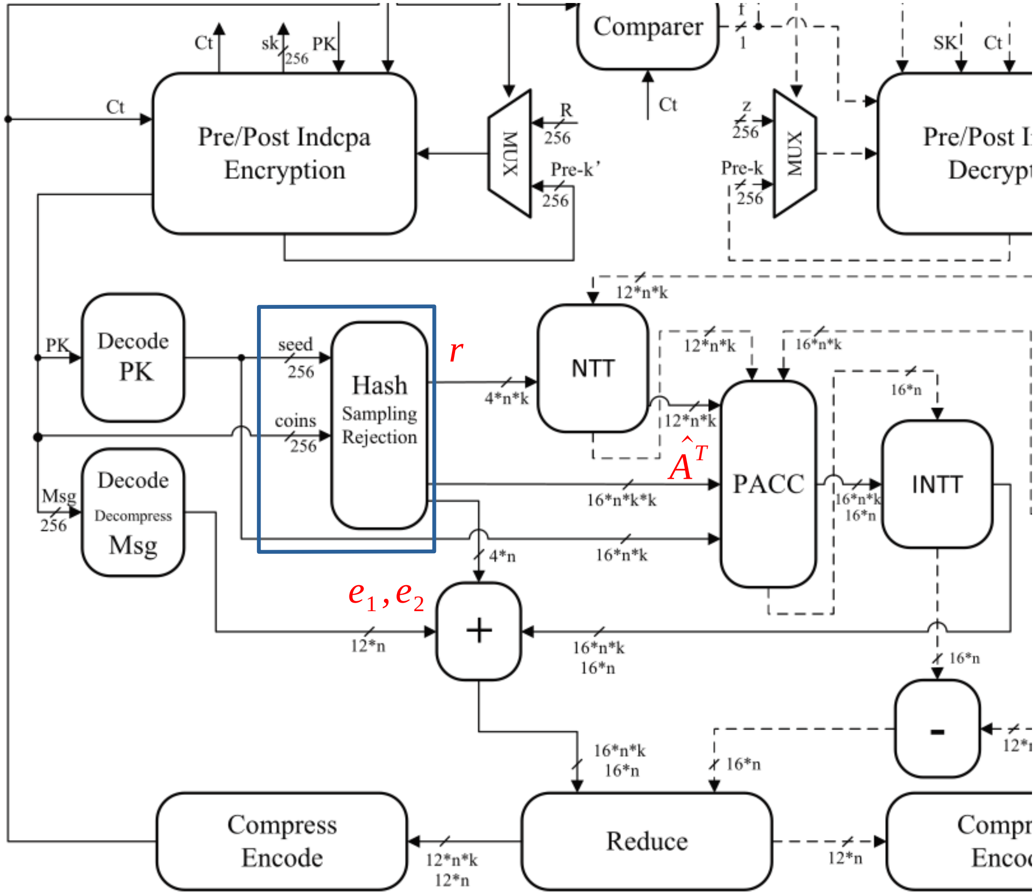
q=3329

$$m_{poly} = rounded(q/2)m = 1665m$$

For every bit covert: 0 → 0, 1 → 1665

Output : m_{poly} (polynomial form R_q)

$$m_{poly} = R_q$$



Hash Sampling Rejections

Use SHAKE-128 or SHAKE-256

Input

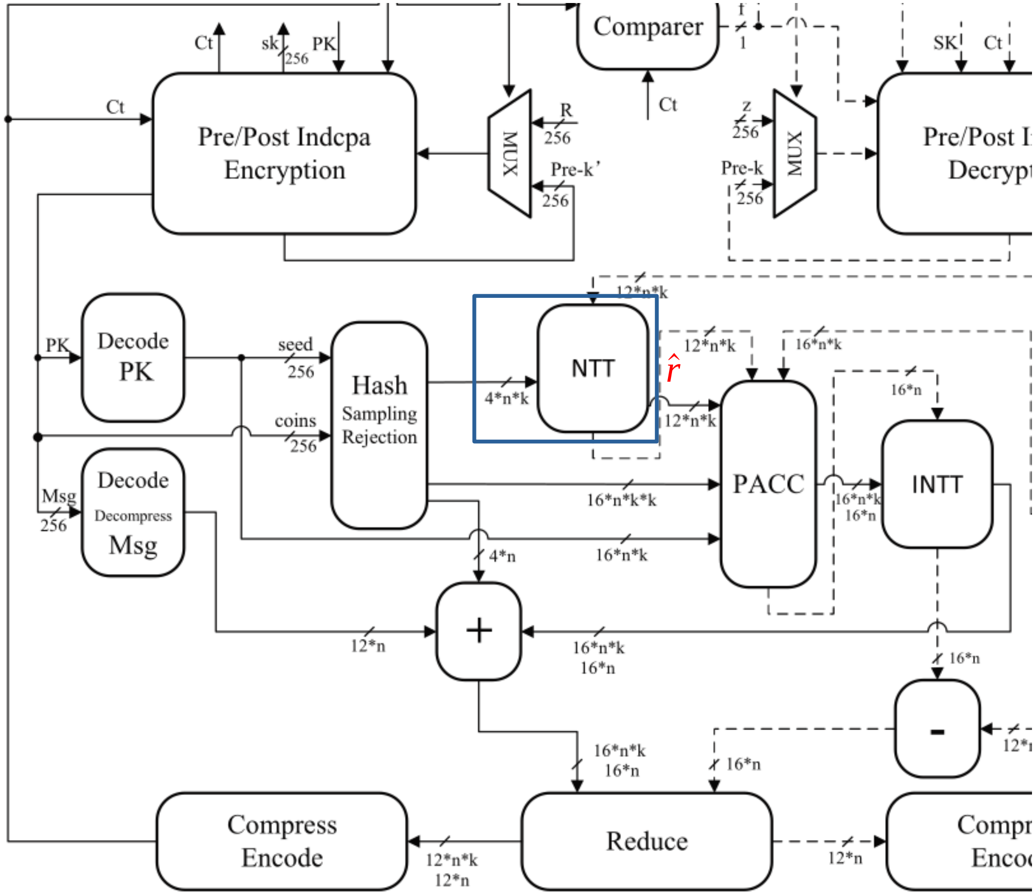
- Seed = $\{0,1\}^{256}$
- Coins = $\{0,1\}^{256}$

Output : small polynomial errors & Part of public keys

- e_1
- e_2
- r
- $\hat{A}^T$

$$r = \begin{pmatrix} S_\eta \\ S_\eta \\ S_\eta \end{pmatrix},\ e_1 = \begin{pmatrix} S_\eta \\ S_\eta \\ S_\eta \end{pmatrix},\ e_2 = S_\eta$$

$$\hat{A}^T = \begin{pmatrix} P & P & P \\ P & P & P \\ P & P & P \end{pmatrix}$$



