



PHOENIX

Crypto-Agile Hardware Sharing for ML-KEM and HQC

Antonio Ras, PhD Candidate

Laboratoire Sécurité des Composants (LSCO), CEA-Leti

Encadrants CEA :

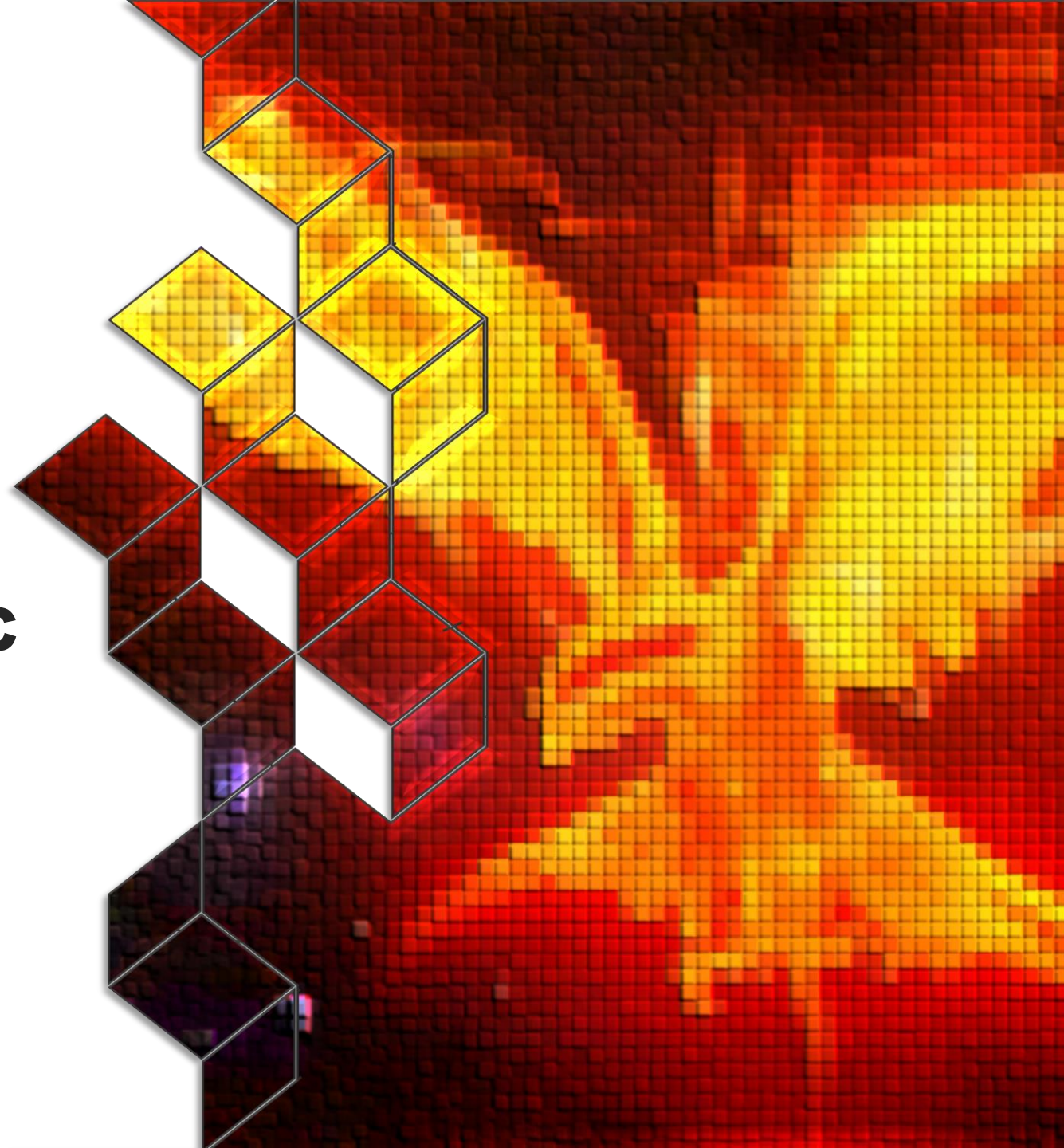
Mikael Carmona, Antoine Loiseau, Simon Pontié, Emanuele Valea

Directeurs de thèse :

Guénaél Renault, Benjamin Smith



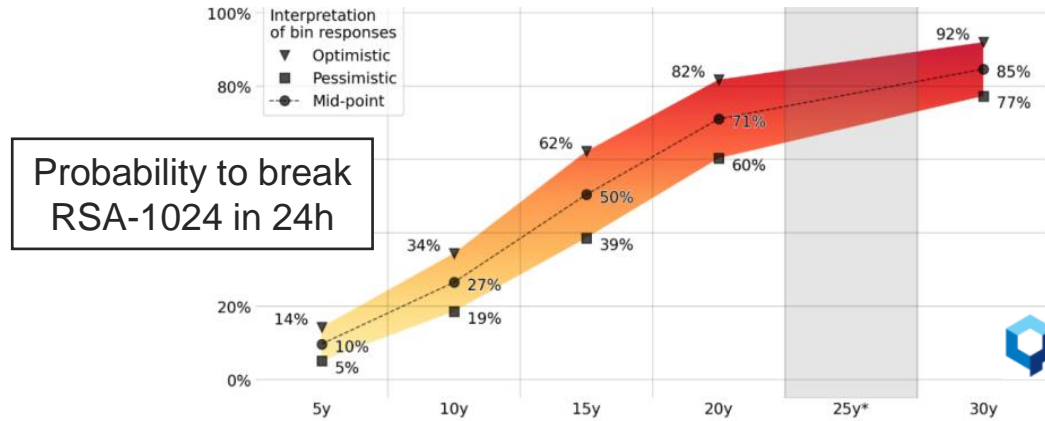
Workshop PHISIC 2025 | 21 May 2025



Introduction

➤ Quantum-threat

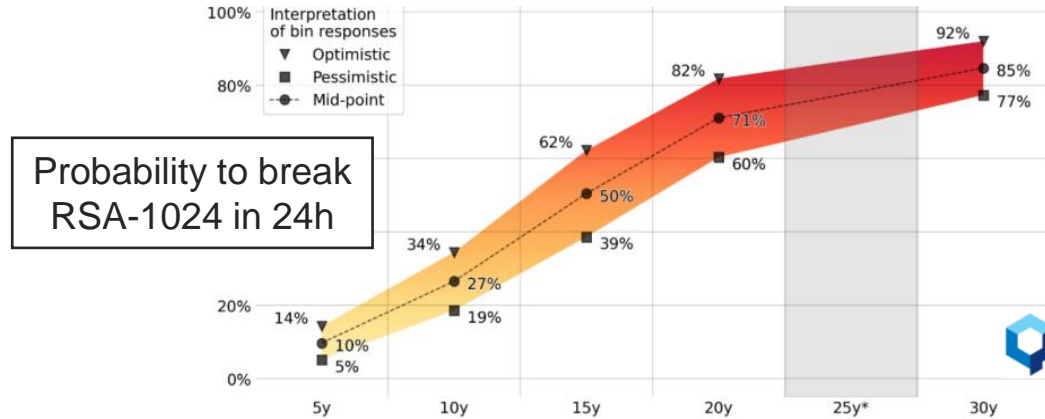
- ❑ Shor algorithm can easily solve hard problem
 - ❑ Discrete logarithm ([Elliptic Curve Cryptography](#))
 - ❑ Integer factorization ([RSA](#))



Introduction

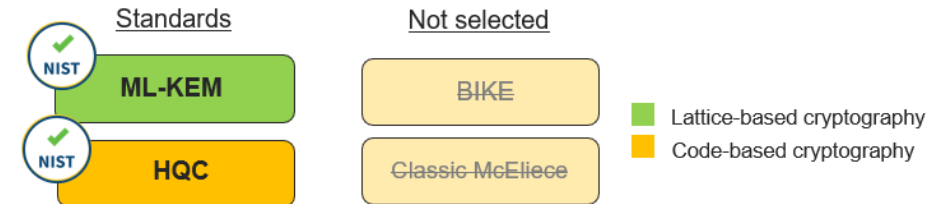
➤ Quantum-threat

- ❑ Shor algorithm can easily solve hard problem
 - ❑ Discrete logarithm ([Elliptic Curve Cryptography](#))
 - ❑ Integer factorization ([RSA](#))



➤ Post-Quantum Cryptography

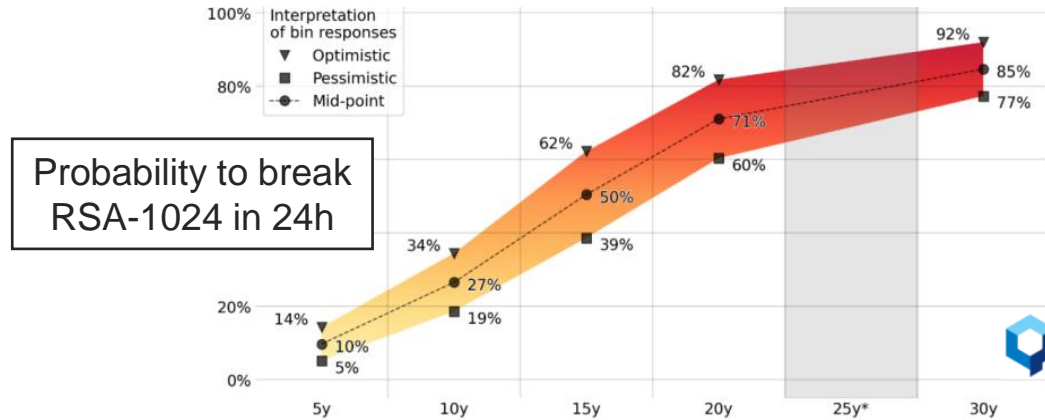
- ❑ New mathematical hard problem
 - ❑ Lattice-based and Code-based cryptography
 - ❑ ...
- ❑ **Key Encapsulation Mechanism (KEM)**
- ❑ Digital Signature (DS)
- ❑ End of PQC NIST Process (2017 → 2025)



Introduction

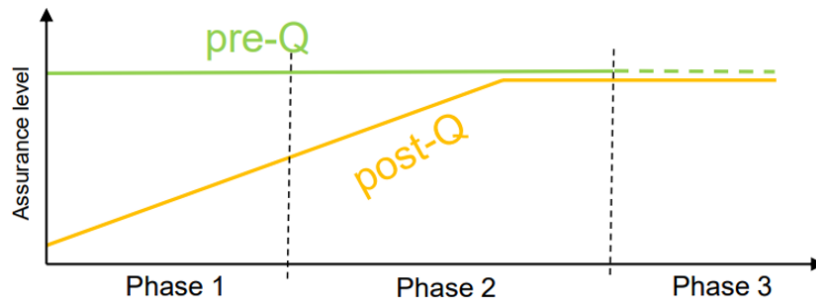
➤ Quantum-threat

- ❑ Shor algorithm can easily solve hard problem
 - ❑ Discrete logarithm ([Elliptic Curve Cryptography](#))
 - ❑ Integer factorization ([RSA](#))



➤ Quantum-safe transition

- ❑ Hybridization : *Post-quantum* + *pre-quantum* cryptography
- ❑ Crypto-Agility : Ability to switch between different PQC solutions



➤ Post-Quantum Cryptography

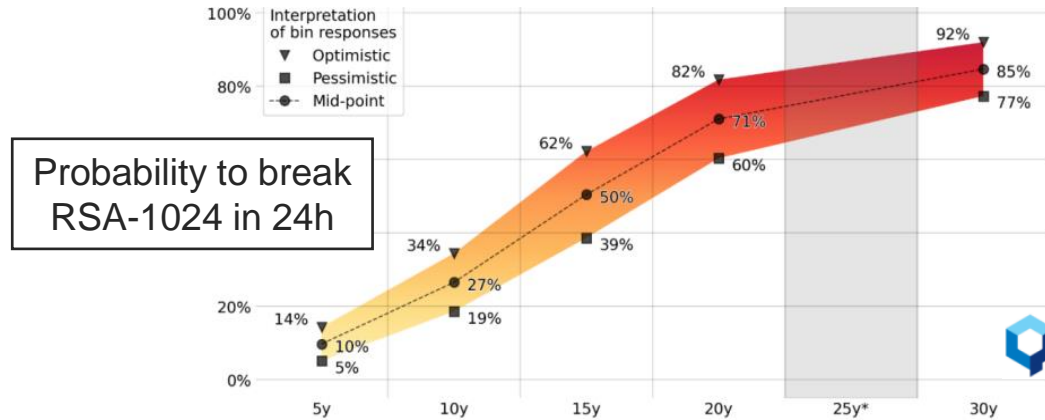
- ❑ New mathematical hard problem
 - ❑ *Lattice-based* and *Code-based* cryptography
 - ❑ ...
- ❑ Key Encapsulation Mechanism (KEM)
- ❑ Digital Signature (DS)
- ❑ End of PQC NIST Process (2017 → 2025)



Introduction

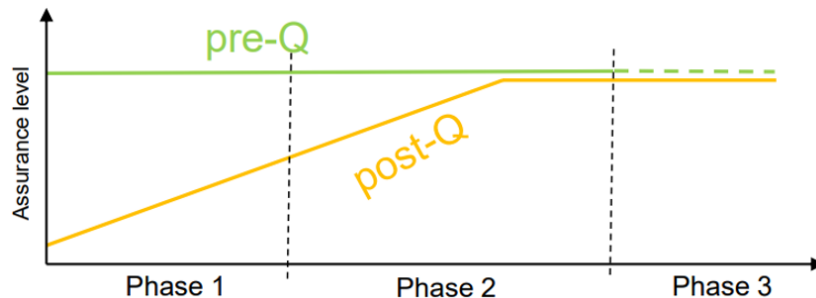
➤ Quantum-threat

- ❑ Shor algorithm can easily solve hard problem
 - ❑ Discrete logarithm ([Elliptic Curve Cryptography](#))
 - ❑ Integer factorization ([RSA](#))



