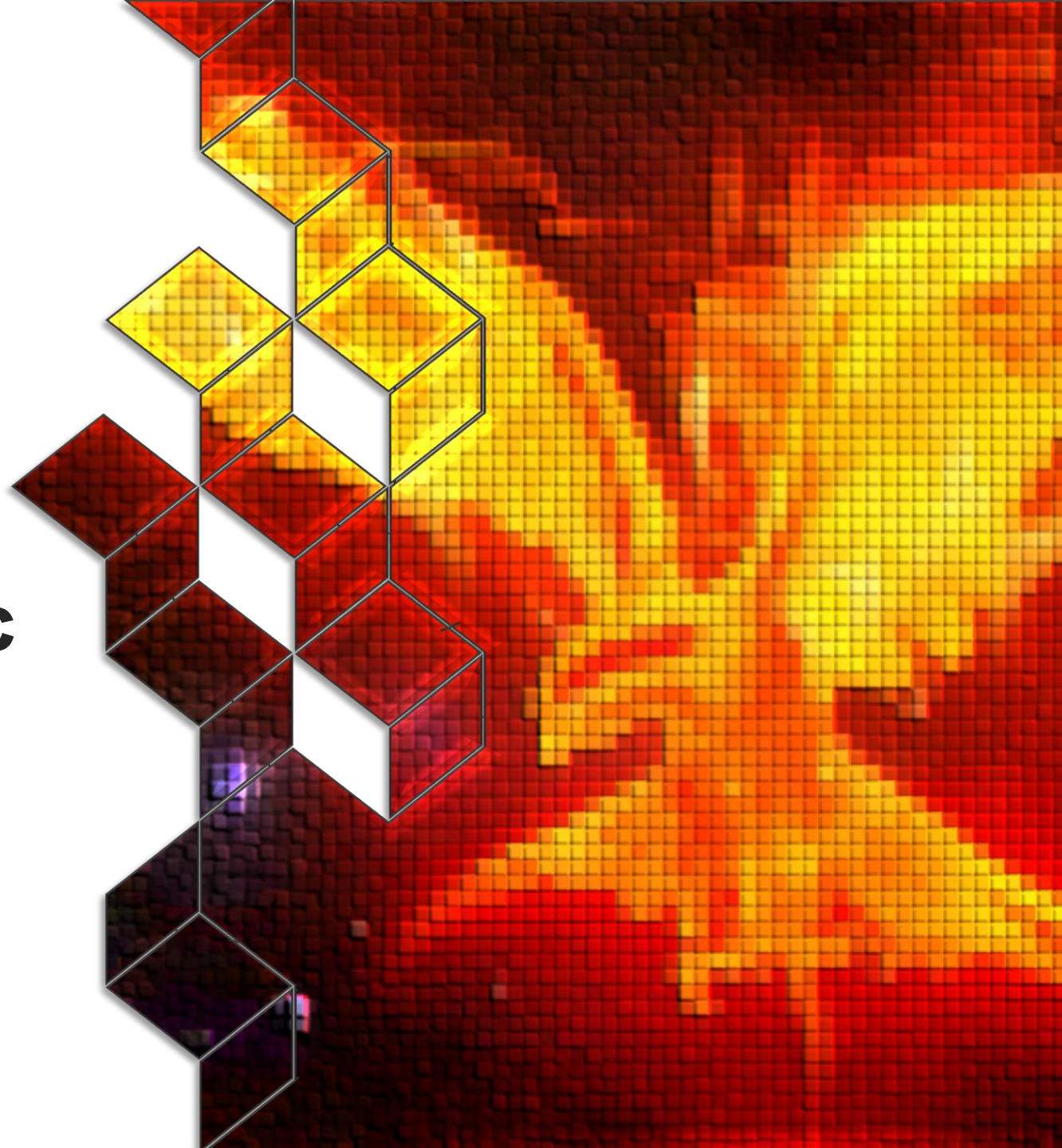
PHOENIX

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 **:**
Guénaël Renault, Benjamin Smith