NTT Module

Convert polynomial to NTT form

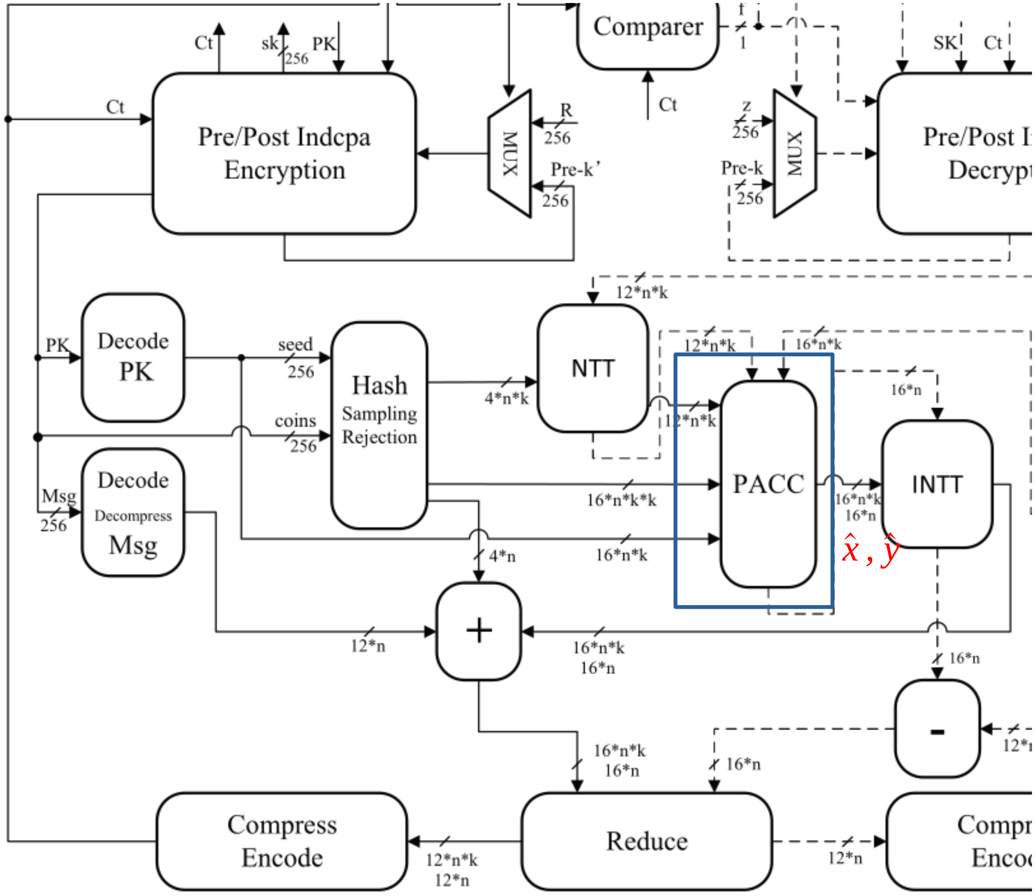
Input : small polynomial r

$$r = \begin{pmatrix} S_\eta \\ S_\eta \\ S_\eta \end{pmatrix}$$

Output : polynomial in NTT form,
Polynomial Ring R_q

$$\hat{r} = NTT(r)$$

$$\hat{r} = \begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix}$$



PACC : Polynomial Accumulator

Compute polynomial aritmetic in NTT form with mondgomery multiplications

Input : Error Polynomial, variables from PK

1) $\hat{r}$
2) $\hat{A}^T$
3) $\hat{t}$

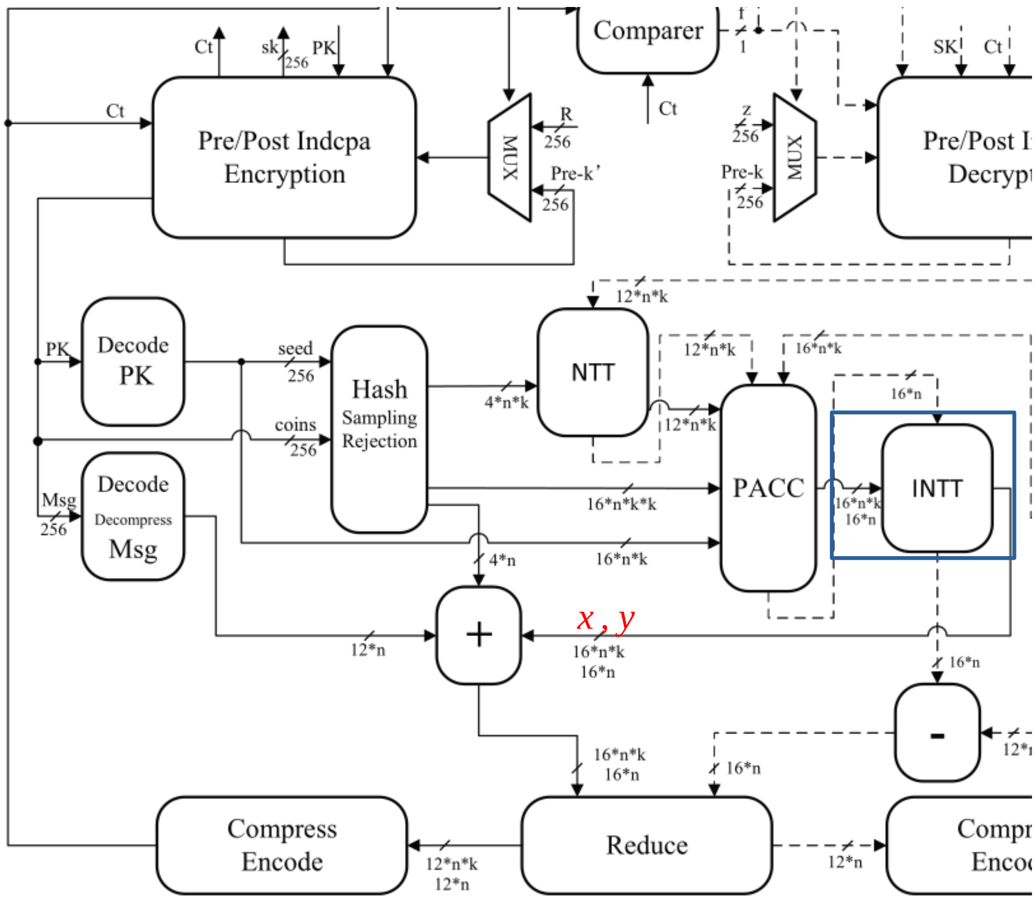
$$\hat{r} = \begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix} \quad \hat{A}^T = \begin{pmatrix} P & P & P \\ P & P & P \\ P & P & P \end{pmatrix} \quad \hat{t} = \begin{pmatrix} P \\ P \\ P \end{pmatrix}$$

Output

1) $\hat{x} = \hat{A}^T \hat{r}$
2) $\hat{y} = \hat{t}^T \hat{r}$

$$\hat{x} = \begin{pmatrix} P \\ P \\ P \end{pmatrix} \quad \hat{y} = P$$

Note $\hat{x}, \hat{y}$ is only intermediate value before computation of $\hat{u}, \hat{v}$



INTT Module

Convert Polynomial in NTT back to normal form

Input : $\hat{x}, \hat{y}$

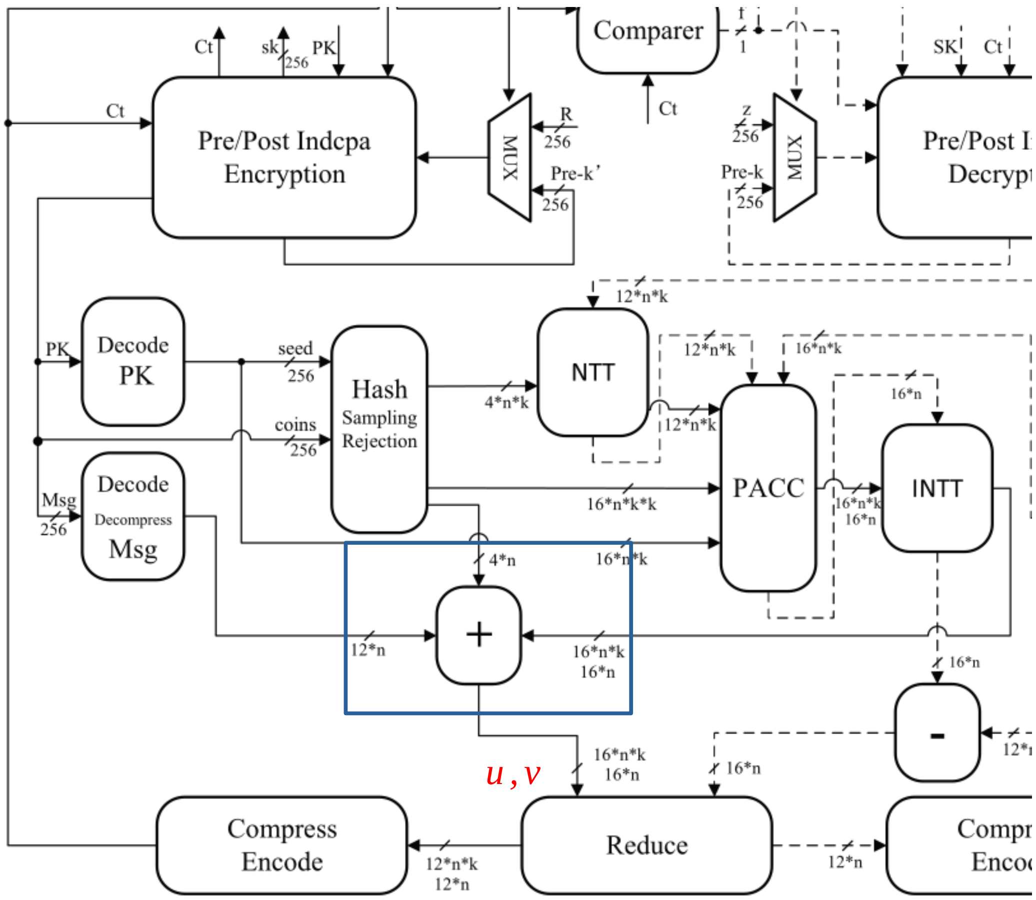
$$\hat{x} = \begin{pmatrix} P \\ P \\ P \end{pmatrix}, \quad \hat{y} = P$$

Output

1) $x = NTT^{-1}(\hat{x})$

2) $y = NTT^{-1}(\hat{y})$

$$x = \begin{pmatrix} P \\ P \\ P \end{pmatrix}, \quad y = P$$



Addition Module (+)

Compute polynomial addition

Input

1) x, y

2) m_{poly}

3) e_1, e_2

$$x = \begin{pmatrix} P \\ P \\ P \end{pmatrix}, \; y = P$$

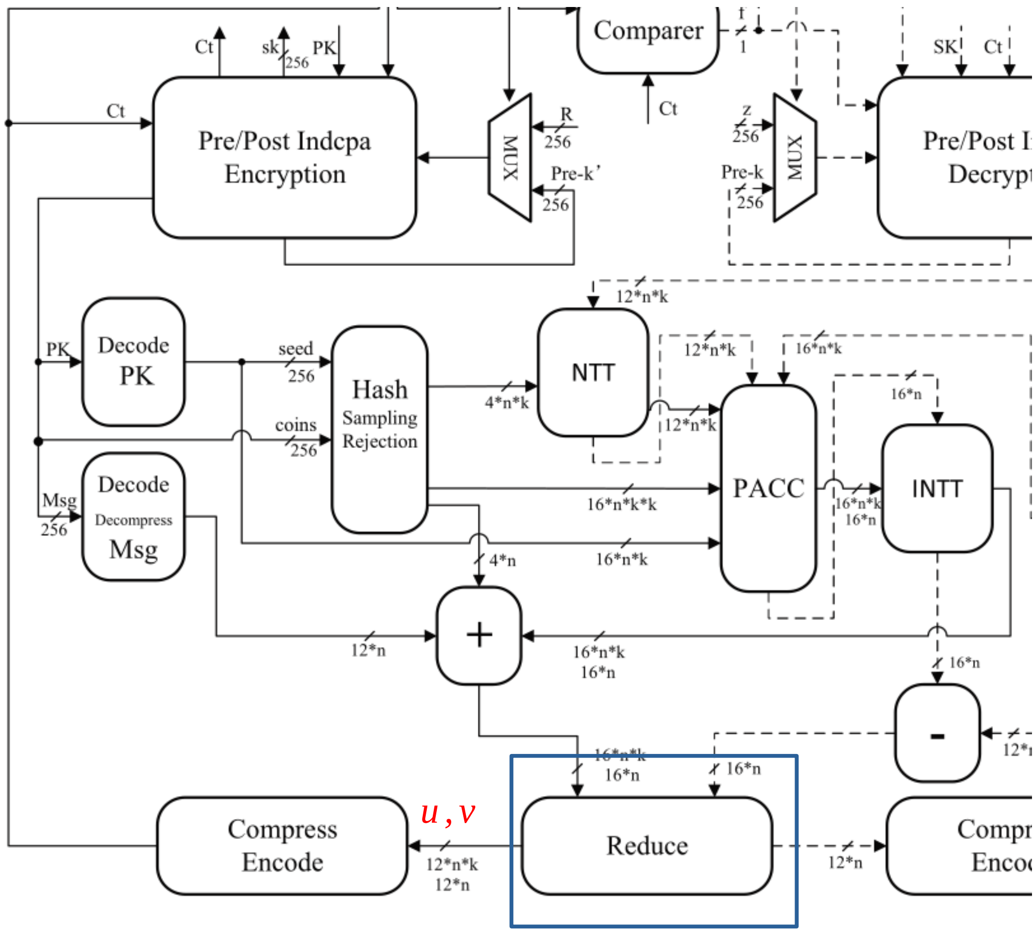
$$m_{poly} = R_q \quad e_1 = \begin{pmatrix} S_\eta \\ S_\eta \\ S_\eta \end{pmatrix}, \; e_2 = S_\eta$$

Output

1) $u = x + e_1$

2) $v = y + e_2 + m_{poly}$

$$u = \begin{pmatrix} P \\ P \\ P \end{pmatrix}, \; v = P$$



Reduce Module

Modular reduction reduce the coefficient to be with in [0,q)