➤ Quantum-safe transition

- ❑ Hybridization : *Post-quantum* + *pre-quantum* cryptography
- ❑ Crypto-Agility : Ability to switch between different PQC solutions



➤ Post-Quantum Cryptography

- ❑ New mathematical hard problem
 - ❑ *Lattice-based* and *Code-based* cryptography
 - ❑ ...
- ❑ Key Encapsulation Mechanism (KEM)
- ❑ Digital Signature (DS)
- ❑ End of PQC NIST Process (2017 → 2025)



➤ Crypto-Agility

- ❑ Purpose : **Maintain security along PQC families**
- ❑ Our targeted agility + Hardware :
 - ❑ ML-KEM (lattice-based) + HQC (code-based)

	ML-KEM	HQC	Lattice-code agility
Hardware implementation	~40	3	2

Agenda

1. **Crypto-Agility on NIST standards**
2. **Number-Theoretic Transform (NTT)**
3. **HQC using alternative multiplication**
4. **Proposed sharing strategy**
5. **Integration results**
6. **Conclusion**



1 ■ **Crypto-Agility on NIST standards**

A case study for ML-KEM and HQC

Targeted Crypto-agility

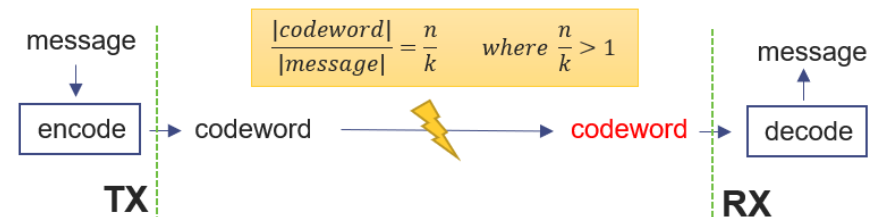
ML-KEM : *Module-Learning with Error*, lattice problem

$$\begin{bmatrix} t_0 \\ \vdots \\ t_{k-1} \end{bmatrix} = \begin{bmatrix} a_{00} & \cdots & a_{0(k-1)} \\ \vdots & \ddots & \vdots \\ a_{(k-1)0} & \cdots & a_{(k-1)(k-1)} \end{bmatrix} \times \begin{bmatrix} s_0 \\ \vdots \\ s_{k-1} \end{bmatrix} + \begin{bmatrix} e_0 \\ \vdots \\ e_{k-1} \end{bmatrix}$$

public A , uniform secret s , e binomial

Knowing $\mathbf{t} = A\mathbf{s} + \mathbf{e}$ and A , it is difficult to find s

Hamming Quasi-Cyclic (HQC): *Quasi-Cyclic Syndrome Decoding*, code-corrector problem



Knowing $s = eH^t$ and H , it is difficult to find the small e

Targeted Crypto-agility

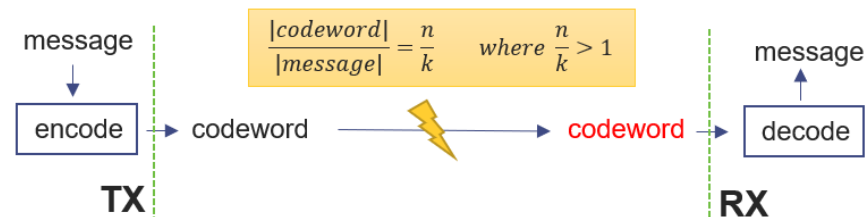
ML-KEM : *Module-Learning with Error*, lattice problem

$$\begin{bmatrix} t_0 \\ \vdots \\ t_{k-1} \end{bmatrix} = \begin{bmatrix} a_{00} & \cdots & a_{0(k-1)} \\ \vdots & \ddots & \vdots \\ a_{(k-1)0} & \cdots & a_{(k-1)(k-1)} \end{bmatrix} \times \begin{bmatrix} s_0 \\ \vdots \\ s_{k-1} \end{bmatrix} + \begin{bmatrix} e_0 \\ \vdots \\ e_{k-1} \end{bmatrix}$$

public A , uniform secret s , e binomial

Knowing $\mathbf{t} = A\mathbf{s} + \mathbf{e}$ and A , it is difficult to find s

Hamming Quasi-Cyclic (HQC): *Quasi-Cyclic Syndrome Decoding*, code-corrector problem



Knowing $s = eH^t$ and H , it is difficult to find the small e

Frameworks seem close, but...

Targeted Crypto-agility

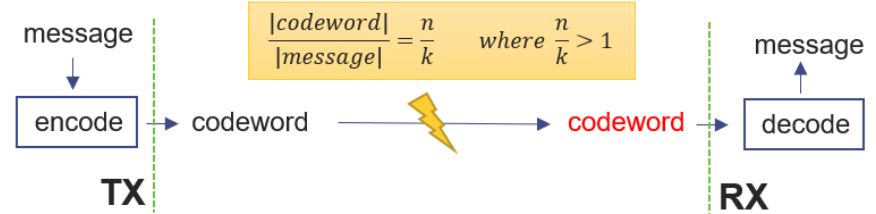
ML-KEM : *Module-Learning with Error*, lattice problem

$$\begin{bmatrix} t_0 \\ \vdots \\ t_{k-1} \end{bmatrix} = \begin{bmatrix} a_{00} & \cdots & a_{0(k-1)} \\ \vdots & \ddots & \vdots \\ a_{(k-1)0} & \cdots & a_{(k-1)(k-1)} \end{bmatrix} \times \begin{bmatrix} s_0 \\ \vdots \\ s_{k-1} \end{bmatrix} + \begin{bmatrix} e_0 \\ \vdots \\ e_{k-1} \end{bmatrix}$$

public A , uniform secret s , e binomial

Knowing $t = As + e$ and A , it is difficult to find s

Hamming Quasi-Cyclic (HQC): *Quasi-Cyclic Syndrome Decoding*, code-corrector problem



Knowing $s = eH^t$ and H , it is difficult to find the small e

Frameworks seem close, but...

	n	q	k	η_1	η_2	d_u	d_v
ML-KEM-512	256	3329	2	3	2	10	4
ML-KEM-768	256	3329	3	2	2	10	4
ML-KEM-1024	256	3329	4	2	2	11	5

$$R_q = \frac{\mathbb{Z}_q}{x^{n+1}}$$

VS

Instance	n_1	n_2	n	w	$w_r = w_e$	security
hqc-128	46	384	17,669	66	75	128
hqc-192	56	640	35,851	100	114	192
hqc-256	90	640	57,637	131	149	256

$$\mathcal{R}_2 = \frac{\mathbb{Z}_2}{x^{n+1}}$$

Targeted Crypto-agility

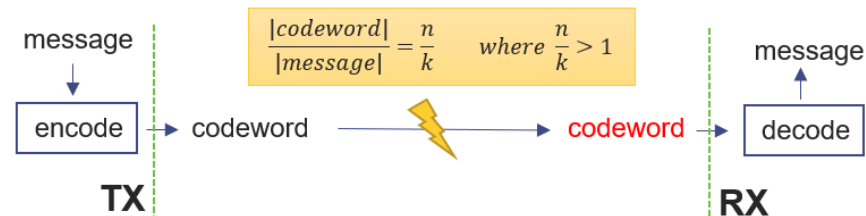
ML-KEM : *Module-Learning with Error*, lattice problem

public A , uniform secret s , e binomial

$$\begin{bmatrix} t_0 \\ \vdots \\ t_{k-1} \end{bmatrix} = \begin{bmatrix} a_{00} & \cdots & a_{0(k-1)} \\ \vdots & \ddots & \vdots \\ a_{(k-1)0} & \cdots & a_{(k-1)(k-1)} \end{bmatrix} \times \begin{bmatrix} s_0 \\ \vdots \\ s_{k-1} \end{bmatrix} + \begin{bmatrix} e_0 \\ \vdots \\ e_{k-1} \end{bmatrix}$$

Knowing $t = As + e$ and A , it is difficult to find s

Hamming Quasi-Cyclic (HQC): *Quasi-Cyclic Syndrome Decoding*, code-corrector problem



Knowing $s = eH^t$ and H , it is difficult to find the small e

Frameworks seem close, but...

	n	q	k	η_1	η_2	d_u	d_v
ML-KEM-512	256	3329	2	3	2	10	4
ML-KEM-768	256	3329	3	2	2	10	4
ML-KEM-1024	256	3329	4	2	2	11	5

$$R_q = \frac{\mathbb{Z}_q}{x^{n+1}}$$

VS

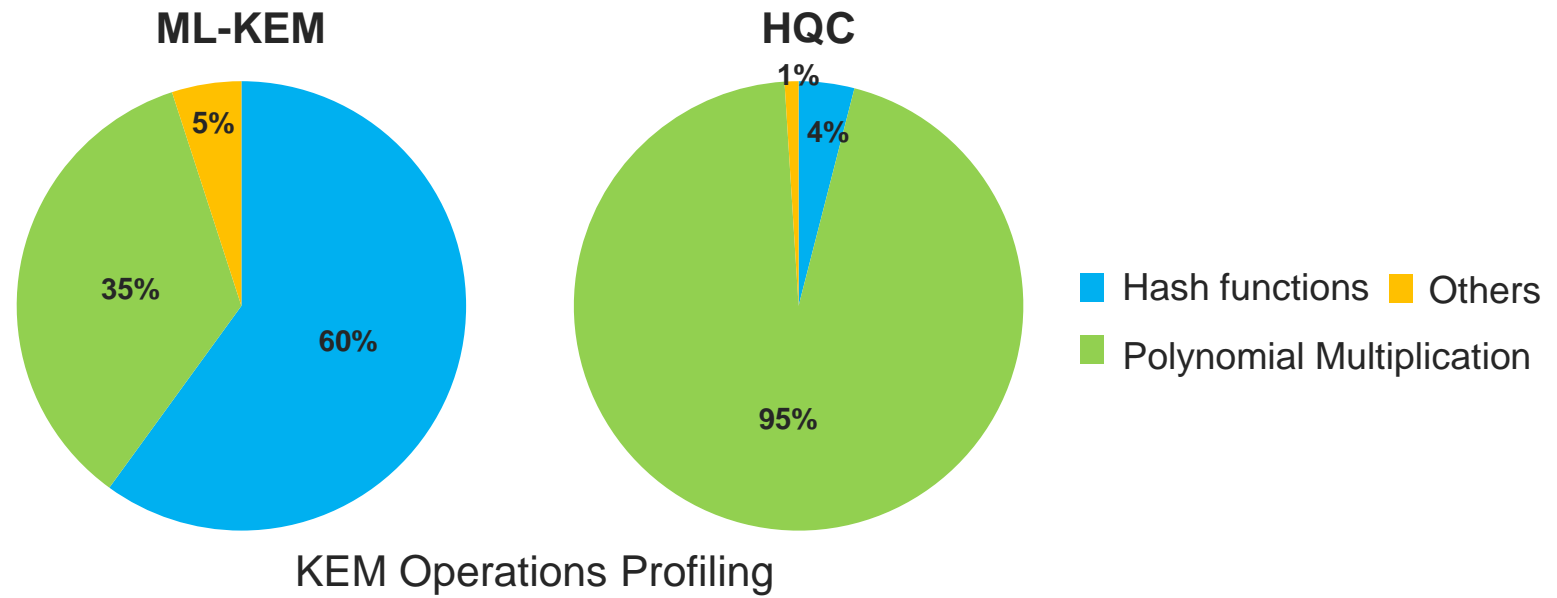
Instance	n_1	n_2	n	w	$w_r = w_e$	security
hqc-128	46	384	17,669	66	75	128
hqc-192	56	640	35,851	100	114	192
hqc-256	90	640	57,637	131	149	256

$$\mathcal{R}_2 = \frac{\mathbb{Z}_2}{x^{n+1}}$$

How to implement **Crypto-Agility** efficiently ?

Sharing strategy

➤ Software profiling



➤ Sharing strategy

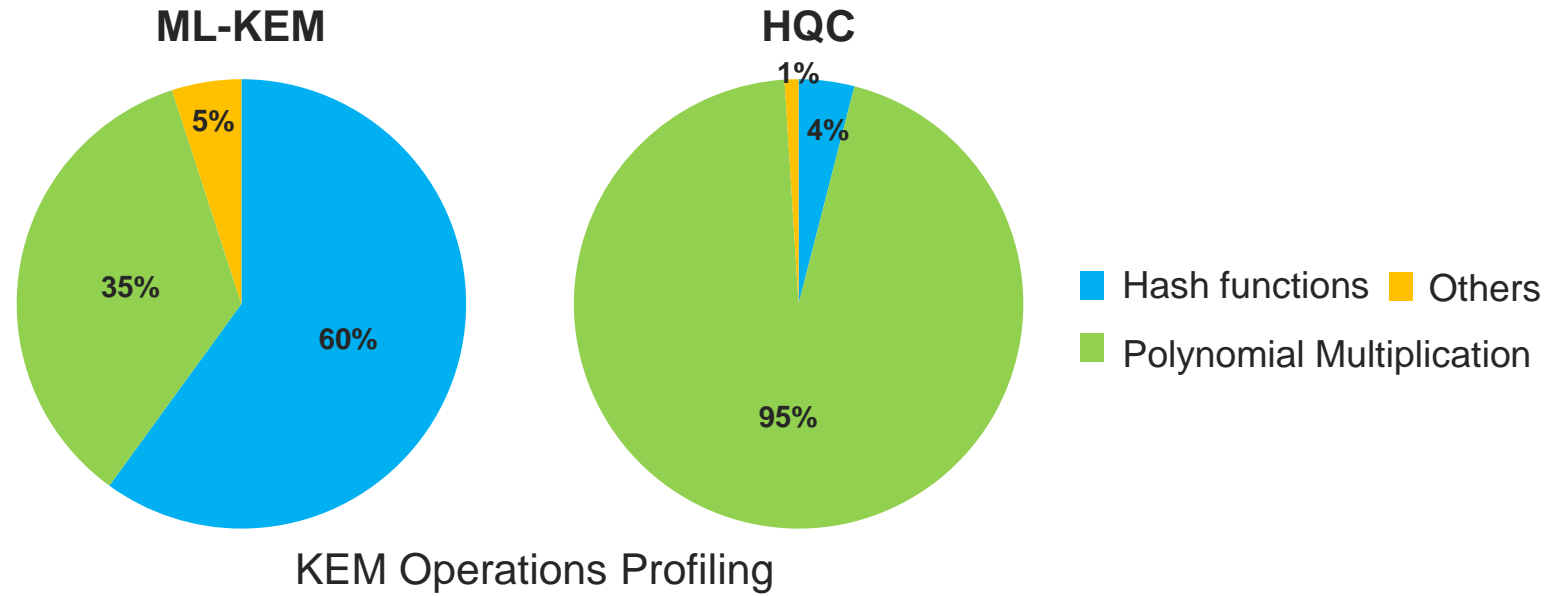
Purpose : Identify common / similar operations