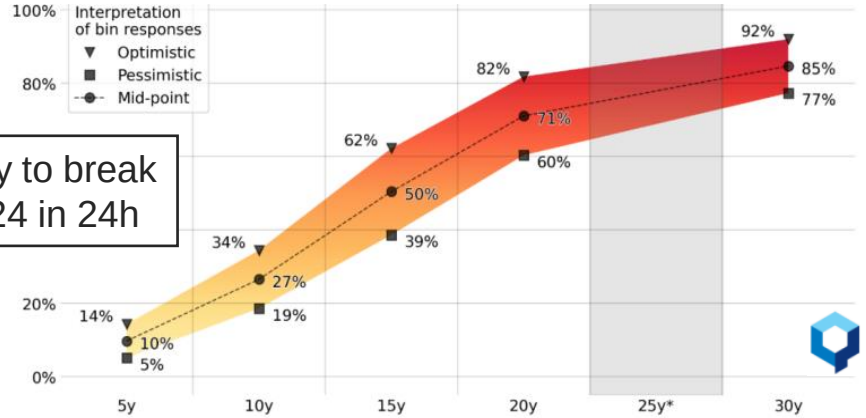
Séminaire SecuElec, Rennes | 25 April 2025

Introduction

Quantum-threat

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

Probability to break RSA-1024 in 24h

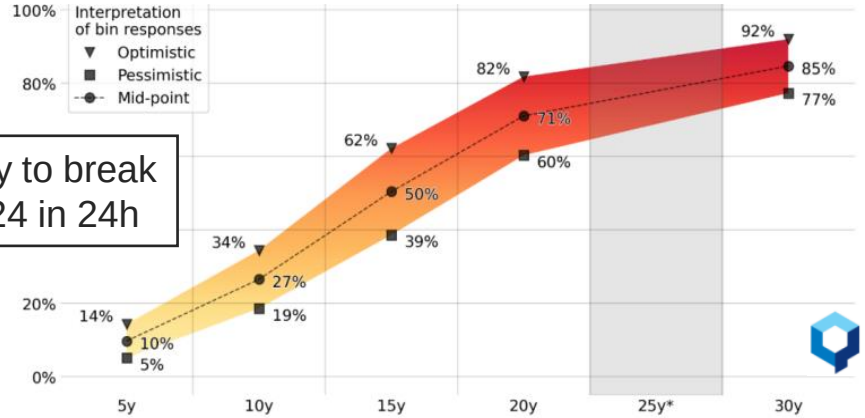


Introduction

Quantum-threat

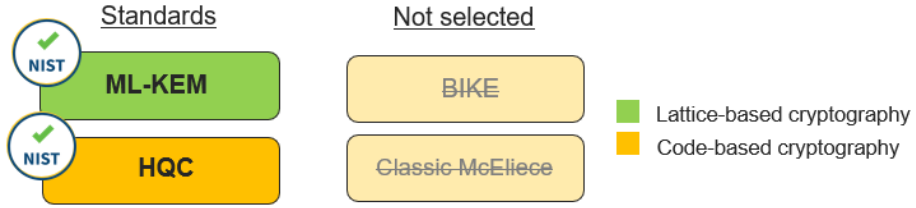
- Shor algorithm can easily solve hard problem
  - Discrete logarithm (Elliptic Curve Cryptography)
  - Integer factorization (RSA)

Probability to break RSA-1024 in 24h



Post-Quantum Cryptography

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

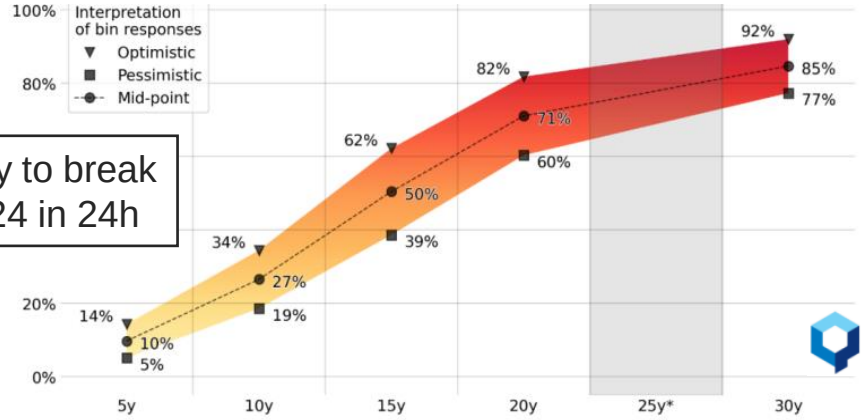


Introduction

Quantum-threat

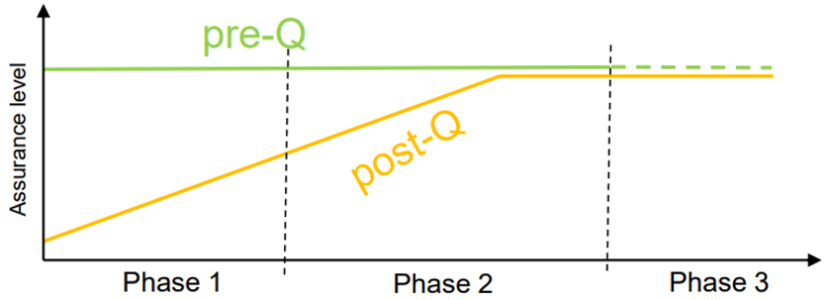
- Shor algorithm can easily solve hard problem
  - Discrete logarithm (Elliptic Curve Cryptography)
  - Integer factorization (RSA)