Input : polynomials

1) v

2) u

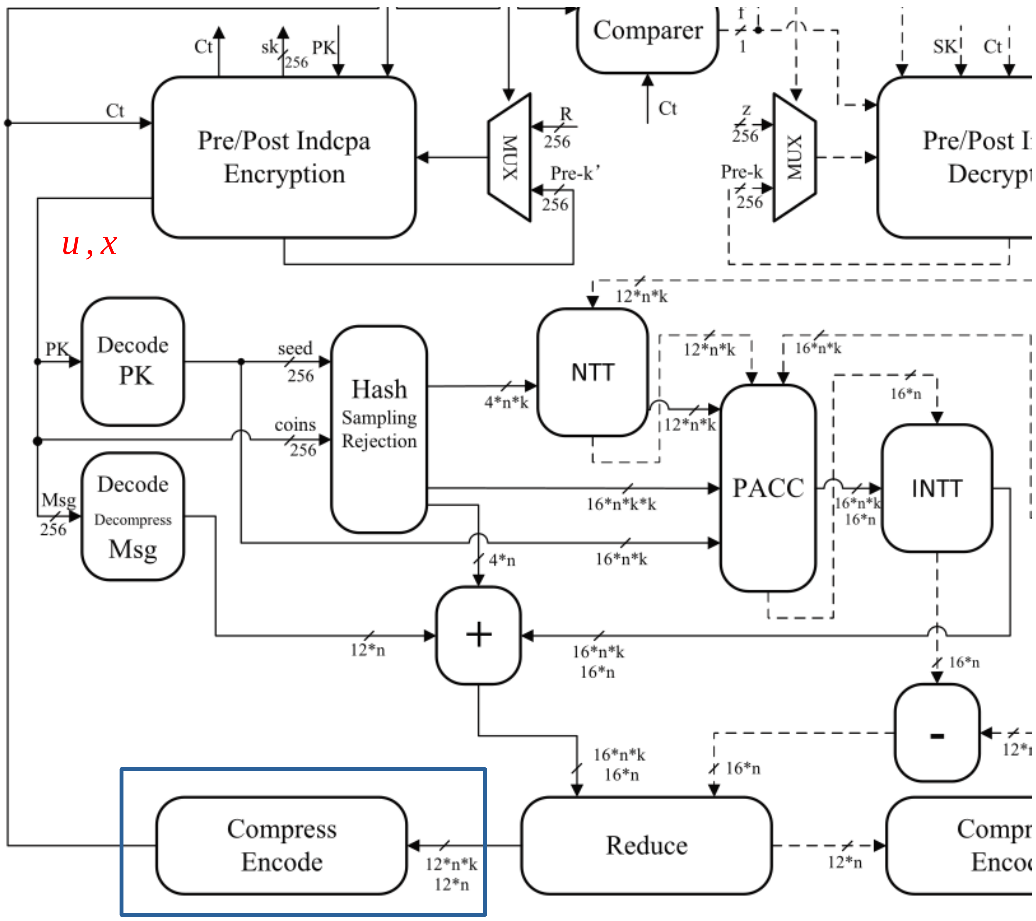
$$u=\begin{pmatrix} P \\ P \\ P \end{pmatrix}, \ v=P$$

Output : polynomials ring R_q

1) u = u mod q

2) v = v mod q

$$u=\begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix}, \ v=R_q$$



Compress encode

Compress ciphertext using rounded funciton

Input : u,v

$$u=\begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix},\ v=R_q$$

Compute

$$c_1=compress(u,d_u)=round_q((2^{d_u}/q)x)\,mod\,2^{d_u}$$

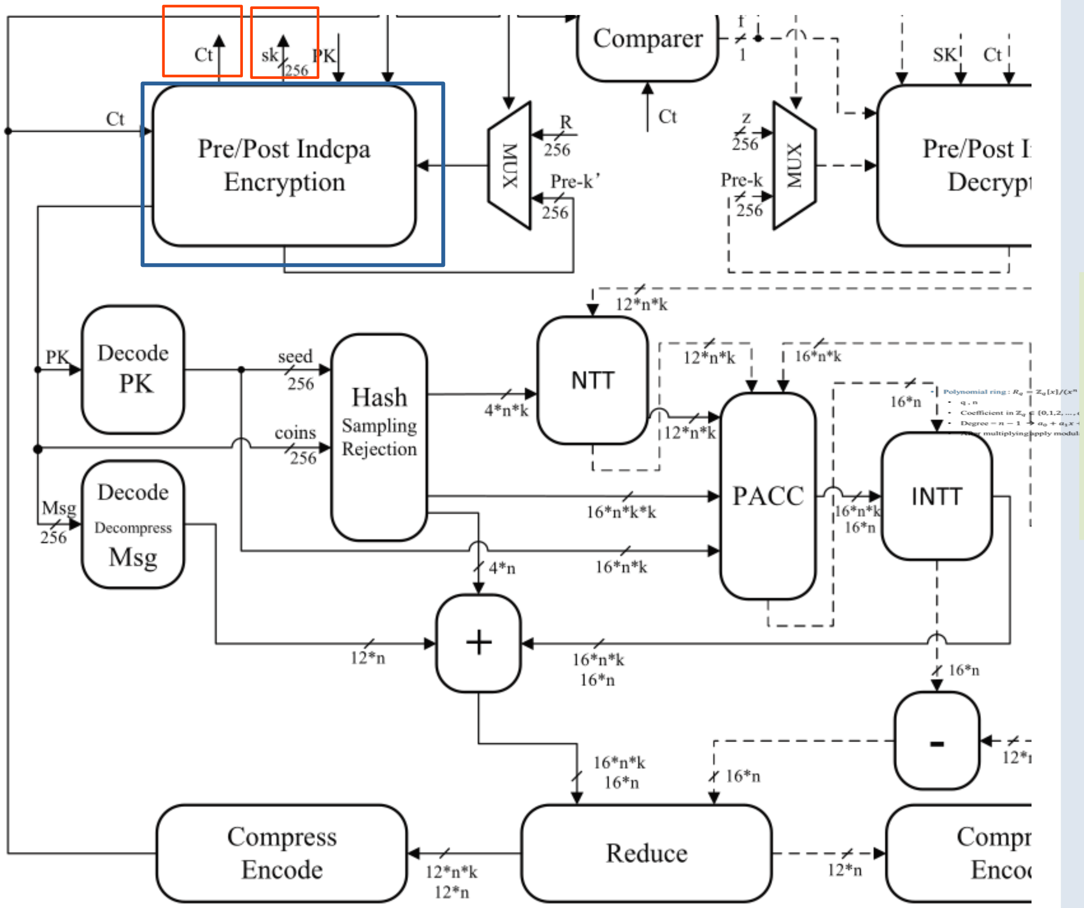
$$c_2=compress(v,d_v)=round_q((2^{d_v}/q)x)\,mod\,2^{d_v}$$

Output : Cipher text with lower coefficient bits

$$Ct=(c_1,c_2)$$

$$c_1=\begin{pmatrix} P \\ P \\ P \end{pmatrix}\ \text{with coef}=d_u\ \text{bits}$$

$$c_2=\ P\ \ \text{with coef}=d_v\ \text{bits}$$



Post Indcpa Encryption

Input

1) Cipher text: $Ct = (c_1, c_2)$

2) Part of PK: $Pre - k'$

$$c_1 = \begin{pmatrix} P \\ P \\ P \end{pmatrix} \text{ with coef} = d_u \text{ bits} \qquad Pre - k' = \{0,1\}^{256}$$

$$c_2 = \ P \ \text{ with coef} = d_v \text{ bits}$$

Output

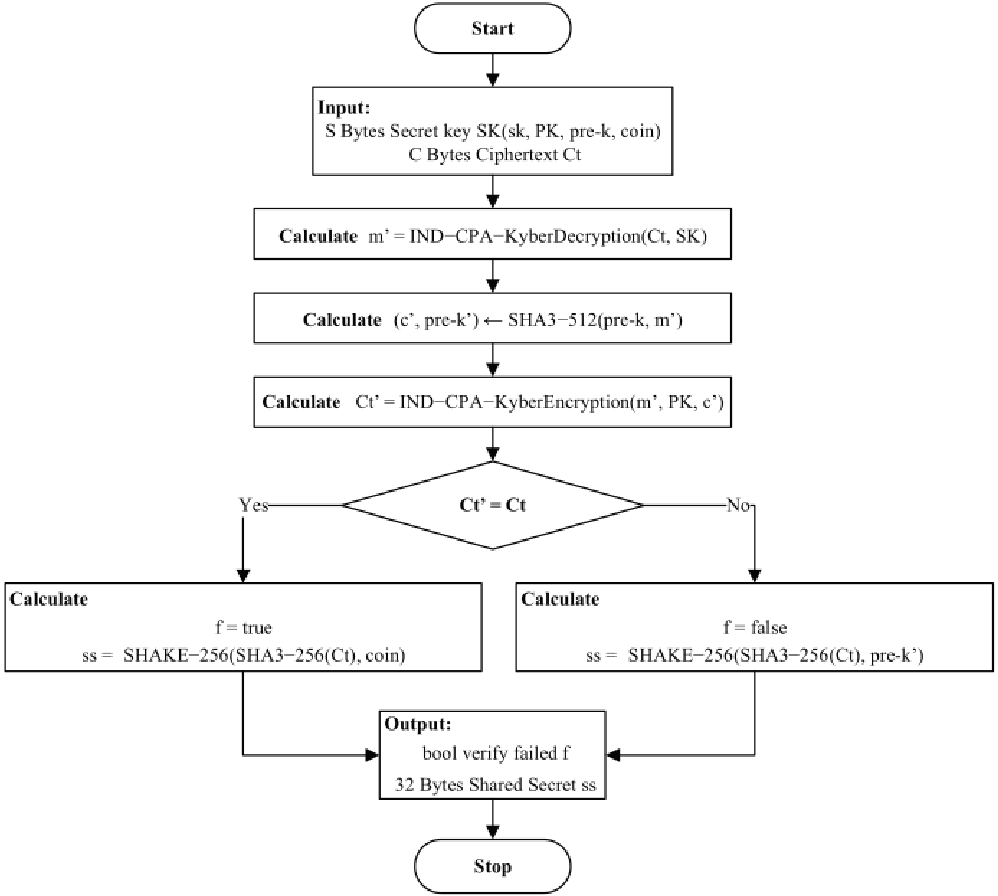
1) Cipher text : $Ct = (c_1, c_2)$

2) ss = SHAKE-256(SHA3-256(Ct), $Pre - k$')

$$c_1 = \begin{pmatrix} P \\ P \\ P \end{pmatrix} \text{ with coef} = d_u \text{ bits} \qquad ss = \{0,1\}^{256}$$