Possible sharing strategies :

- ☐ Hash function : *straightforward* approach, FIPS 202
- ☐ Polynomial multiplication :  approach

Sharing strategy

➤ Software profiling



➤ Sharing strategy

Purpose : Identify common / similar operations

Possible sharing strategies :

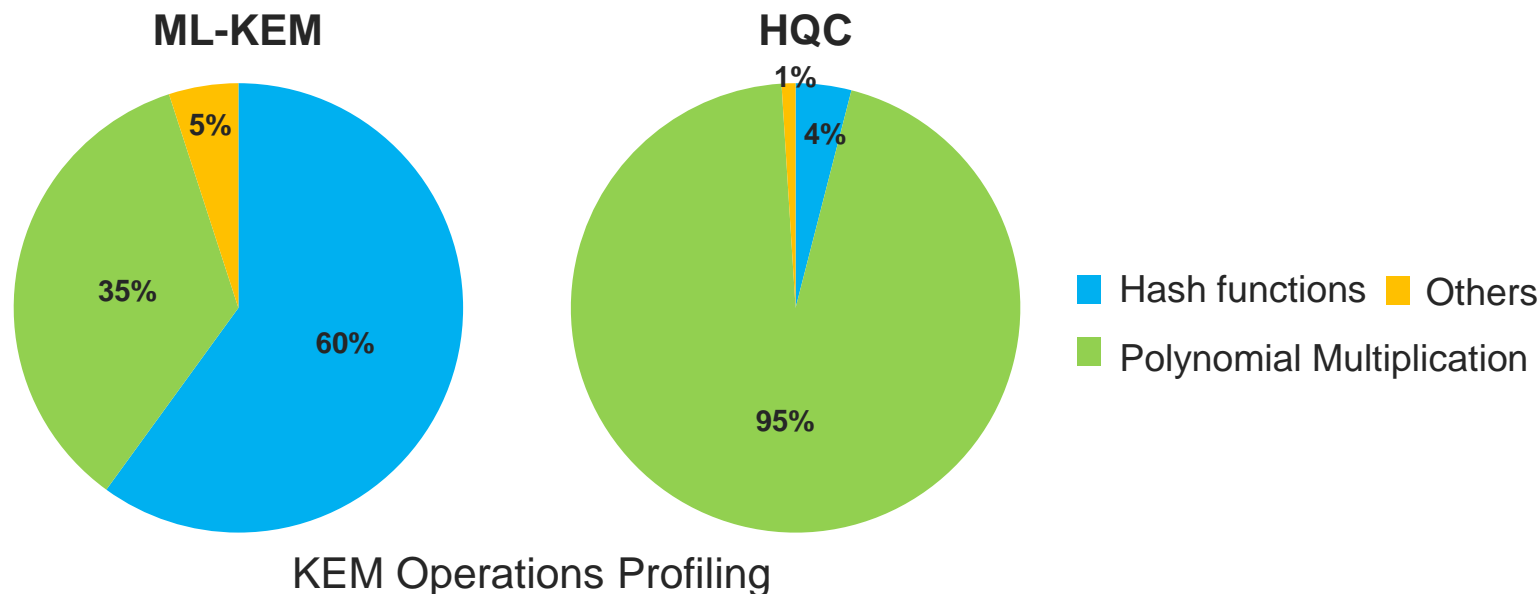
- ❑ Hash function : **straightforward** approach, FIPS 202
- ❑ Polynomial multiplication : **CHALLENGE** approach

Which multiplication strategy to mutualize

ML-KEM	HQC
Number-Theoretic Transform	<i>Sparse - Dense</i> : Not constant time
&	<i>2-way Karatsuba</i> : New bottleneck
	Frobenius Additive FFT : Our proposal

Sharing strategy

➤ Software profiling



➤ Sharing strategy

Purpose : Identify common / similar operations

Possible sharing strategies :

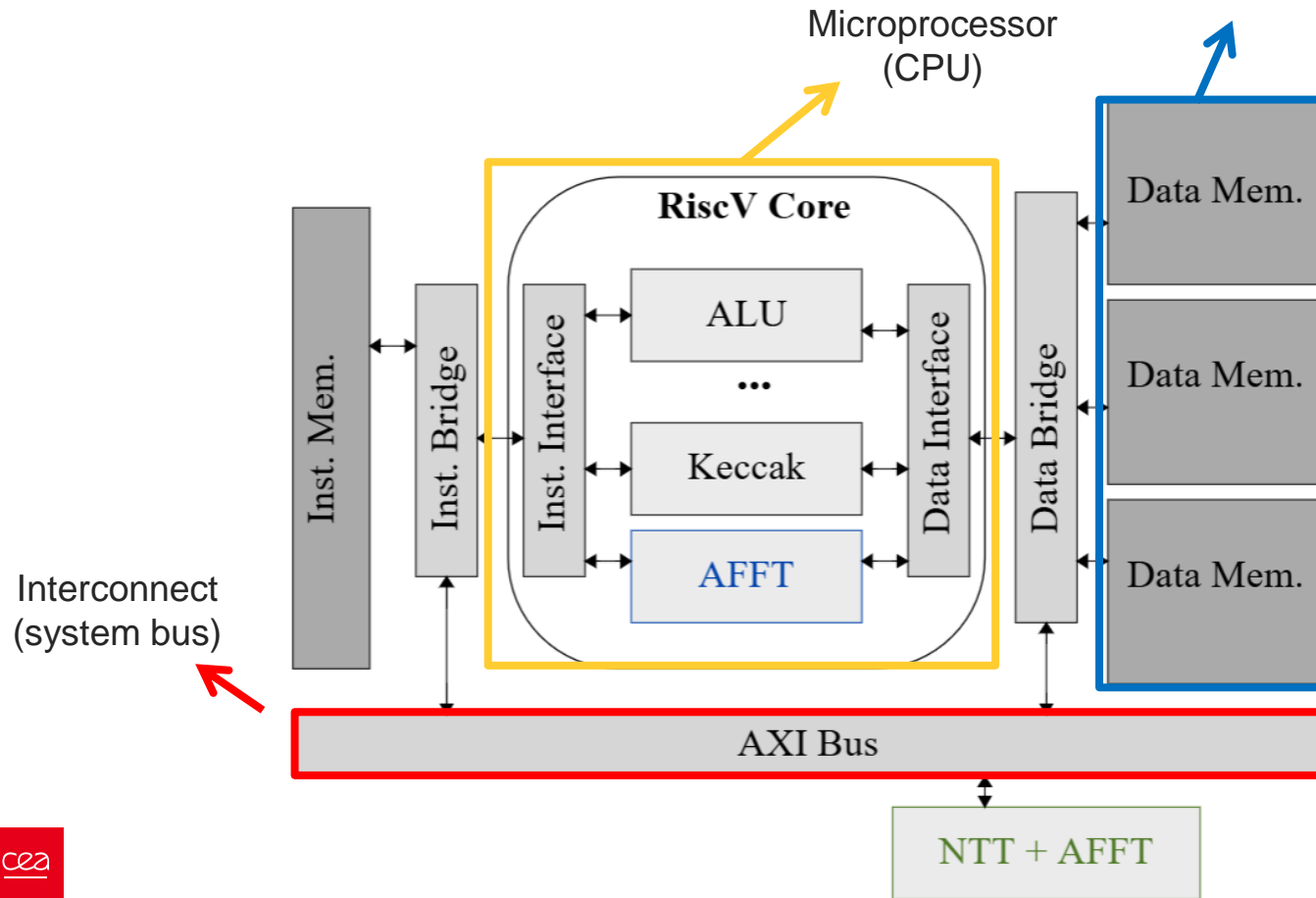
- ❑ Hash function : **straightforward** approach, FIPS 202
- ❑ Polynomial multiplication :  approach

Which multiplication strategy to mutualize

ML-KEM	HQC
Number-Theoretic Transform	<i>Sparse - Dense</i> : Not constant time
&	<i>2-way Karatsuba</i> : New bottleneck
	Frobenius Additive FFT : <i>Our proposal</i>

System on Chip and hardware agility

➤ System-on-Chip

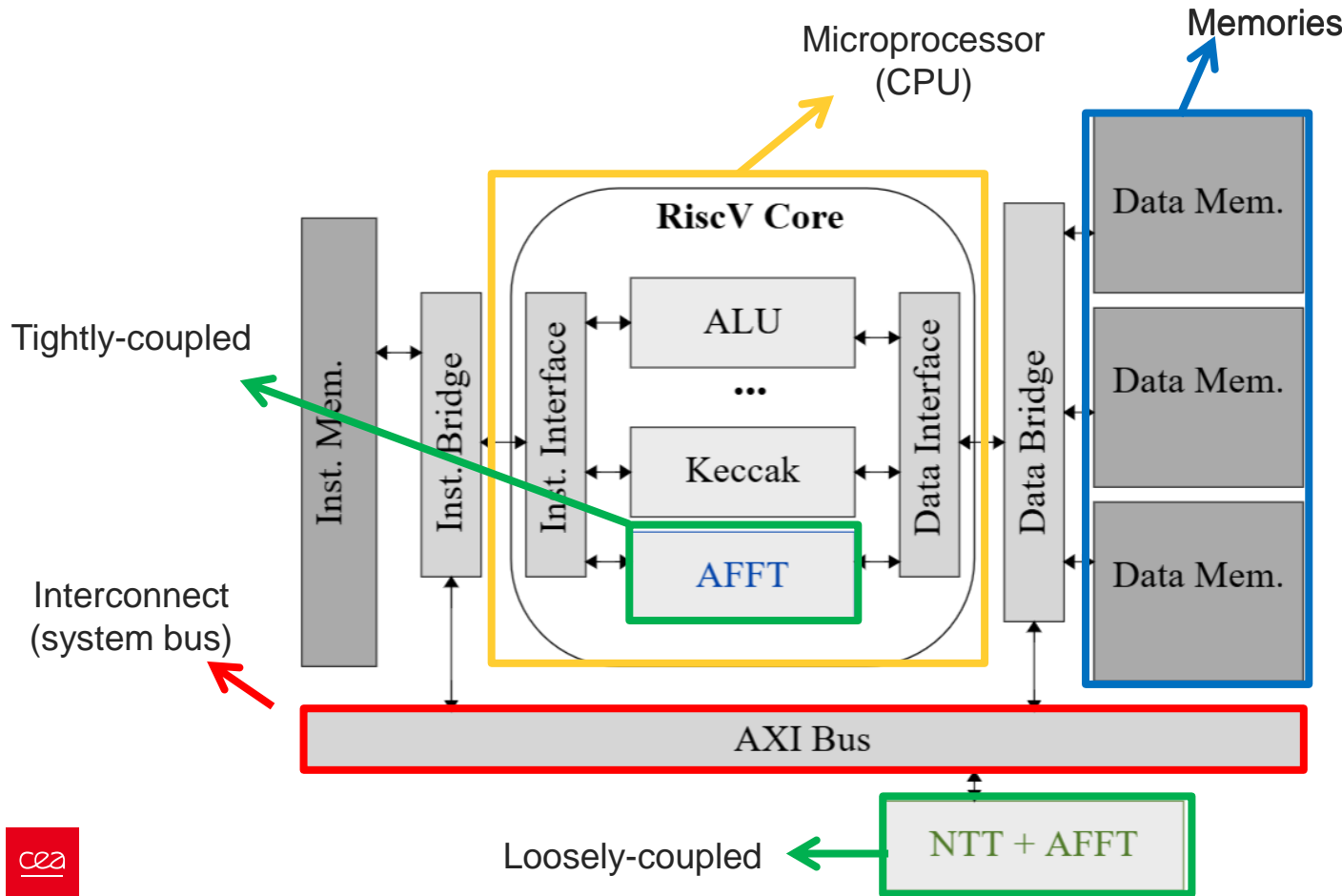


System on Chip and hardware agility

➤ System-on-Chip

❑ Application-specific accelerators

- ❖ **Tightly-coupled** : No flexibility, low data exchange latency
- ❖ **Loosely-coupled** : High flexibility, high data exchange latency