$$c_2 = \ P \ \text{ with coef} = d_v \text{ bits}$$

Decapsulation Module & Dataflow



Note : reference for more detailed explaination in Page 37-42

Pre Indcpa Decryption

Recive and pass input to other module as it is

Input

1) Cipher text : $Ct=(c_1,c_2)$

2) Private key : $SK=(\hat{s},PK,pre-k,coin)$

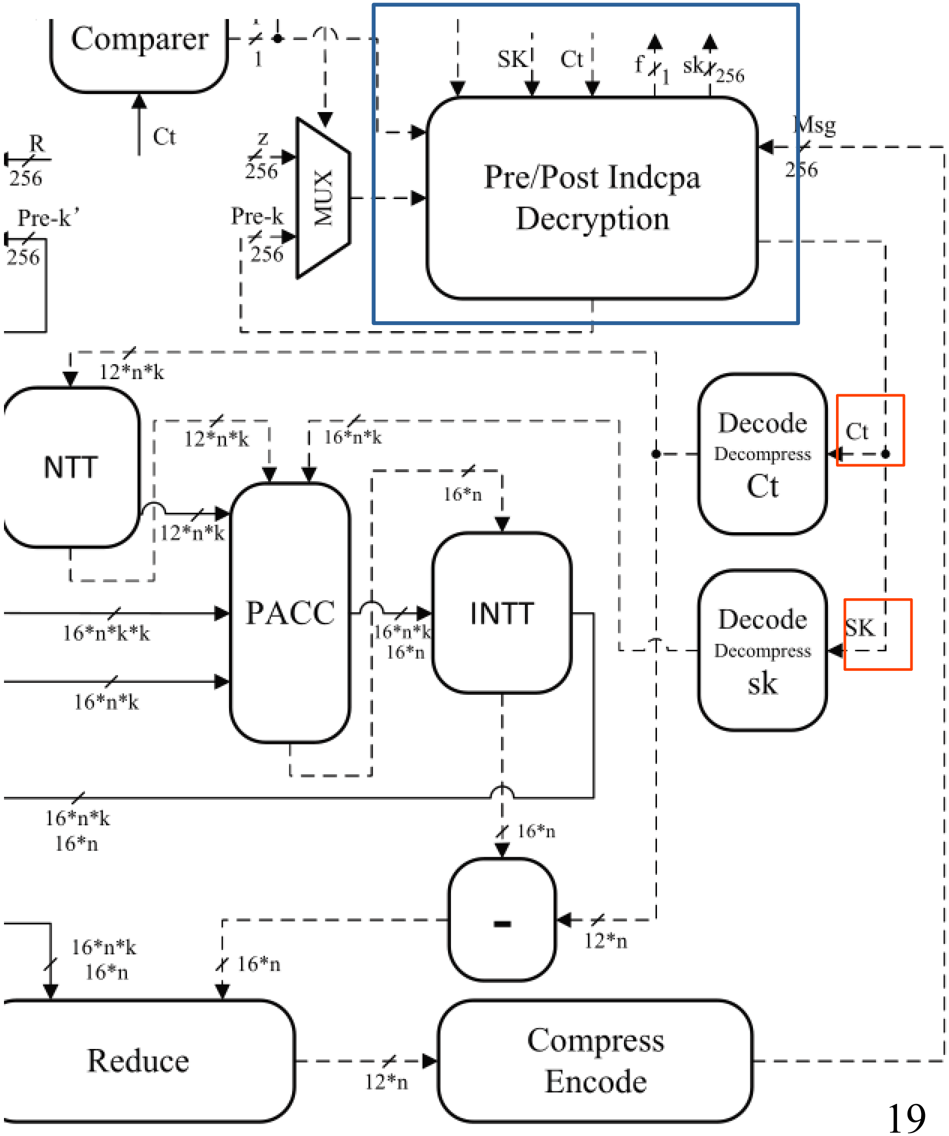
Output:

1) Cipher text : $Ct=(c_1,c_2)$

2) Private key : $SK=(\hat{s},PK,pre-k,coin)$

$$c_1=\begin{pmatrix}P\\P\\P\end{pmatrix} \text{ with coef}=d_u \text{ bits} \quad \hat{s}=\begin{pmatrix}S_\eta\\S_\eta\\S_\eta\end{pmatrix} \quad \hat{t}=\begin{pmatrix}R_q\\R_q\\R_q\end{pmatrix}$$

$$c_2= P \text{ with coef}=d_v \text{ bits} \quad pre-k,coin,\rho=\{0,1\}^{256}$$



Decode Ct

Recive and pass input to other module

Input : Cipher text $Ct=(c_1,c_2)$

$$c_1=\begin{pmatrix}P\\P\\P\end{pmatrix}\ \text{with coef}=d_u\ \text{bits}$$

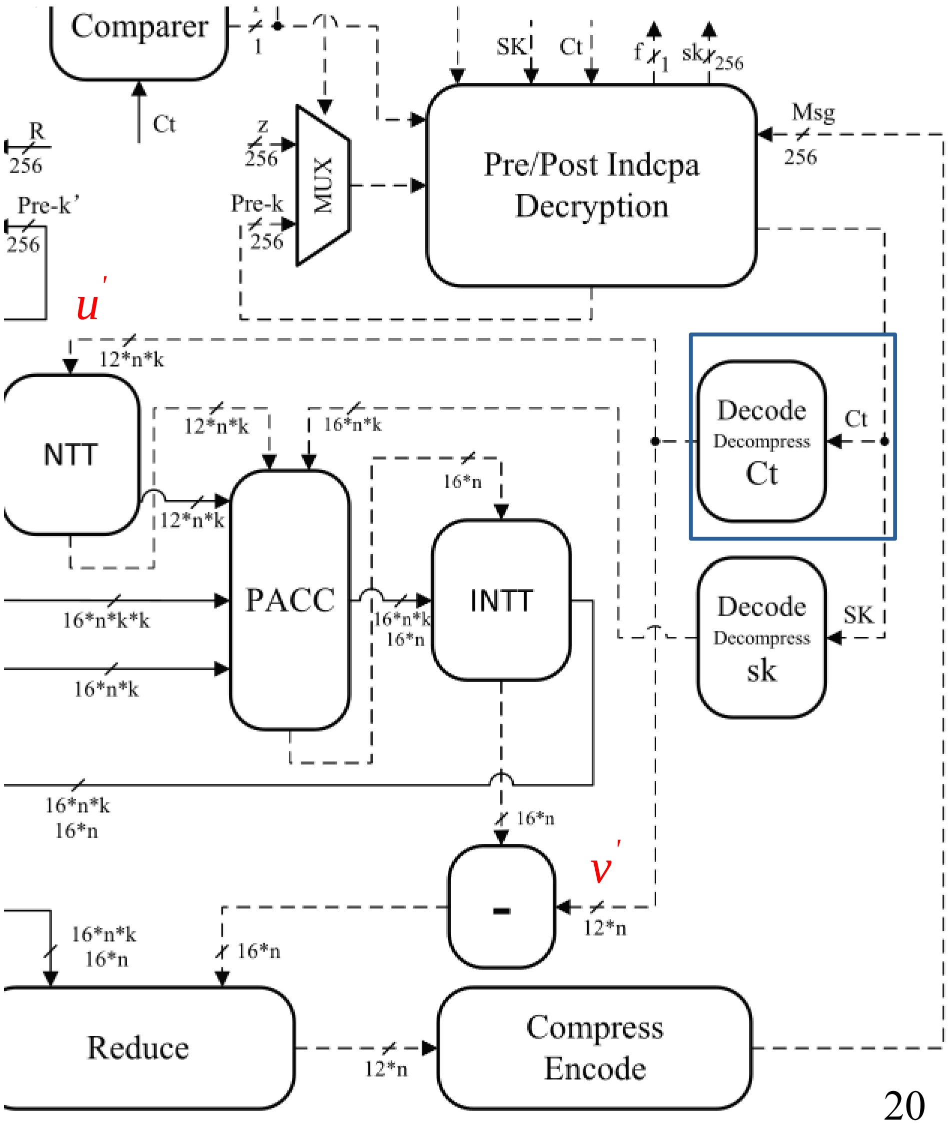
$$c_2=\ P\ \ \text{with coef}=d_v\ \text{bits}$$

Output: Cipher text in polynomial ring R_q

1) $u'=decompress(c_1,d_u)=round((q/2^{d_u})c_1)\,mod\,q$

2) $v'=decompress(c_2,d_v)=round((q/2^{d_v})c_2)\,mod\,q$

$$u'=\begin{pmatrix}R_q\\R_q\\R_q\end{pmatrix}\quad v'=\ R_q$$



Decode SK

Decode Encapsulation key then Transpose encryption key

Input : Private key

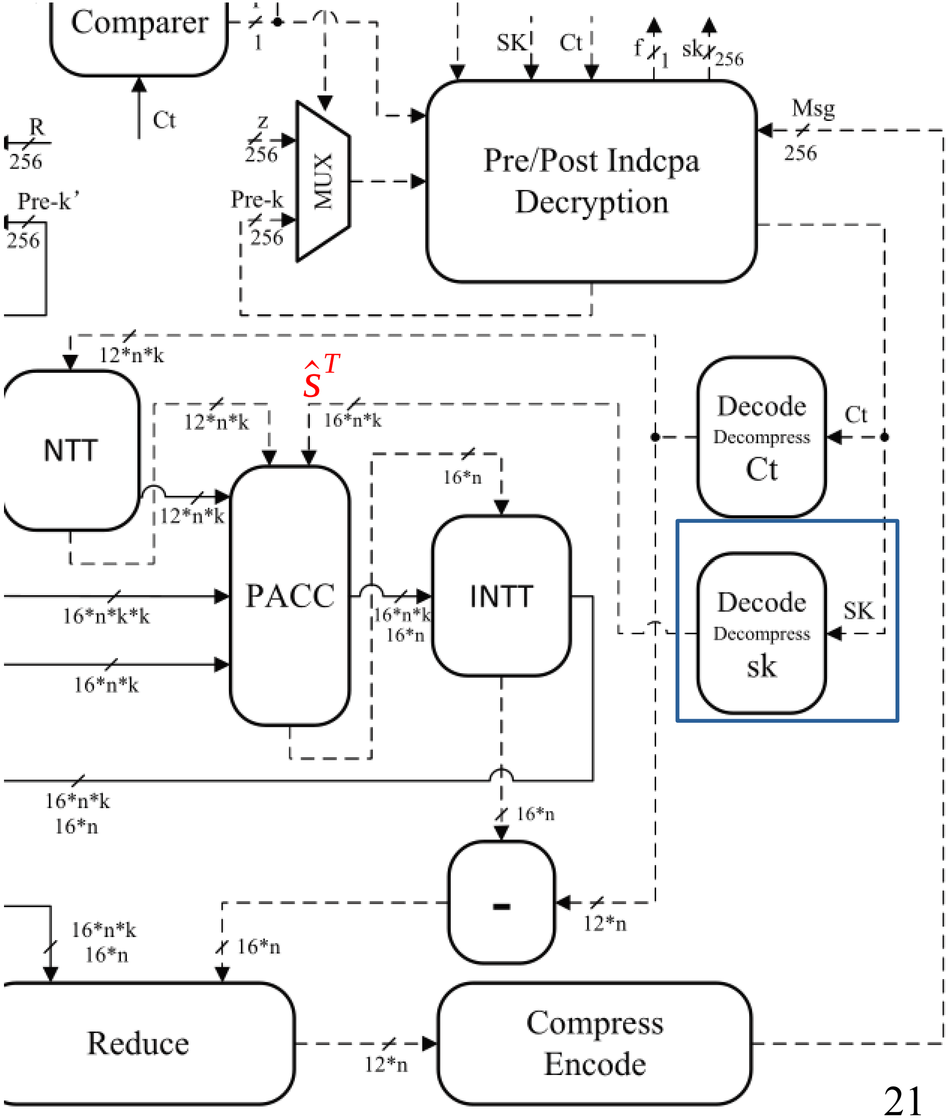
$$SK=(\hat{s}, PK, pre-k, coin)$$

$$\hat{s}=\begin{pmatrix} S_\eta \\ S_\eta \\ S_\eta \end{pmatrix} \quad \hat{t}=\begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix}$$