2. ■ Number-Theoretic Transform

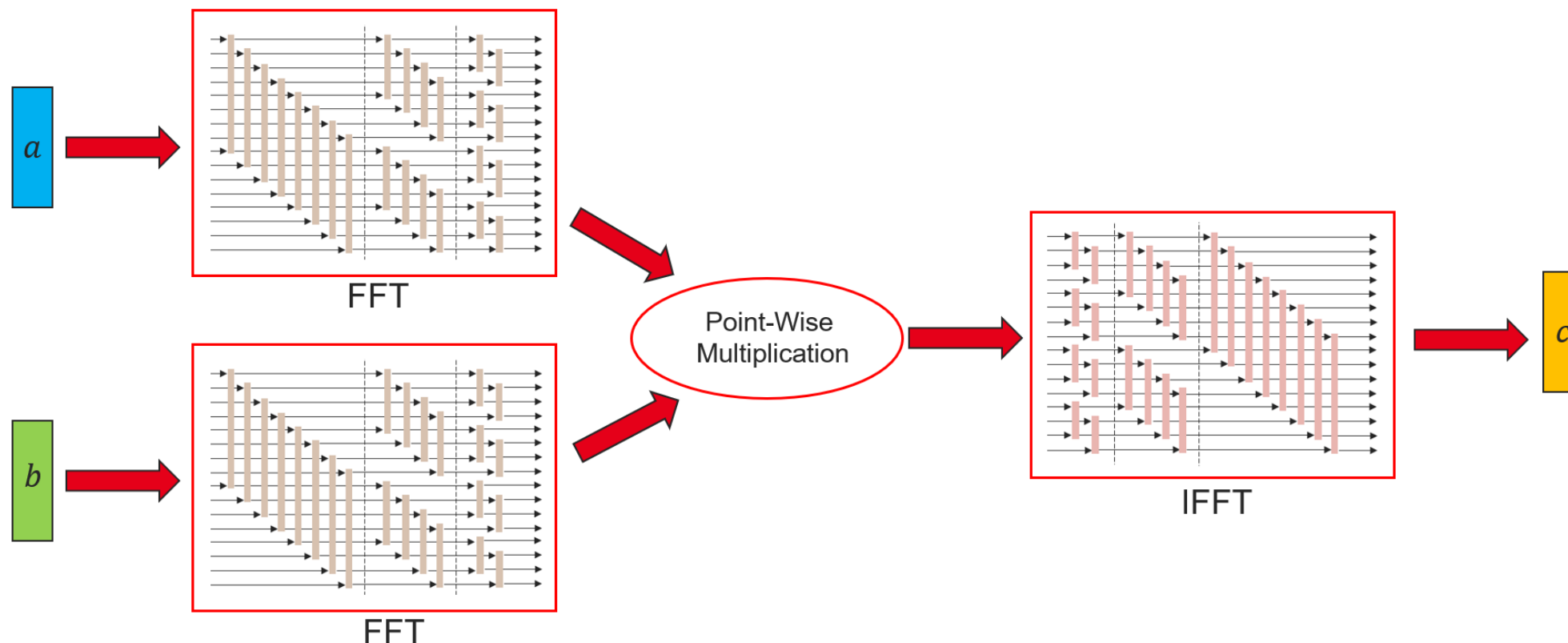
The polynomial multiplication strategy for ML-KEM

FFT-based polynomial multiplication

How to perform multiplication

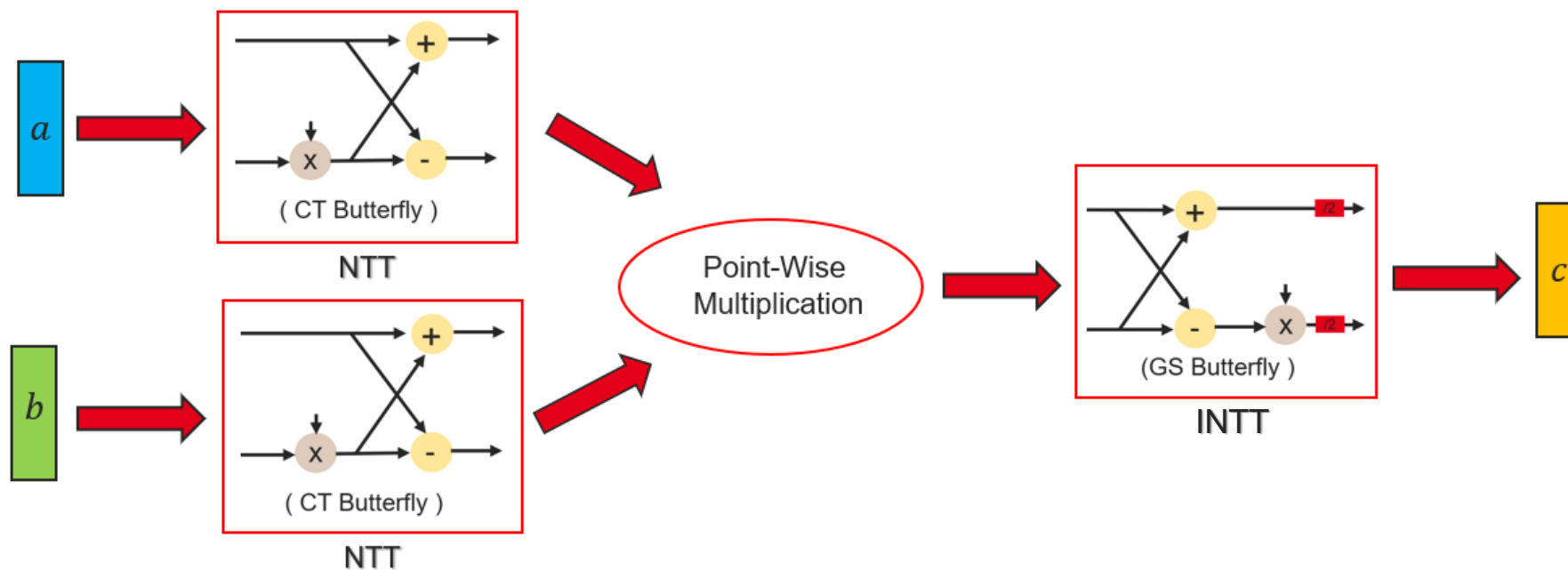
- ❑ Evaluating input polynomial in a different domain using an evaluation points set → **FFT step**
- ❑ Perform multiplication between evaluated points → **PWM step**
- ❑ Interpolate back to get final result → **Inverse FFT (IFFT) step**

➔ **IMPORTANT** : FFT and IFFT are performed using different **processing elements**, and



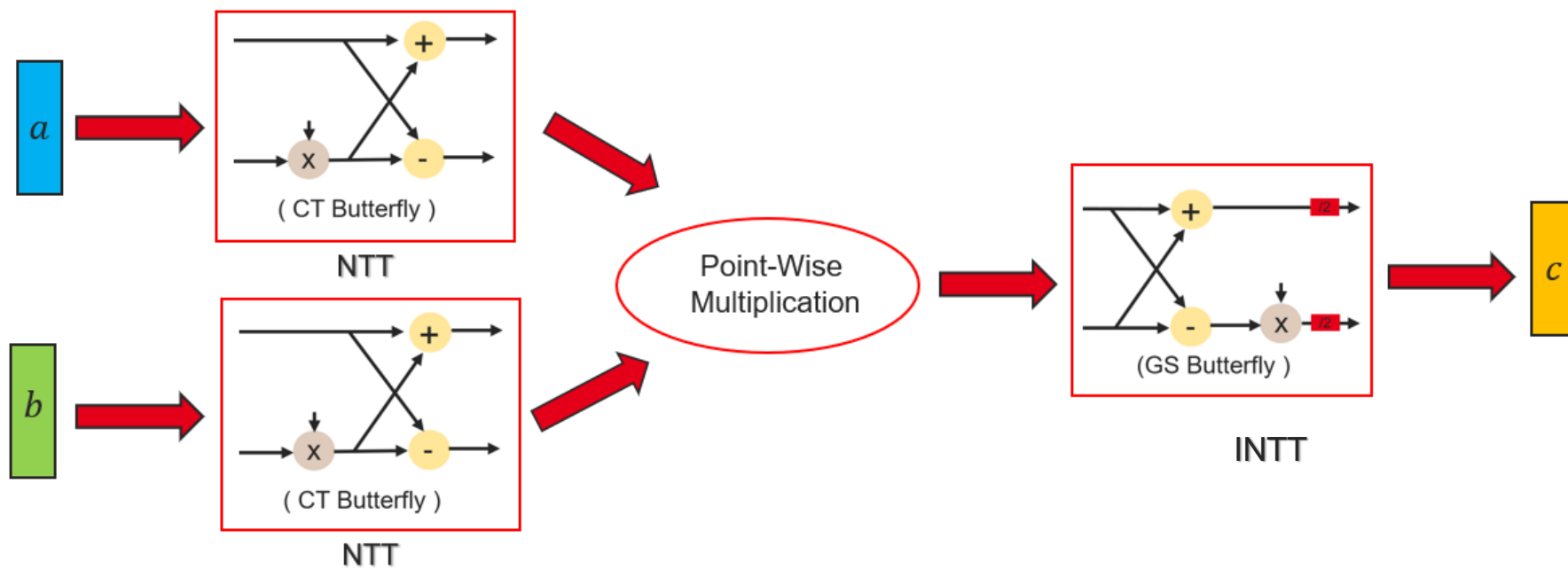
Polynomial multiplication in ML-KEM

- **Number-Theoretic-Transform** : FFT-like for polynomials in $R_q = \frac{\mathbb{Z}_q}{x^{n+1}}$, where $q = 3329$ and $n = 256$.



Polynomial multiplication in ML-KEM

- **Number-Theoretic-Transform** : FFT-like for polynomials in $R_q = \frac{\mathbb{Z}_q}{x^{n+1}}$, where $q = 3329$ and $n = 256$.



Which operation we compute in practice ?

□ \mathbb{Z}_q where $q = 3329$ (12-bits coefficient)

Modular addition

$$h = (f + g) \bmod q$$

Modular subtraction

$$h = (f - g) \bmod q$$

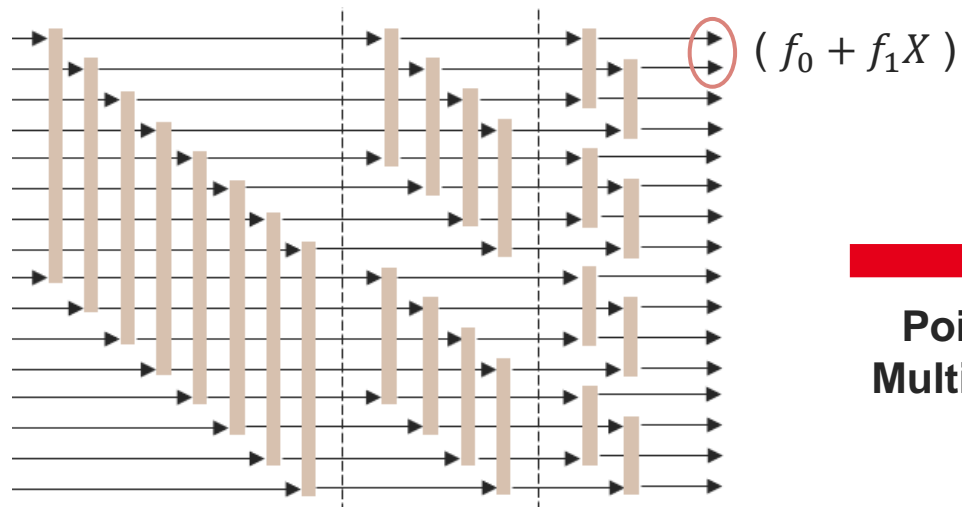
Modular integer multiplication

$$h = (f * g) \bmod q$$

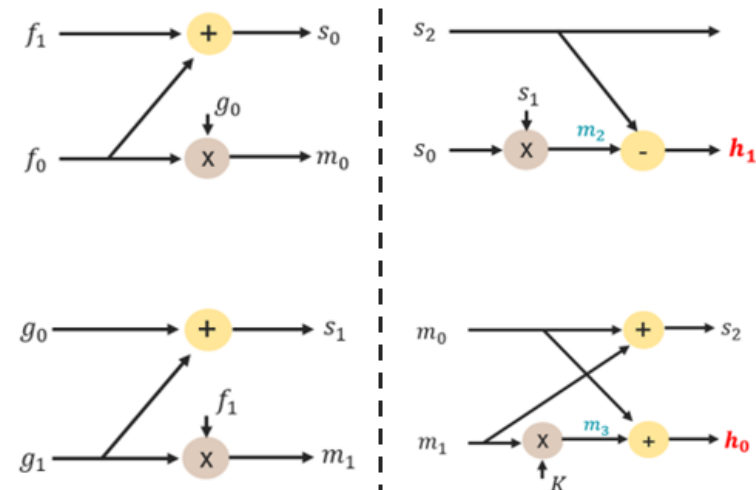
Barrett or Montgomery reduction

Polynomial multiplication in ML-KEM

- **Incomplete NTT** : Factorize modulo $X^n + 1$ into $\frac{n}{2}$ polynomials of *degree* 1.



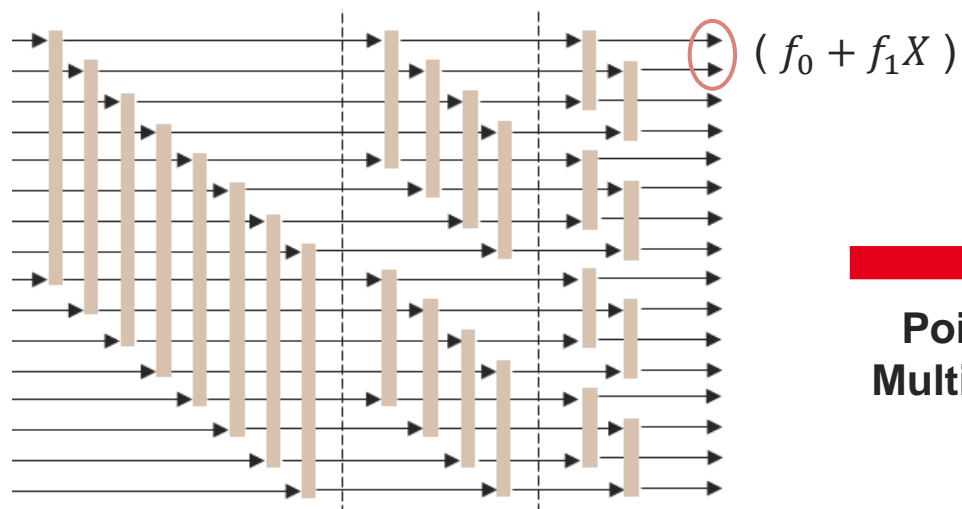
**Point-wise
Multiplication**



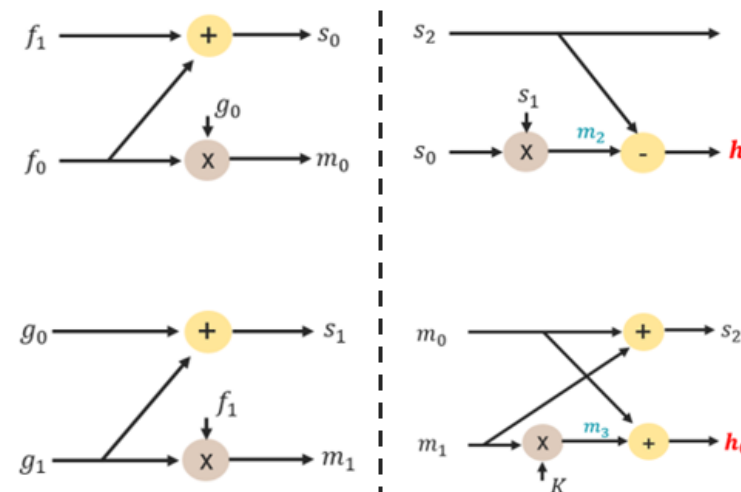
2-way Karatsuba : $h_0 = f_0g_0 + f_1g_1K$; $h_1 = (f_0+f_1)(g_0+g_1) - (f_0g_0 + f_1g_1)$

Polynomial multiplication in ML-KEM

- **Incomplete NTT** : Factorize modulo $X^n + 1$ into $\frac{n}{2}$ polynomials of *degree* 1.