$$pre-k, coin, \rho=\{0,1\}^{256}$$

Output : decryption key $\hat{s}^T$ in polynomial form

$$\hat{s}^T=(P \;\; P \;\; P)$$



NTT module

Convert polynomial to NTT

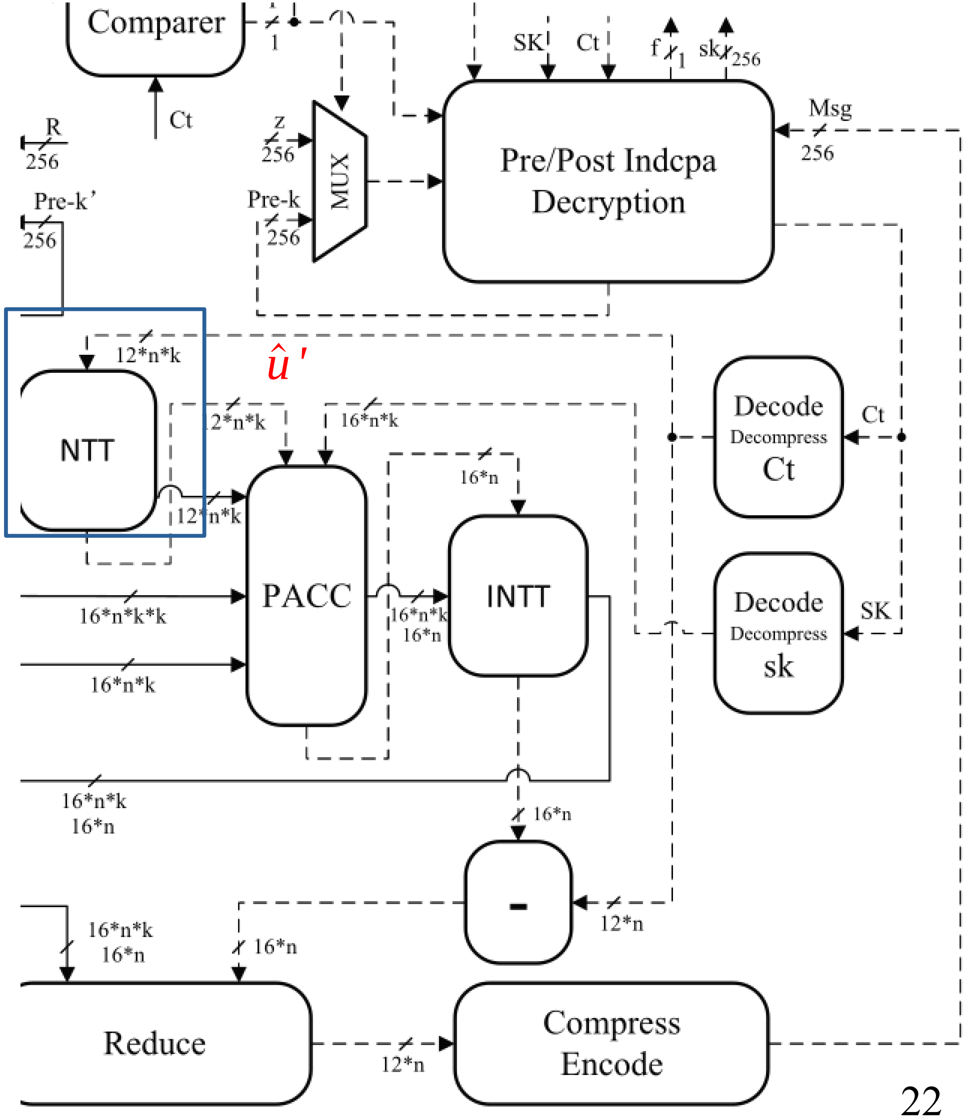
Input : Part of cipher text u'

$$u' = \begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix}$$

Output : cipher text in NTT form

$\hat{u}' = NTT(u')$

$$\hat{u}' = \begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix}$$



PACC : Polynomial Accumulator

Compute polynomial aritmetic in NTT form with mondgomery multiplications

Input

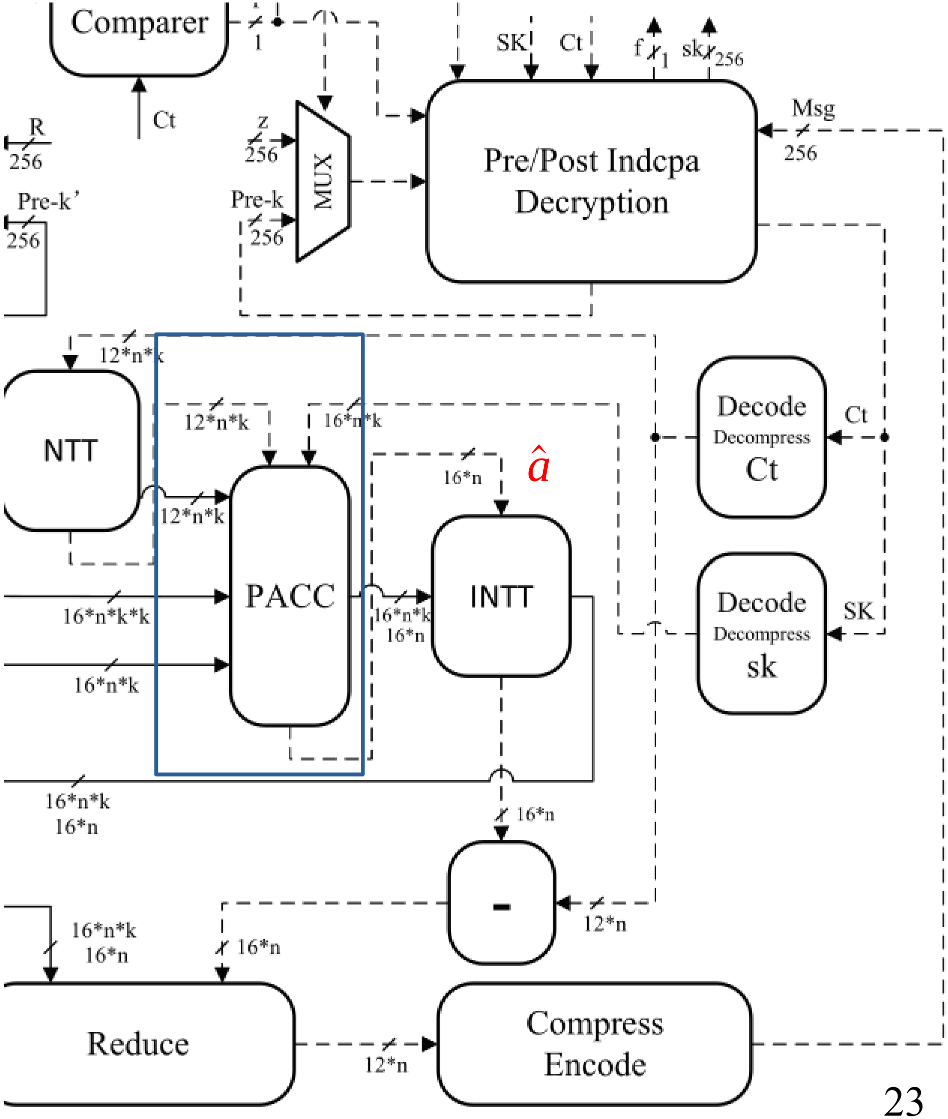
1) Part of cipher text : $\hat{u}'$
2) Encryptions key (transposed) : $\hat{s}^T$

$$\hat{u}' = \begin{pmatrix} R_q \\ R_q \\ R_q \end{pmatrix} \quad \hat{s}^T = (S_\eta \;\; S_\eta \;\; S_\eta)$$

Output : intermidiate data

$$\hat{a} = \hat{s}^T \times \hat{u}'$$

$$\hat{a} = P$$



INTT module

Inverse polynomial in NTT to normal form

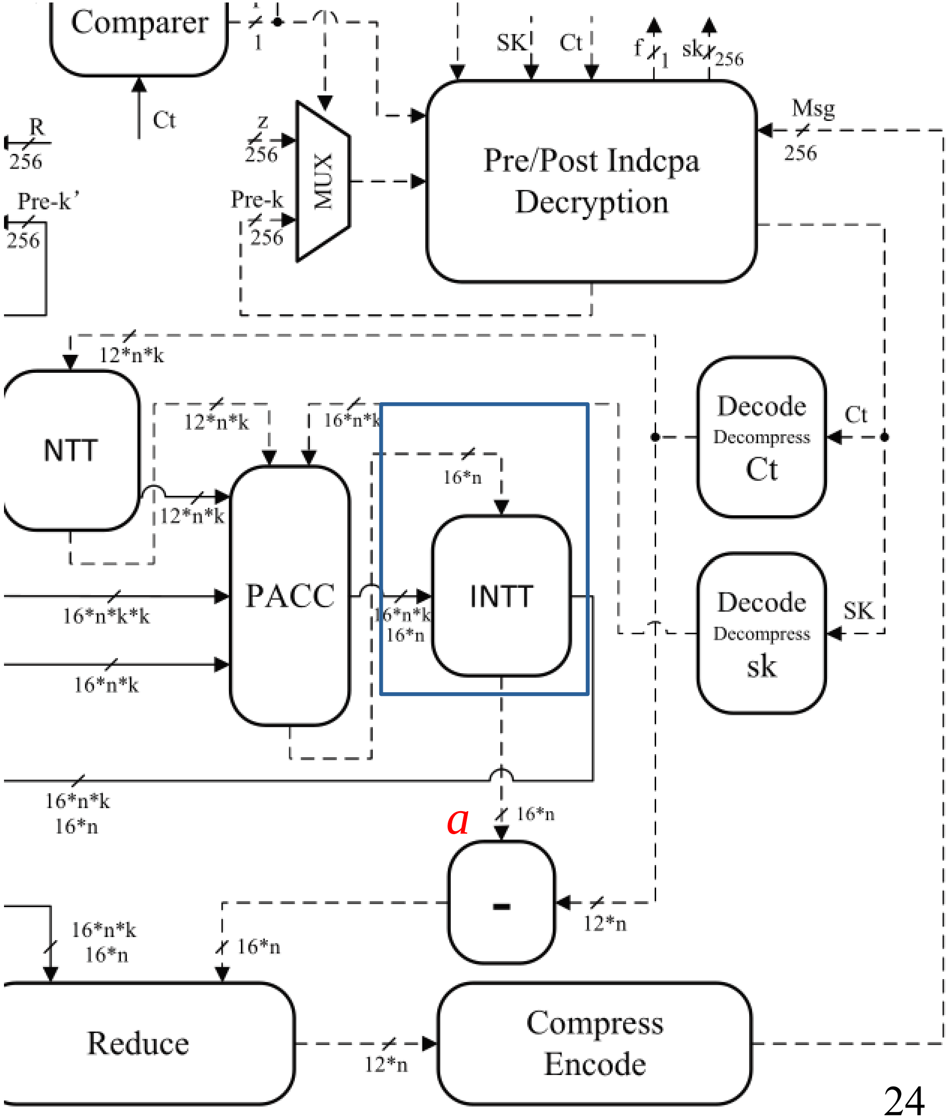
Input : intermediate data (NTT form) : $\hat{a}$

Output : intermidiate data (normal form)

$$a = NTT^{-1}(\hat{a})$$

$\hat{a} = P$

$a = P$



Subtraction Module (-)

Polynomial subtraction

Input

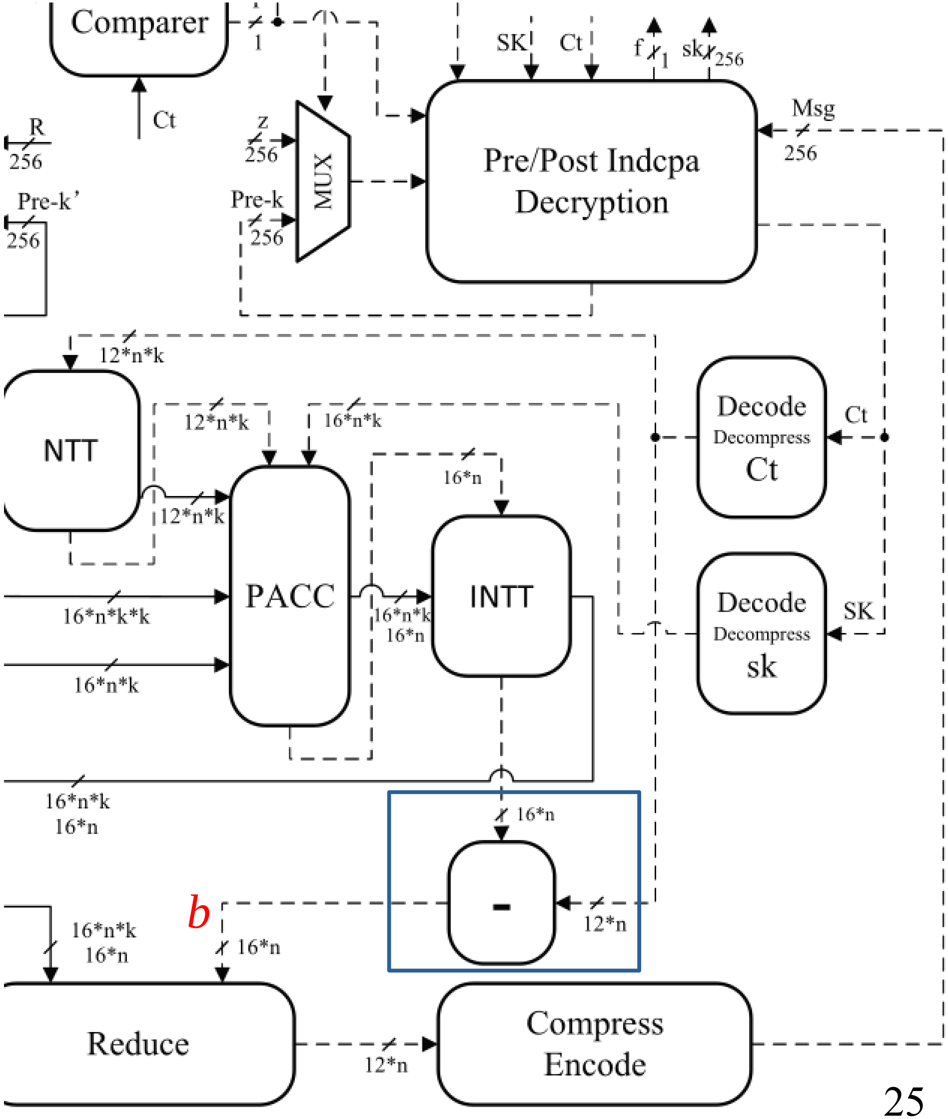
1) Intermediate data : a
2) Part of cipher text : v'

$a = P$
$v' = R_q$

Output : Polynomial Ring

$$b = v' - a = v' - (s^T u')$$

$b = P$



Reduce Module

Modular reduction reduce the coefficient to be with in [0,q)

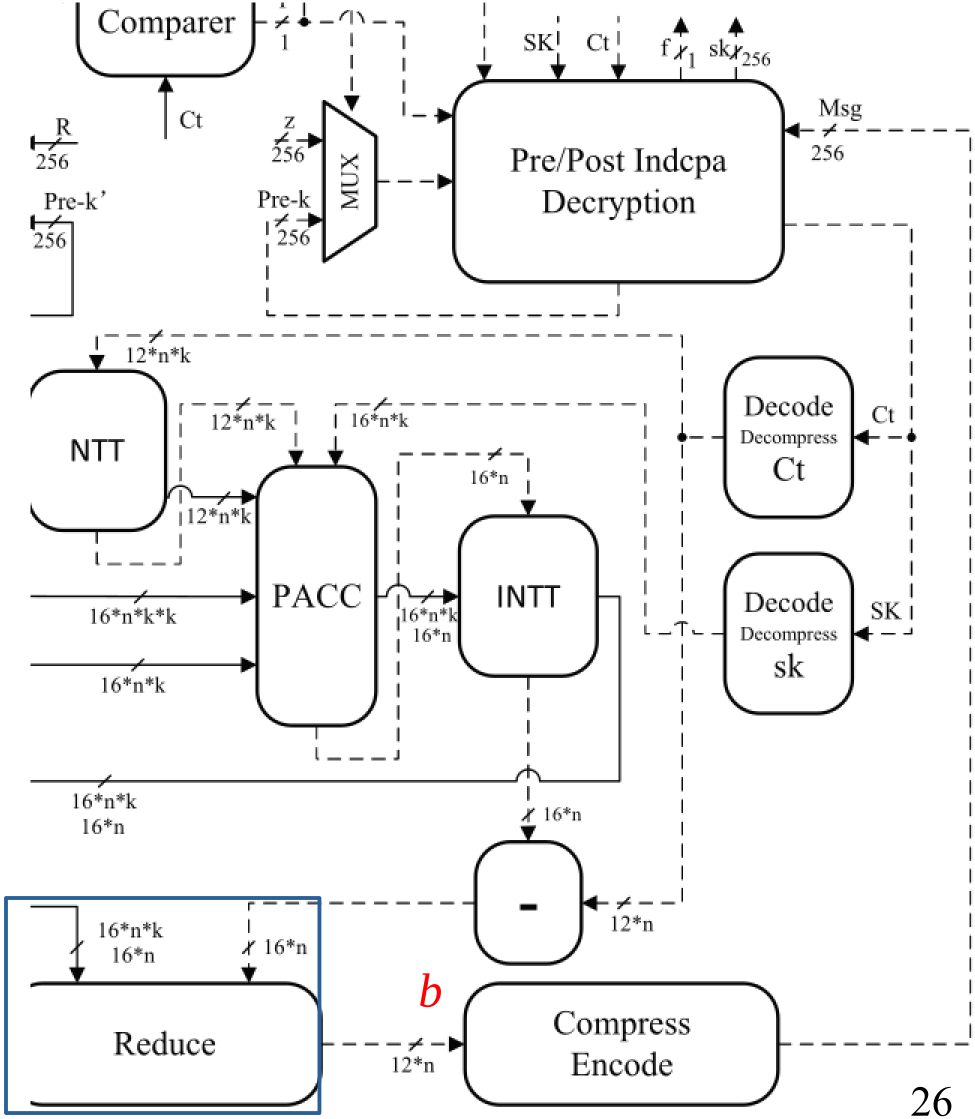
Input : intermediate data b

$$b = P$$

Output : Polynomial Ring R_q

$$b = b \bmod q$$

$$b = R_q$$



Compress Encode

Use $round_q()$ to round the polynomial back to plain text message

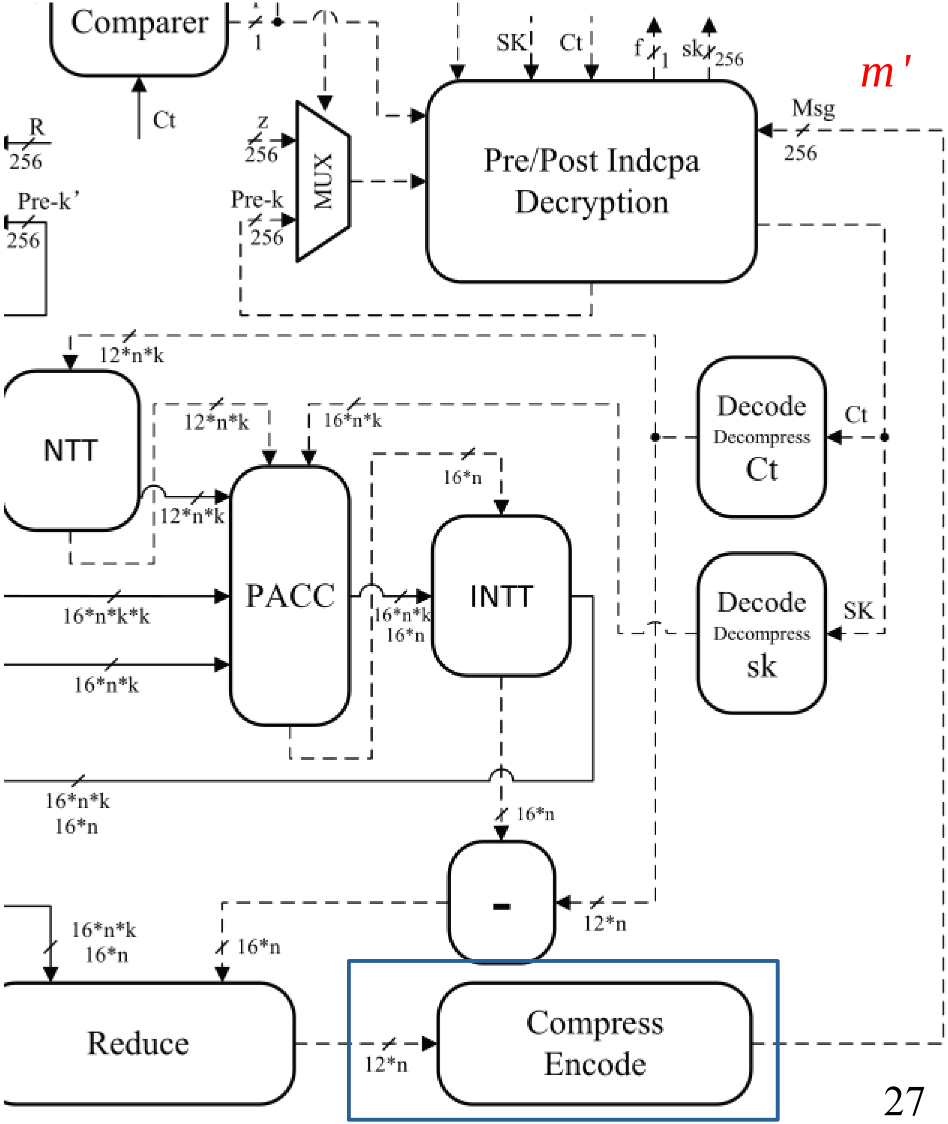
Input : polynomial Ring b

$b = R_q$

Output : Plain text Msg

$m' = Round_q(b) = Round_q(v' - s^T u')$

$m' = \{0,1\}^{256}$



Post Indcpa Decryption

Re-encrypted the plain text message again and compare new and received cipher text