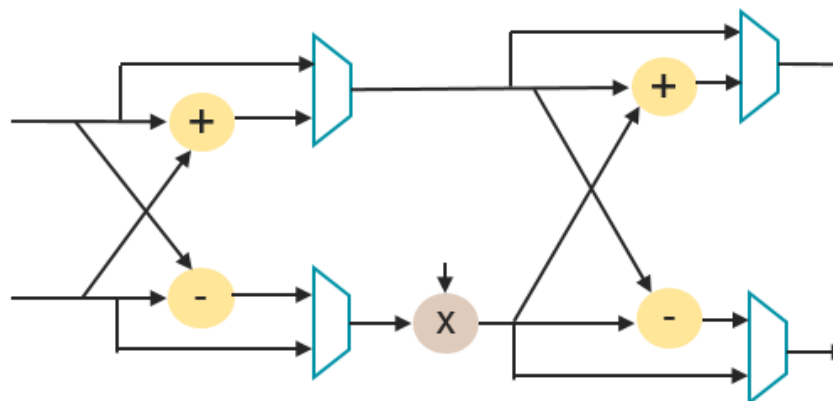
**Point-wise
Multiplication**



2-way Karatsuba : $h_0 = f_0 g_0 + f_1 g_1 K$; $h_1 = (f_0 + f_1)(g_0 + g_1) - (f_0 g_0 + f_1 g_1)$

➤ **Hardware State-of-the-Art**

- ❑ Example of fully configurable hardware design
- ❑ Include CT, GS and PWM butterflies
- ❑ Same polynomial ring → **easy to design**





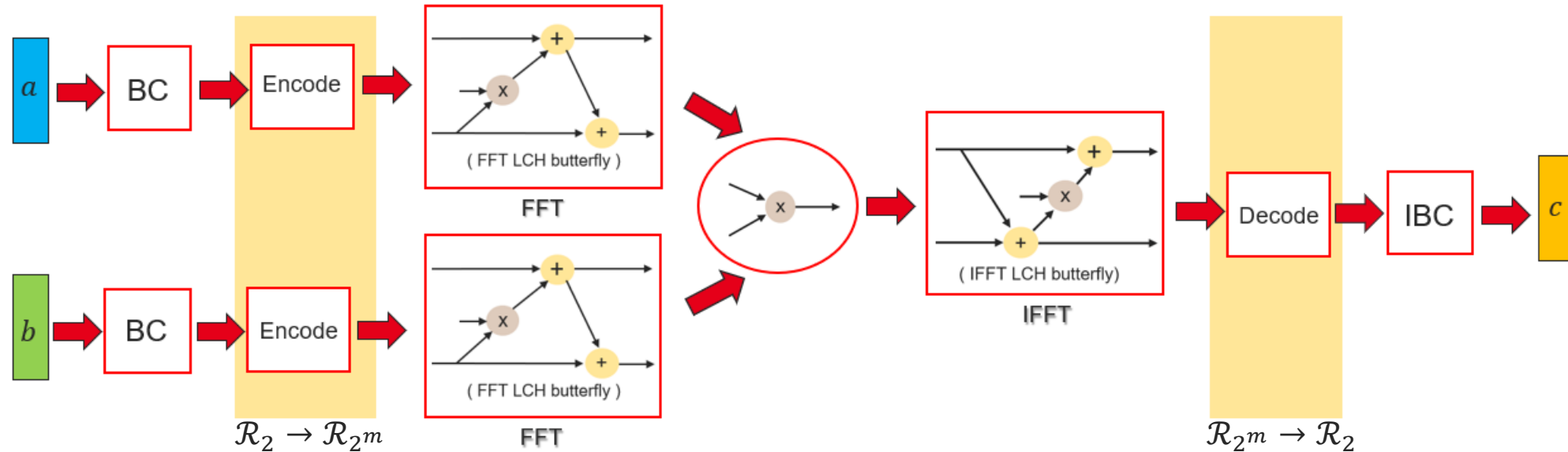
3. Frobenius Additive FFT

The alternative polynomial multiplication strategy for HQC

Frobenius Additive FFT (FAFFT)

➤ FFT-like Binary power-of-two polynomial multiplication

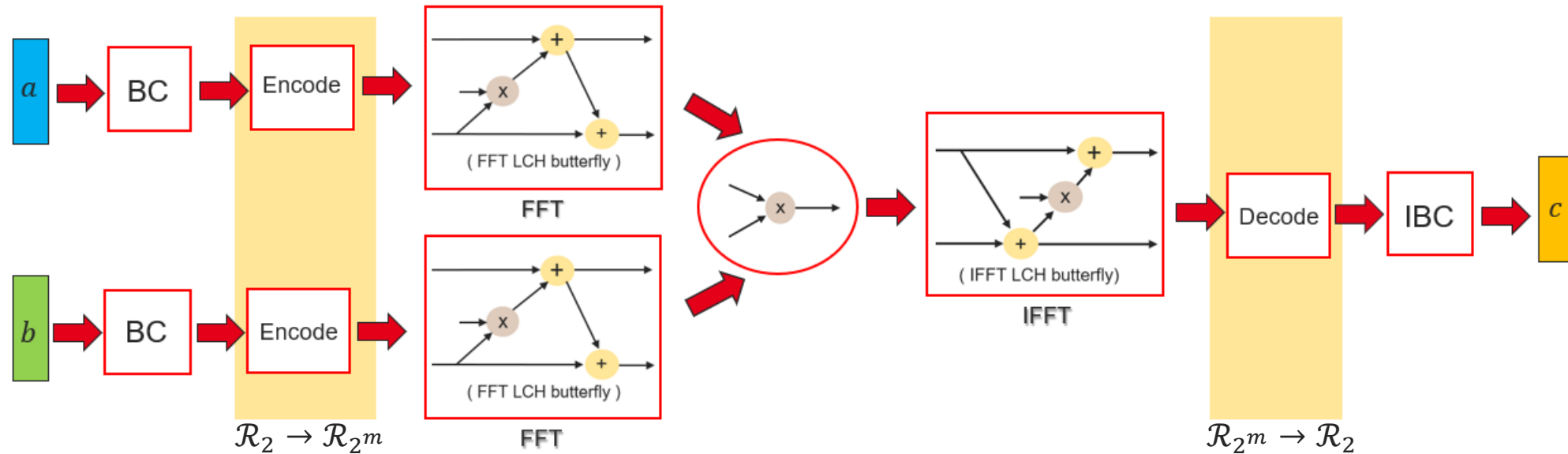
- ❑ New polynomial basis using **Cantor basis**
- ❑ Encode and Decode : **Frobenius map** reduce evaluation points



Frobenius Additive FFT (FAFFT)

➤ FFT-like Binary power-of-two polynomial multiplication

- ❑ New polynomial basis using **Cantor basis**
- ❑ Encode and Decode : **Frobenius map** reduce evaluation points



Which operation we compute in practice ?

- ❑ $\mathcal{R}_{2^m} := \mathbb{Z}_2[x] / g(x)$
- ❑ $m = 32$ and $g(x) = x^{32} + x^{22} + x^2 + x + 1$

Carryless addition

$$h = f \oplus g$$

Modular carryless multiplication

$$h = (f * g) \bmod g(x)$$

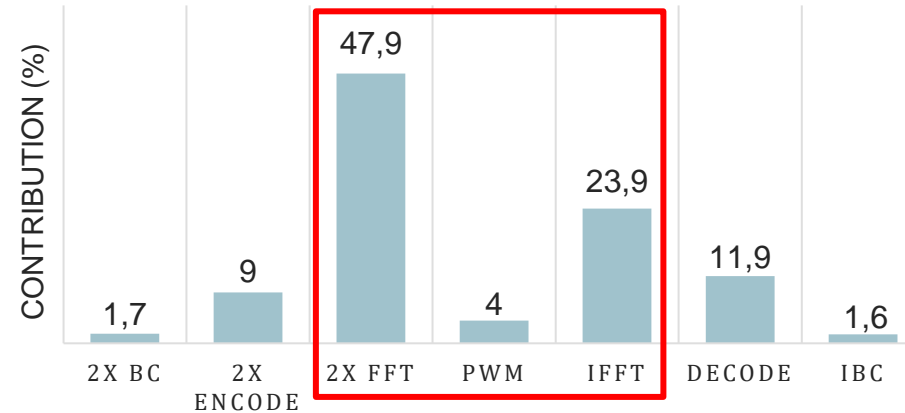
Shift-and-Add strategy

HQC using FAFFT in software

➤ FAFFT profiling

- ❑ Implementation in C, without ASM optimization
- ❑ Improved version of FAFFT in BIKE [CCK21]
- ❑ Adapt for HQC polynomial size
 - ❑ HQC-128 : 32 768 bits
 - ❑ HQC-192/256 : 65 536 bits

FAFFT PROFILING CORTEX-A9 @125 MHZ



HQC using FAFFT in software

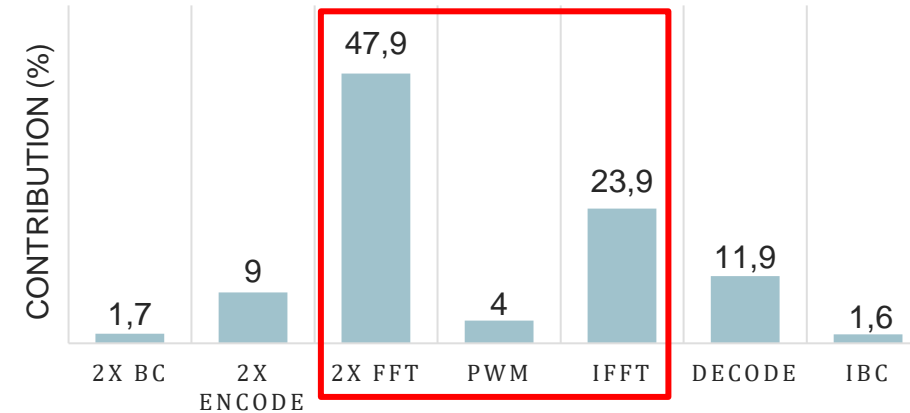
➤ FAFFT profiling

- ❑ Implementation in C, without ASM optimization
- ❑ Improved version of FAFFT in BIKE [CCK21]
- ❑ Adapt for HQC polynomial size
 - ❑ HQC-128 : 32 768 bits
 - ❑ HQC-192/256 : 65 536 bits

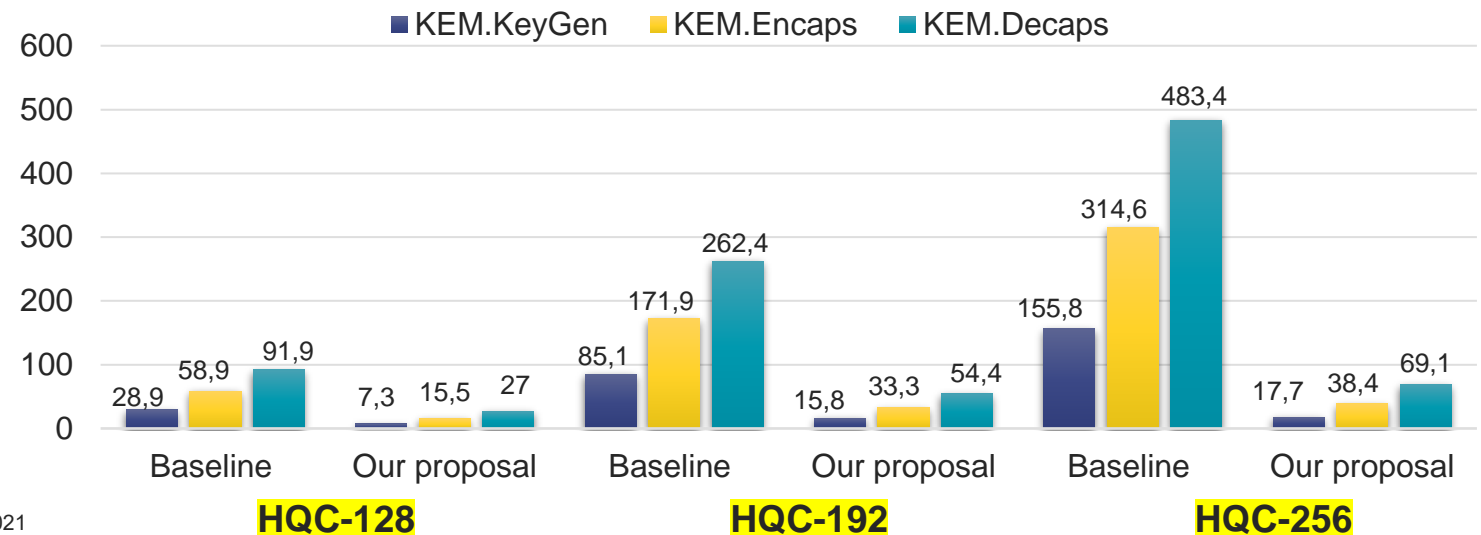
➤ HQC using FAFFT profiling

- ❑ First FAFFT integration in HQC
- ❑ Speed up x3.7 / x5 / x8 for respective HQC-X
- ❑ Baseline : 2-way Karatsuba
- ❑ Our proposal : FAFFT

FAFFT PROFILING CORTEX-A9 @125 MHZ



Benchmark HQC-X (Mcycles)





4. Proposed sharing strategy

The solution to manage different butterfly structures with different mathematical foundations

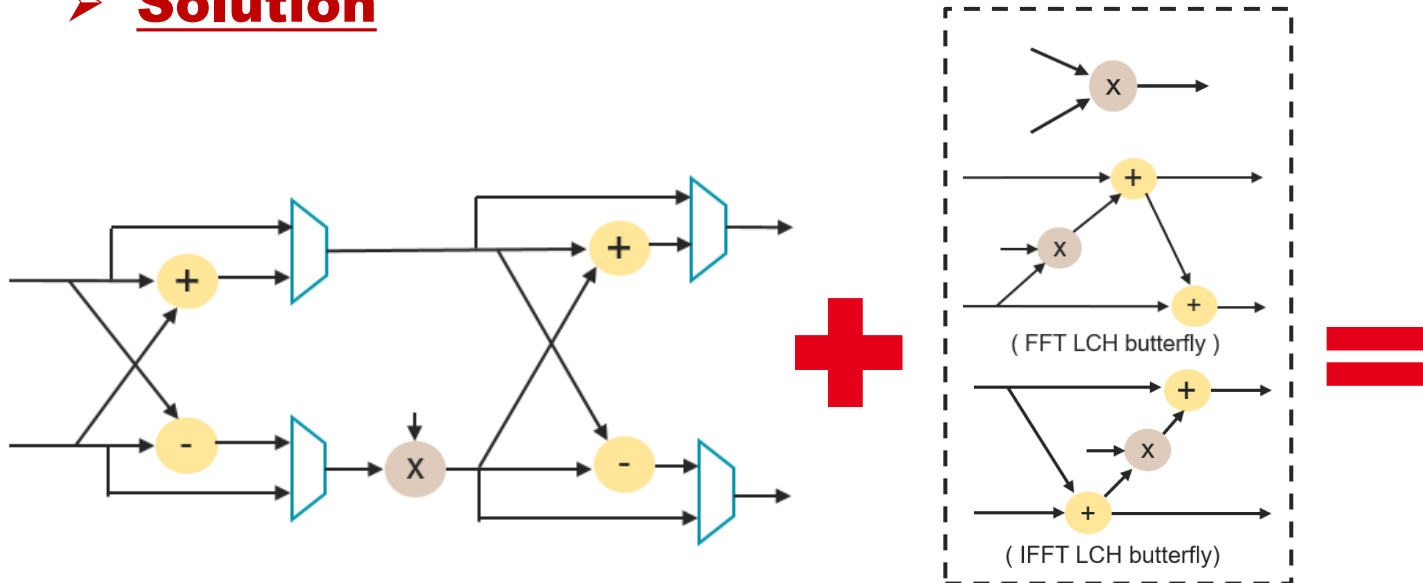
Super-Butterfly

➤ Challenges

- ❑ How to combine different butterfly structures ?
 - ❖ Different coefficient size
- ❑ How to combine different mathematical foundation ?
 - ❖ Integer vs Carryless
 - ❖ Different reduction method

	NTT in ML-KEM	FAFFT in HQC
Structure	\mathbb{Z}_{3329}	$\mathcal{R}_{2^{32}}$
Coefficient size	12-bits	32-bits
Addition	Modulo q = 3329	Carryless (XOR)
Subtraction		
Multiplication	Integer	Carryless
\perp Reduction	Barrett	Shift-and-Add

➤ Solution



Super-Butterfly

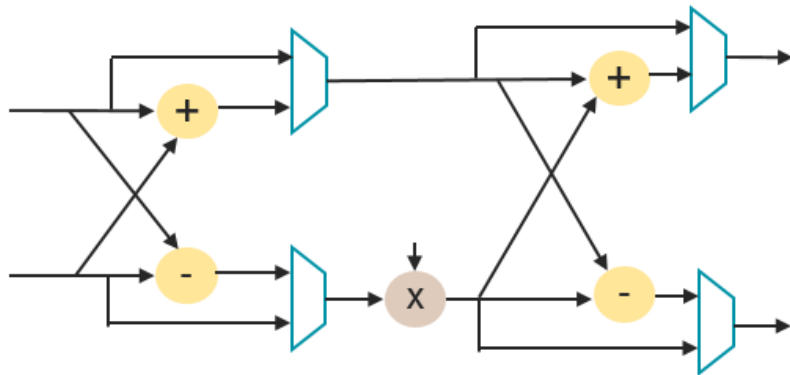
➤ Challenges

- ❑ How to combine different butterfly structures ?
 - ❖ Different coefficient size
- ❑ How to combine different mathematical foundation ?
 - ❖ Integer vs Carryless
 - ❖ Different reduction method

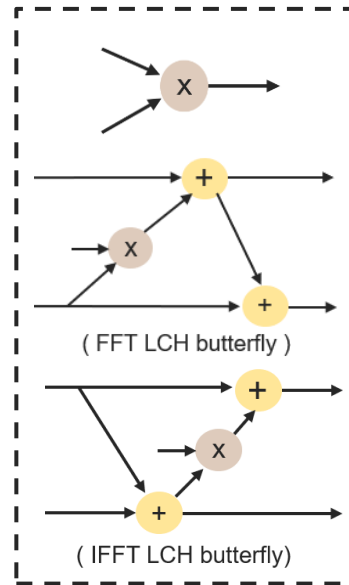
	NTT in ML-KEM	FAFFT in HQC
Structure	\mathbb{Z}_{3329}	\mathcal{R}_2^{32}
Coefficient size	12-bits	32-bits
Addition	Modulo q = 3329	Carryless (XOR)
Subtraction		
Multiplication	Integer	Carryless
\perp Reduction	Barrett	Shift-and-Add

➤ Solution

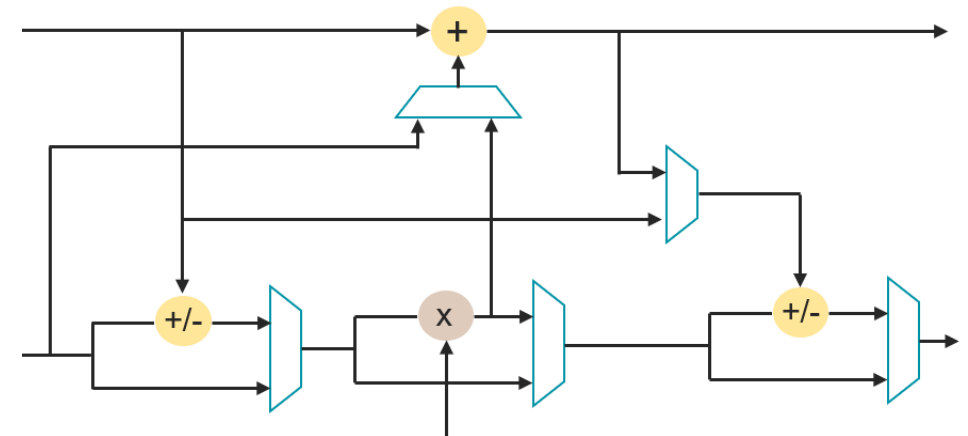
- ❑ Data parallelism = 32



2x NTT butterfly instances



1x FAFFT butterfly instances



Super-Butterfly

Agility on mathematical foundation



➤ **Agile Modular Arithmetics**

➤ **Agile Modular Multiplier**

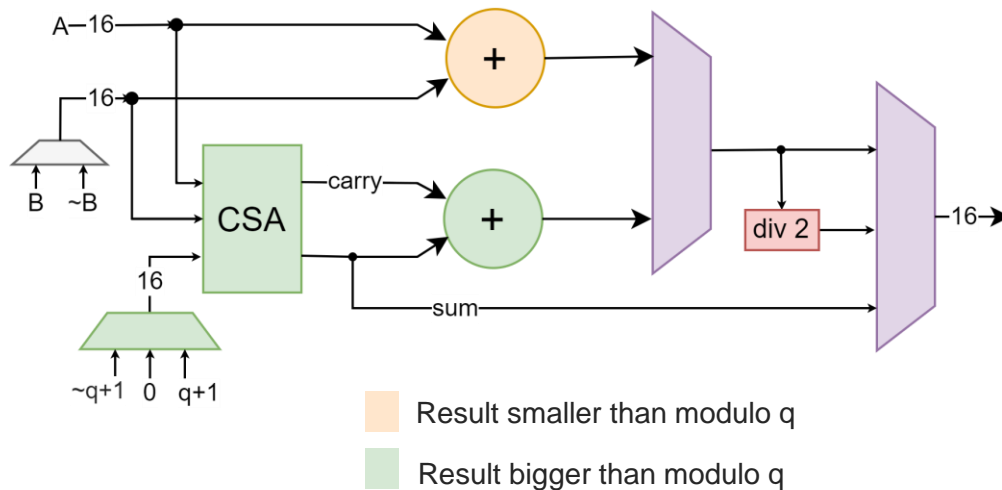
- ❑ Carryless addition \rightarrow XOR operation

- ## ❑ Modular addition and subtraction

- ❖ Two's complement to have **only additions**

$$c = (a + b) \bmod q = \begin{cases} c_t & \text{if } c_t < q, \\ c_t + (\bar{q} + 1) & \text{otherwise} \end{cases} \quad \text{where } c_t := a + b;$$

$$d = (a - b) \bmod q = \begin{cases} d_t & \text{if } d_t < q, \\ d_t + q & \text{otherwise} \end{cases} \quad \text{where } d_t := a + (\bar{b} + 1)$$



Agility on mathematical foundation

➤ Agile Modular Arithmetics

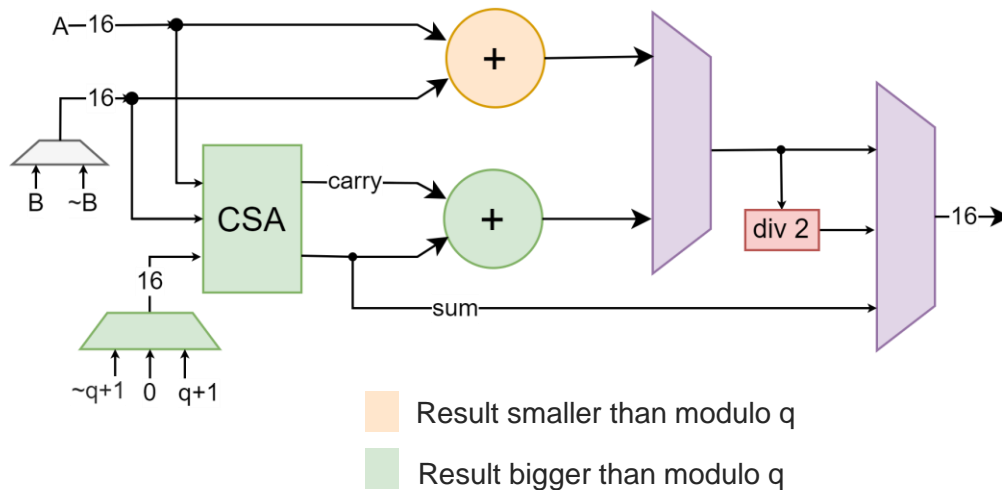
❑ Carryless addition → XOR operation

❑ Modular addition and subtraction

❖ Two's complement to have **only additions**

$$c = (a + b) \bmod q = \begin{cases} c_t & \text{if } c_t < q, \\ c_t + (\bar{q} + 1) & \text{otherwise} \end{cases} \quad \text{where } c_t := a + b;$$

$$d = (a - b) \bmod q = \begin{cases} d_t & \text{if } d_t < q, \\ d_t + q & \text{otherwise} \end{cases} \quad \text{where } d_t := a + (\bar{b} + 1)$$



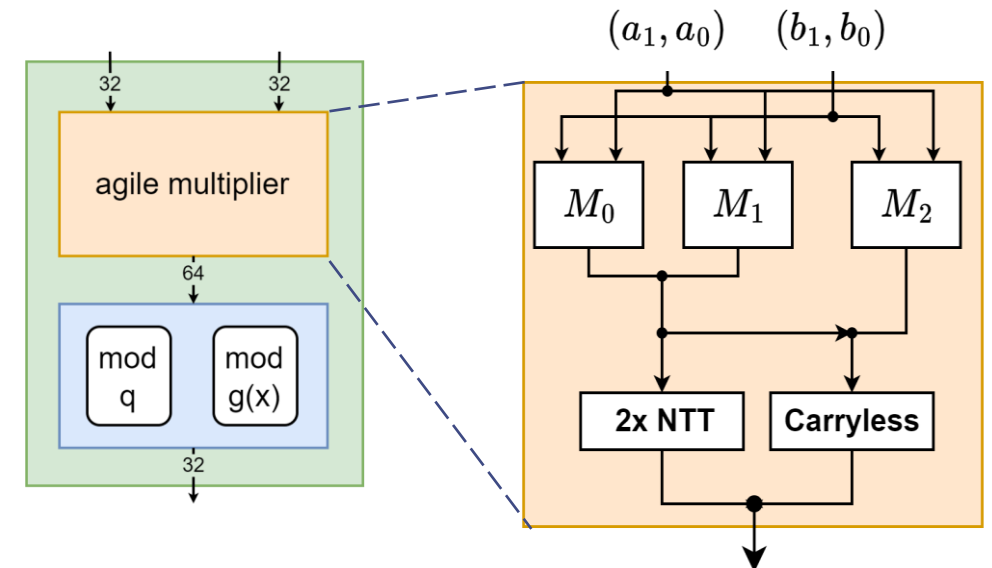
➤ Agile Modular Multiplier

❑ Hybrid 2-way Karatsuba

❑ Schoolbook-based M_x multiplier

$$\begin{aligned} a &= (a_1, a_0) \\ b &= (b_1, b_0) \end{aligned} \quad \begin{aligned} z_0 &= a_0 \times_{M_0} b_0 \\ z_1 &= a_1 \times_{M_1} b_1 \\ z_2 &= [(a_1 \oplus a_0) \times_{M_2} (b_1 \oplus b_0)] \oplus z_0 \oplus z_1 \end{aligned} \quad \text{NTT butterfly multiplication}$$