Input

1) plain text msg : m'
2) Hash of public key pre-k
3) Cipher text : Ct

$$c_1 = \begin{pmatrix} P \\ P \\ P \end{pmatrix} \text{ with coef} = d_u \text{ bits}$$

$$c_2 = P \text{ with coef} = d_v \text{ bits}$$

$$pre-k, m' = \{0,1\}^{256}$$

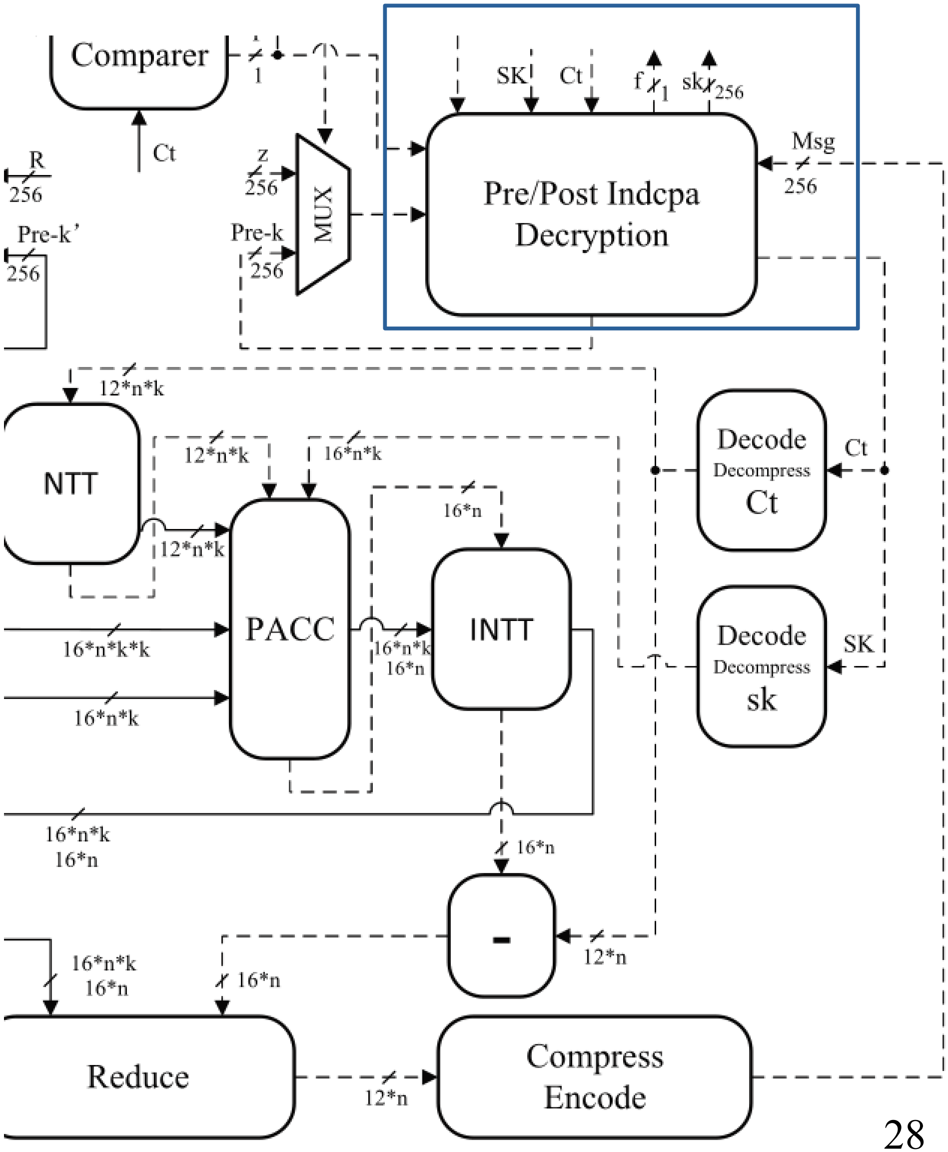
Compute

$$(c', pre-k') = SHA3-512(pre-k, m')$$

Encrypt message gain by using the process as explain in Encryption part (pg. 6-17)

$$Ct' = IND-CPA-KyberEncryption(m', PK, c')$$

Compare if $Ct = Ct'$



Post Indcpa Decryption

Re-encrypted the plain text message again and compare new and received cipher text

Input

1) plain text msg : m'
2) Hash of public key pre-k
3) Cipher text : Ct

$$c_1 = \begin{pmatrix} P \\ P \\ P \end{pmatrix} \text{ with coef} = d_u \text{ bits}$$

$$c_2 = P \text{ with coef} = d_v \text{ bits}$$

$$pre-k, m' = \{0,1\}^{256}$$

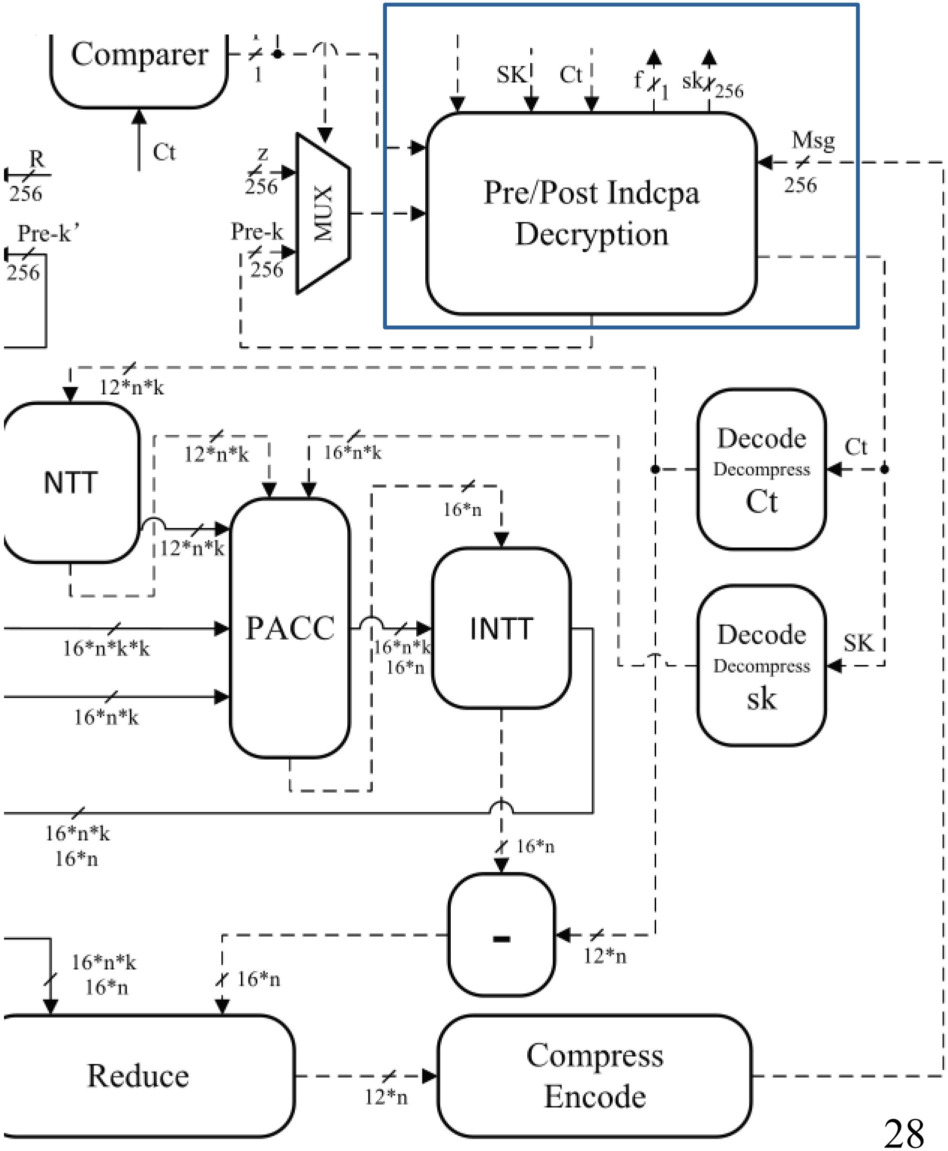
Compute

$$(c', pre-k') = SHA3-512(pre-k, m')$$

Encrypt message gain by using the process as explain in Encryption part (pg. 6-17)

$$Ct' = IND-CPA-KyberEncryption(m', PK, c')$$

Compare if $Ct = Ct'$



Post Indcpa Decryption

If $Ct = Ct'$ return the correct share secret key

Output

1) Verify flag : f = True
2) Share secret key

$$ss = SHAKE-265(SHA3-256(Ct), coin)$$

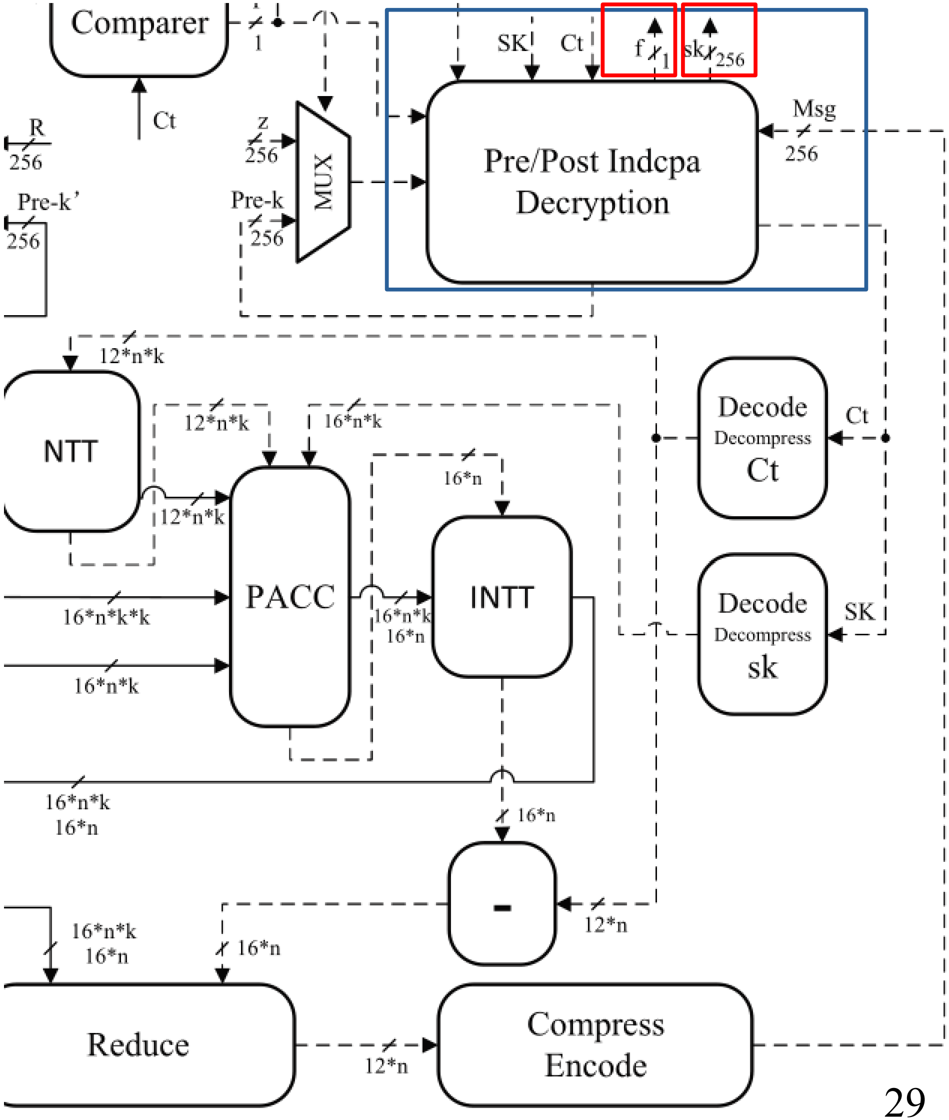
If $Ct \neq Ct'$ return the correct share secret key