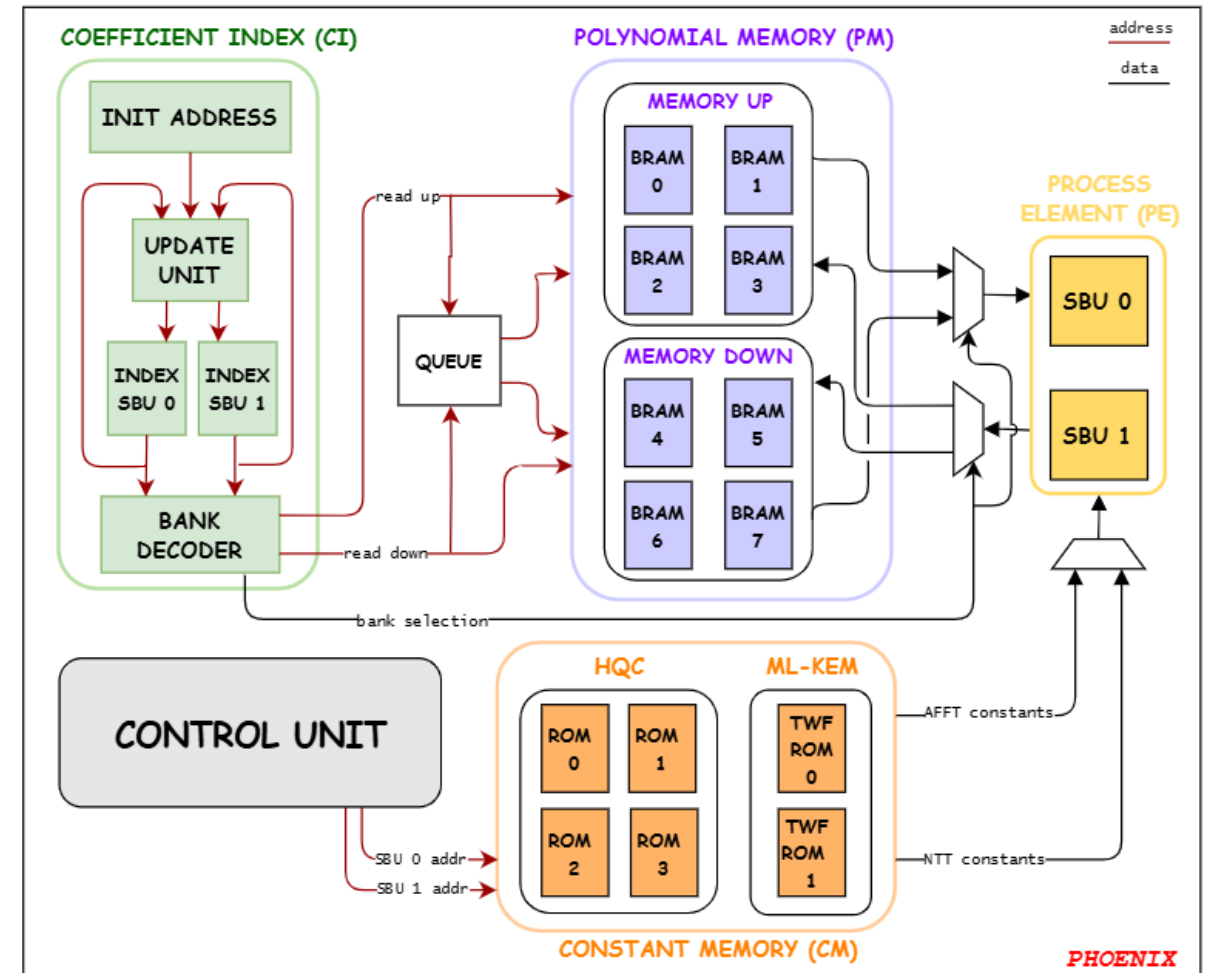
M_0, M_1 : agile multiplier (integer + carryless)
 M_2 : carryless multiplier



PHOENIX

Features

- ❑ **Loosely-coupled accelerator**
- ❑ **Process Element** : Super-Butterfly valorisation
- ❑ **Coefficient Generation** : Generate coefficient pairs to be processed
- ❑ **Polynomial Memory** : Conflict-free memory
- ❑ **Constant Memory** : Different constant values
- ❑ **Control Unit** : Orchestrate everything



PHOENIX

Features

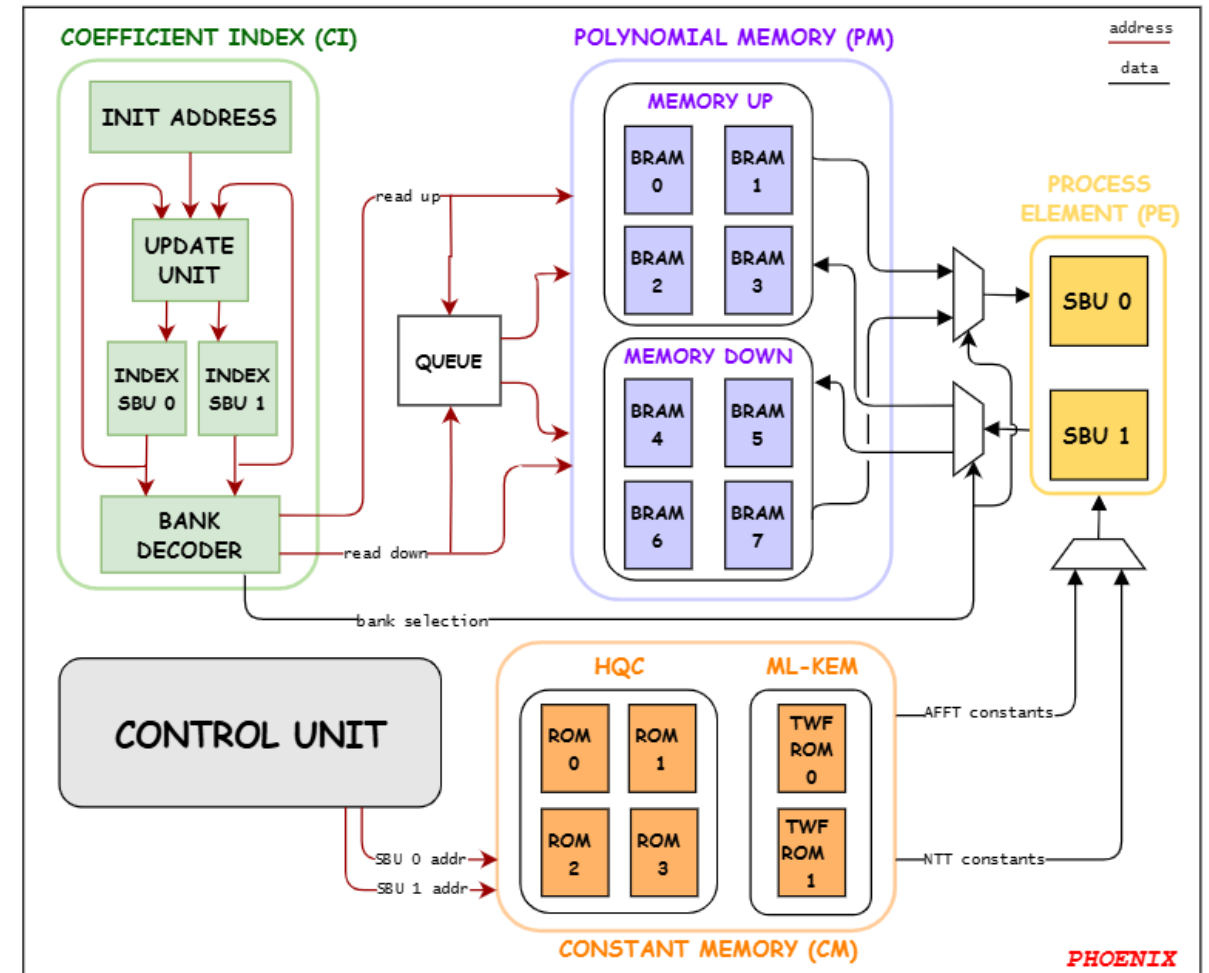
- ❑ **Loosely-coupled accelerator**
- ❑ **Process Element** : Super-Butterfly valorisation
- ❑ **Coefficient Generation** : Generate coefficient pairs to be processed
- ❑ **Polynomial Memory** : Conflict-free memory
- ❑ **Constant Memory** : Different constant values
- ❑ **Control Unit** : Orchestrate everything

Further optimization for ML-KEM

- ❑ Vector-like polynomial multiplication
- ❑ Addition of polynomial error

$$\begin{bmatrix} t_0 \\ \vdots \\ t_{k-1} \end{bmatrix} = \begin{bmatrix} a_{00} & \cdots & a_{0(k-1)} \\ \vdots & \ddots & \vdots \\ a_{(k-1)0} & \cdots & a_{(k-1)(k-1)} \end{bmatrix} \times \begin{bmatrix} s_0 \\ \vdots \\ s_{k-1} \end{bmatrix} + \begin{bmatrix} e_0 \\ \vdots \\ e_{k-1} \end{bmatrix}$$

- Increase performance
- reducing communication overhead





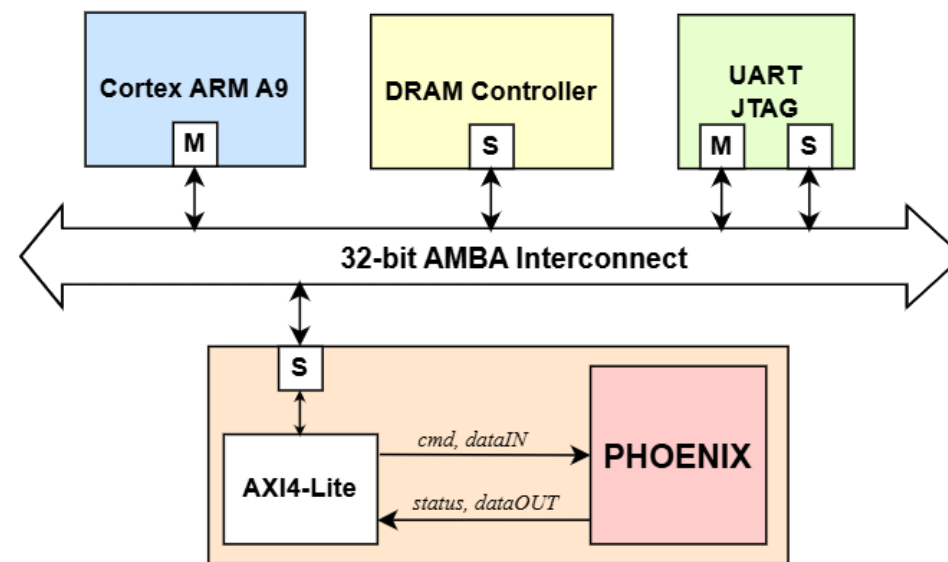
5. Results

Crypto-Agility using PHOENIX for all NIST security levels

Resource utilization on FPGA

➤ System-on-Chip integration

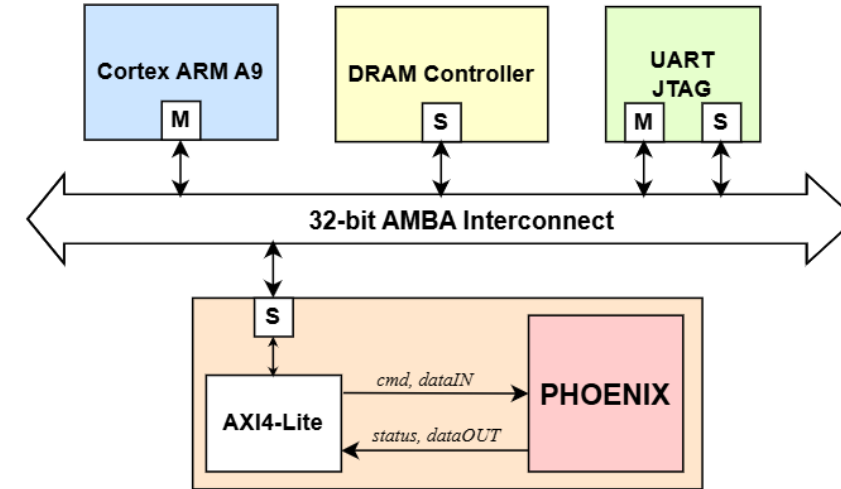
- ❑ Zybo-Z7 @125 MHz, Artix-7 FPGA
- ❑ AXI4-Lite system bus
- ❑ All NIST security levels



Resource utilization on FPGA

➤ System-on-Chip integration

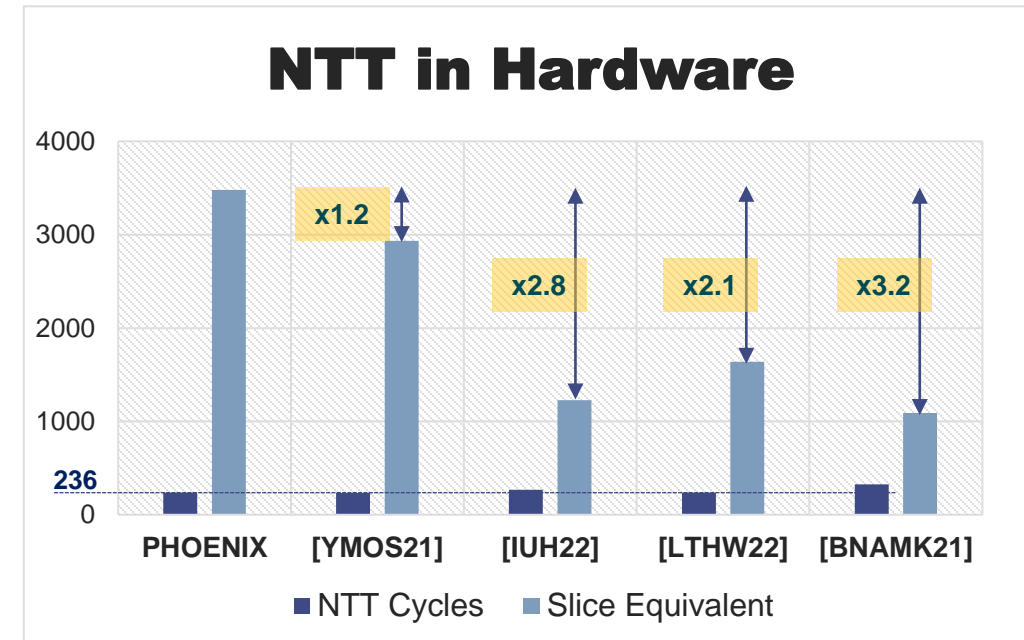
- ❑ Zybo-Z7 @125 MHz, Artix-7 FPGA
- ❑ AXI4-Lite system bus
- ❑ All NIST security levels



➤ Resource comparison

Observation

- ❑ Absence of FAFFT HW implem
- ❑ NTT cycles aligned to state-of-the-art
- ❑ High-Speed vs Lightweight design
- ❑ Agility bring resource overhead



[YMOS21] Bisheh-Niasar et al, "High-Speed NTT-based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography", 2021

[IUH22] Itabashi et al, "Efficient modular poly-nomial multiplier for NTT accelerator of Crystals-Kyber", 2022

[LTHW22] Li et al, "Reconfigurable and high-efficiency polynomial multiplication accelerator for Crystals-Kyber.", 2022

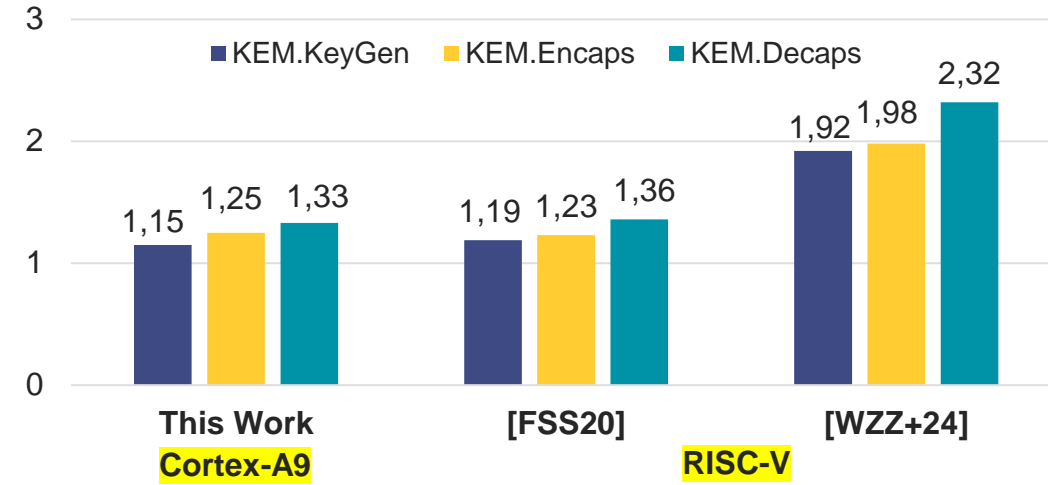
[BNAMK21] Bisheh-Niasar et al, "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography", 2021

Performance using PHOENIX

➤ ML-KEM

- ❑ **Speed-up results** ($\frac{SW}{SW+HW}$) due to different platforms
- ❑ [FSS20] uses tightly-coupled strategy for NTT
- ❑ Comparing [WZZ+24]
 - ❖ Loosely-coupled NTT
 - ❖ Assembly optimization for FIPS202
- ❑ **Promising results → system bus overhead**

Comparison for ML-KEM 768 (speed-up)

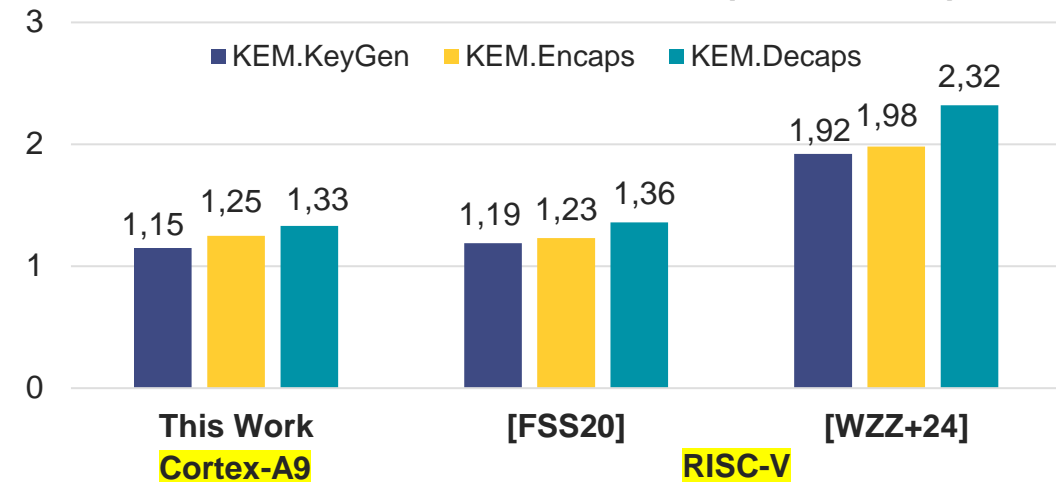


Performance using PHOENIX

➤ ML-KEM

- ❑ **Speed-up results** ($\frac{SW}{SW+HW}$) due to different platforms
- ❑ [FSS20] uses tightly-coupled strategy for NTT
- ❑ Comparing [WZZ+24]
 - ❖ Loosely-coupled NTT
 - ❖ Assembly optimization for FIPS202
- ❑ **Promising results → system bus overhead**

Comparison for ML-KEM 768 (speed-up)



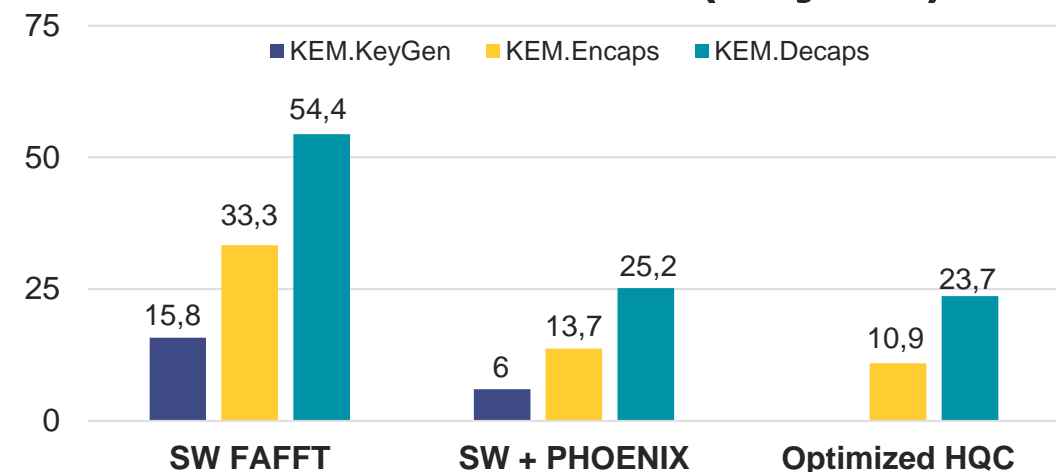
➤ HQC

- ❑ PHOENIX *accelerate* x2.3 / x2.4 / x2.0 baseline KEM FAFFT-based

Further optimization for HQC (Optimized HQC)

- ❑ Same approach conducted in ML-KEM for NTT
- ❑ **Save some FFT** but doubling public key size
- ❑ Case scenario with static key
- ❑ Speed up **x1.15** for Encaps / Decaps HQC levels

PHOENIX in HQC-192 (Mcycles)



Conclusion

Crypto-Agility

- ❑ The key to maintain PQC-based security
- ❑ Need of sharing strategy for efficient implementation

Polynomial Multiplication

- ❑ Alternative Frobenius AFFT improve current HQC bottleneck
- ❑ Polynomial multiplication FFT-based for ML-KEM and HQC
- ❑ **Challenge 1** : Combine different butterfly structures
- ❑ **Challenge 2** : Combine different mathematical foundations
- ❑ **PHOENIX valorizes Super-Butterfly design as challenges solution**

This work proves the existence of efficient sharing strategy on the new NIST KEM standards

Ongoing Improvements

- ❑ $\text{PHOENIX} < \text{NTT} + \text{FAFFT} (?)$
- ❑ ML-KEM result suffers due to system bus overhead

Resource utilization on FPGA

➤ Resource comparison with Hardware NTT

	LUT	FF	BRAM	DSP	SEC	Time		
						Cycles	MHz	μs
This work	5076	4083	8.5	0	3479	236	125	1.88
[YMÖS21]	2543	792	9	4	2935	232	182	1.27
[IUH22]	904	811	2.5	4	1227	268	216	1.24
[LTHW22]	1170	1164	2	4	1638	235	303	0.78
[BNAMK21]	801	717	2	4	1090	324	222	1.46

$$SEC = \frac{LUT}{4} + \frac{FF}{8} + BRAM \times 200 + DSP \times 100$$

FAFFT Background

Another possible way, found in the state-of-the-art, to perform polynomial multiplication in HQC is the AFFT, which is an FFT-based variant adapted to be used in the context of binary field.

Before explaining this technique we have to introduce few mathematical concepts:

- **Cantor basis** : They are a set of (β_i) satisfies $\beta_0 = 1, \beta_i^2 + \beta_i = \beta_{i-1}$ for $i > 0$.
In practice, it is a new polynomial basis for constructing a finite field as well as an FFT over the field
- **Vanishing polynomials** : Given a basis $(\beta_i)_{i=0}^{m-1}$ in the base field, its sequence of subspaces is $V_i := \text{span}\{\beta_0, \beta_1, \dots, \beta_{i-1}\}$. Its subspace vanishing polynomials (s_i) are $s_i(x) := \prod_{a \in V_i} (x - a)$
Properties :
 - (linearity) $s_i(x)$ contains only monomials in the form x^{2^k}
 - (minimal two terms) $s_i(x) = x^{2^i} + x$ iff i is a power of 2
 - (recursivity) $s_i(x) = s_{i-1}^2(x) + s_{i-1}(x)$

Resources utilization of PHOENIX



Module	LUT	FF	BRAM	DSP
Control Unit	826	131	0	0
Datapath	4250	3952	8.5	0
Coeff memory address	238	128	0	0
Write address	30	612	0	0
Polynomial Memories	0	0	8	0
Constant Memories	37	98	0.5	0
Processing element	3319	3114	0	0
1× SUPERBUTTERFLY	1432	1380	0	0
COMP1	160	0	0	0
COMP2	176	0	0	0
COMP3	948	548	0	0
COMP4	148	0	0	0
Total = Control Unit + Datapath	5076	4083	8.5	0

Performance using PHOENIX



ML-KEM

		Device	KeyGen	Encaps	Decaps
ML-KEM 512	[FSS20]	PULPino	939,932 $\times 1.21$	1,223,887 $\times 1.26$	1,051,003 $\times 1.45$
	This Work	Zybo-Z7-20	564,189 $\times 1.14$	570,438 $\times 1.30$	639,403 $\times 1.38$
	[WZZ ⁺ 24]	PolarFire	326,983 $\times 1.91$	415,496 $\times 2.01$	394,661 $\times 2.42$
ML-KEM 768	[FSS20]	PULPino	1,768,400 $\times 1.19$	2,138,810 $\times 1.23$	1,889,930 $\times 1.36$
	This Work	Zybo-Z7-20	916,828 $\times 1.15$	954,992 $\times 1.25$	1,048,867 $\times 1.33$
	[WZZ ⁺ 24]	PolarFire	536,213 $\times 1.92$	671,082 $\times 1.98$	639,024 $\times 2.32$
ML-KEM 1024	[FSS20]	PULPino	2,856,302 $\times 1.18$	3,312,957 $\times 1.21$	2,989,896 $\times 1.32$
	This Work	Zybo-Z7-20	1,450,164 $\times 1.14$	1,492,047 $\times 1.22$	1,614,779 $\times 1.28$
	[WZZ ⁺ 24]	PolarFire	844,008 $\times 1.89$	1,015,251 $\times 1.91$	972,598 $\times 2.18$

HQC

		Ref.	Standard HQC					Optimized HQC			
			FAFFT	PHOENIX				FAFFT	PHOENIX		
HQC-128	KeyGen	28.9	7.3	3.9×	2.8	10.3×	-	-	-	-	-
	Encaps	58.7	15.5	3.8×	6.5	9.0×	9.6	6.1×	5.1	11.5×	
	Decaps	91.9	27.0	3.4×	13.6	6.8×	20.6	4.5×	13.1	7.0×	
HQC-192	KeyGen	85.1	15.8	5.4×	6.0	14.2×	-	-	-	-	-
	Encaps	171.9	33.3	5.2×	13.7	12.6×	20.7	8.3×	10.9	15.8×	
	Decaps	262.4	54.4	4.8×	25.2	10.4×	39.9	6.6×	23.7	11.1×	
HQC-256	KeyGen	155.8	17.7	8.8×	7.9	19.7×	-	-	-	-	-
	Encaps	314.6	38.4	8.2×	18.8	16.7×	25.5	12.3×	15.6	20.2×	
	Decaps	483.4	69.1	7.0×	39.8	12.1×	52.4	9.2×	36.1	13.4×	

HQC : Standard vs Optimized