Output

1) Verify flag : f = False
2) Share secret key(fake)

$$ss = SHAKE-265(SHA3-256(Ct), pre-k')$$

$f = boolean \quad ss = \{0,1\}^{256}$



FPGAs Board used in the Aritcle

AC701

Logic Cells	215,360
DSP Slices	740
Memory	13,140
GTP 6.6Gb/s Transceivers	16
I/0 Pins	500

VC707

Logic Cells	485,760
Memory (Kb)	37,080
DSP Slices	2,800
GTX 12.5 GB/s Transceivers	56
I/0 Pins	700

My board : ArtyS7-25

Logic Cells	23,360
Slices	3,650
Flip-flops	29,200
Block RAM (Kbits)	1,620
Clock Management Tiles	3

I have about 10% resources of the board used in the Research

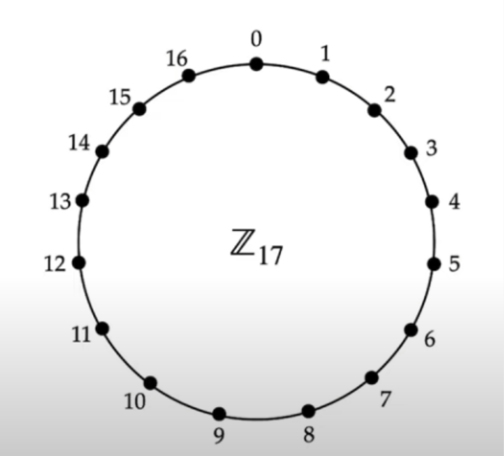
Future work

- Start implementing each module
- Understanding arithmetic pipeline in NTT and Hash module
- Dual port BRAM : usage as FIFO buffer

Reference to Math Notation

- Integer modulo : $\mathbb{Z}_q$
  - Integer 0,1,2,…, q-1
  - After math operation → mod q

e.g. 15+3 = 18%17 = 1



- Polynomial ring : $R_q = \mathbb{Z}_q[x]/(x^n + 1)$
  - q , n
  - Coefficient in $\mathbb{Z}_q \in \{0,1,2, \dots, q-1\}$
  - Degree = $n - 1$ → $a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$
  - After multiplying apply modular reduction $/(x^n + 1)$

e.g. q = 17, n = 4

f(x) = $2 + 16x + 3x^2 + 5x^3$

g(x) = $9 + x + 14x^3$

$$f(x)g(x) = 18 + 146x + 43x^2 + 76x^3 + 229x^4 + 42x^5 + 70x^6$$

$\Downarrow$ coef. mod q

$$= 1 + 10x + 9x^2 + 8x^3 + 8x^4 + 8x^5 + 2x^6$$

$\Downarrow$ modular reduction $/(x^n + 1)$

$$= 10 + 2x + 7x^2 + 8x^3$$

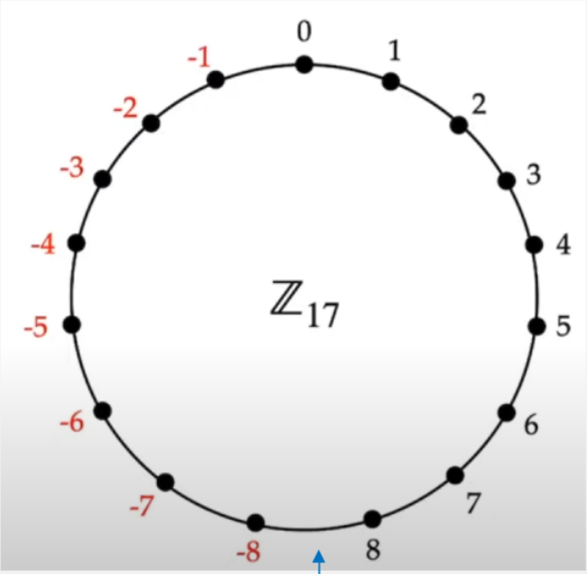
“ Small” Polynomial : S_η

- η (eta) : coefficient $\in [-\eta, \eta]$ when written in symmetric mod
- e.g. $q = 17,\ n = 4, \eta = 2$
- Coefficient can be -2, -1, 0, 1, 2

$$s(x) = 1 + x - 2x^3$$

Noted : $s(x)$ is written is symmetric mods

Symmetric mod → mods



$(q-1)/2$

Kyber-768 domain Parameters

$q = 3329$	Polynomial coefficient can be : 0,1,2,…3328
$n = 256$	Each polynomial has degree 255 : $a_0 + a_1 x + \cdots + a_{255} x^{255}$
$k = 3$	For matrix size
$\eta_1 = 2$	size of "small" polynomial
$\eta_2 = 2$	size of "small" polynomial
$d_u = 10$	for compression/decompression of ciphertext
$d_v = 4$	for compression/decompression of ciphertext

Refernece for Kyber

Kyber Algorithms

https://cryptography101.ca/kyber-dilithium/

Kyber Implementation

https://doi.org/10.1587/elex.17.20200234

Kyber Specification

NIST: Post-Quantum Cryptography Standardization https://csrc.nist.gov/Projects/post-quantum-cryptography.

KYBER-PKE

- Key generation
- Encryption
- Decryption

<u>Hash function:</u> generate A : SHAKE 128

- Base on Learning with errors problem
- Public key encryption
- Security level : "secure against chosen plain text attack" → lower than Kyber-KEM

Domain parameters and key generation

For concreteness, we'll use the ML-KEM-768 domain parameters:

- $q = 3329$
- $n = 256$
- $k = 3$
- $\eta_1 = 2$ and $\eta_2 = 2$
- $d_u = 10$ and $d_v = 4$

Kyber-PKE key generation: Alice does:

1. Select $\rho \in_R \{0,1\}^{256}$ and compute $A = \text{Expand}(\rho)$, where $A \in R_q^{k \times k}$.
2. Select $s \in_{CBD} S_{\eta_1}^k$ and $e \in_{CBD} S_{\eta_2}^k$.
3. Compute $t = As + e$.
4. Alice's encryption (public) key is (ρ, t); her decryption (private) key is s.

Encryption and decryption

Kyber-PKE encryption: To encrypt a message $m \in \{0,1\}^n$ for Alice, Bob does:

1. Obtain an authentic copy of Alice's encryption key (ρ, t) and compute $A = \text{Expand}(\rho)$.
2. Select $r \in_{CBD} S_{\eta_1}^k$, $e_1 \in_{CBD} S_{\eta_2}^k$ and $e_2 \in_{CBD} S_{\eta_2}$.
3. Compute $u = A^T r + e_1$ and $v = t^T r + e_2 + \lceil \frac{q}{2} \rfloor m$.
4. Compute $c_1 = \text{Compress}_q(u, d_u)$ and $c_2 = \text{Compress}_q(v, d_v)$.
5. Output $c = (c_1, c_2)$.

Kyber-PKE decryption: To decrypt $c = (c_1, c_2)$, Alice does:

1. Compute $u' = \text{Decompress}_q(c_1, d_u)$ and $v' = \text{Decompress}_q(c_2, d_v)$.
2. Compute $m = \text{Round}_q(v' - s^T u')$.

KYBER-KEM

- Key generation
- Encapsulation
- Decapsulation

Hash function:
generate A : SHAKE 128
G is SHA3-512, H is SHA3-256, J is SHAKE256

- Kyber PKE alone is not secure enough. Applying "Fujisaki Okamoto" transform → Kyber KEM
- Key Encapsulation mechanism
- Using Kyber-PKE as base building block
- Use Hash() and seed to create Pseudo RNG alongside
  - pure random
  - Central binomial Distribution
- Security : "Secure against chosen ciphertext attack" (more secure than Kyber-PKE)

Domain parameters and key generation

For concreteness, we'll use the ML-KEM-768 domain parameters:

- $q = 3329$
- $n = 256$
- $k = 3$
- $\eta_1 = 2$ and $\eta_2 = 2$
- $d_u = 10$ and $d_v = 4$

Kyber-KEM key generation: Alice does:

1. Use the Kyber-PKE key generation algorithm to select a Kyber-PKE encryption key (ρ, t) and decryption key s.
2. Select $z \in_R \{0,1\}^{256}$.
3. Alice's encapsulation key is $ek = (\rho, t)$; her decapsulation key is $dk = (s, ek, H(ek), z)$.

Encapsulation and decapsulation

Kyber-KEM encapsulation: To establish a shared secret key with Alice, Bob does:

1. Obtain an authentic copy of Alice's encapsulation key ek.
2. Select $m \in_R \{0,1\}^{256}$.
3. Compute $h = H(ek)$ and $(K, R) = G(m, h)$, where $K, R \in \{0,1\}^{256}$.
4. Use the Kyber-PKE encryption algorithm to encrypt m with encryption key ek, and using R to generate the random quantities needed; call the resulting ciphertext c.
5. Output the secret key K and ciphertext c.

Kyber-KEM decapsulation: To recover the secret key K from c using $dk = (s, ek, H(ek), z)$, Alice does:

1. Use the Kyber-PKE decryption algorithm to decrypt c using decryption key s; call the resulting plaintext m'.
2. Compute $(K', R') = G(m', H(ek))$.
3. Compute $\overline{K} = J(z, c)$.
4. Use the Kyber-PKE encryption algorithm to encrypt m' with encryption key ek, and using R' to generate the random quantities needed; call the resulting ciphertext c'.
5. If $c \neq c'$ then return$(\overline{K})$.
6. Return(K').