Probability to break RSA-1024 in 24h



Quantum-safe transition

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



Post-Quantum Cryptography

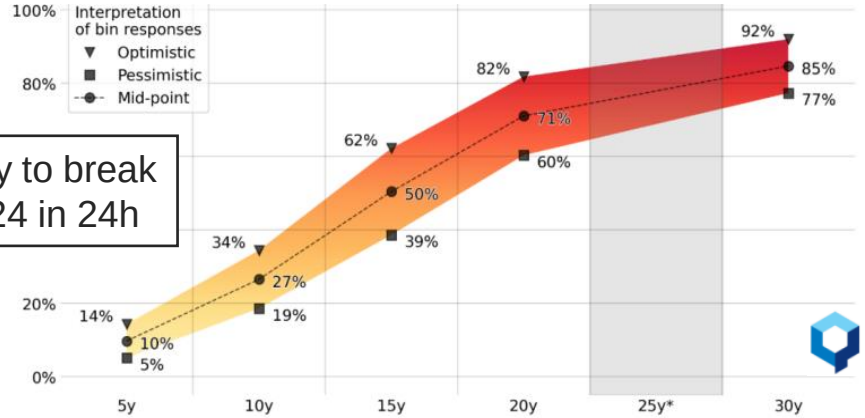
- New matematical hard problem
  - *Lattice-based* and *Code-based* cryptography
  - ...
- **Key Encapsulation Mechanism (KEM)**
- Digital Signature (DS)
- End of PQC NIST Process (2017 → 2025)

Introduction

Quantum-threat

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

Probability to break RSA-1024 in 24h

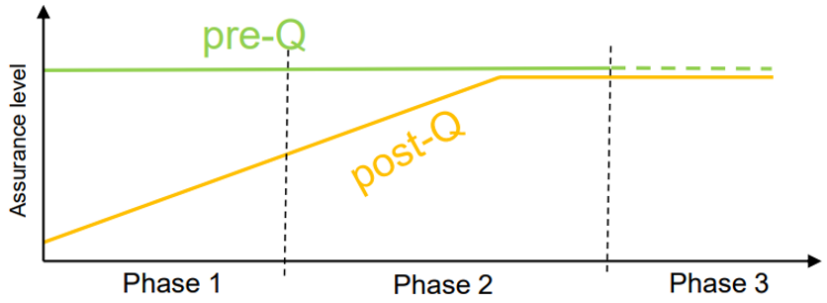


Post-Quantum Cryptography

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

Quantum-safe transition

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



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

Lattice-Code Agility State of the Art

[ZZO+24] Zhu et al, "16.2 A 28nm 69.4kOPS 4.4µJ/Op Versatile Post-Quantum Crypto-Processor Across Multiple Mathematical Problems", 2024
[ZZL+22] Zhu et al, "A 28nm 48KOPS 3.4µJ/Op Agile Crypto-Processor for Post-Quantum Cryptography on Multi-Mathematical Problems", 2022

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

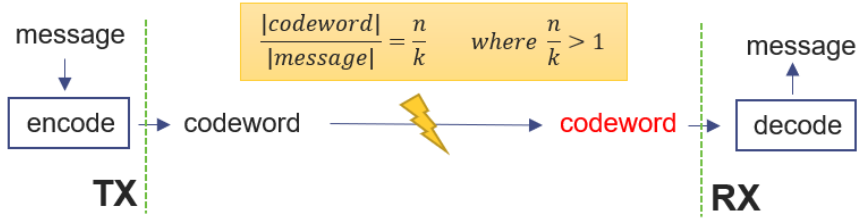
ML-KEM : *Module-Learning with Error,* lattice problem

public $\boldsymbol{A}$, uniform — secret $\boldsymbol{s}$, $\boldsymbol{e}$ 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 $\mathbf{t} = \boldsymbol{As} + \boldsymbol{e}$ and $\boldsymbol{A}$, it is difficult to find $\boldsymbol{s}$

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

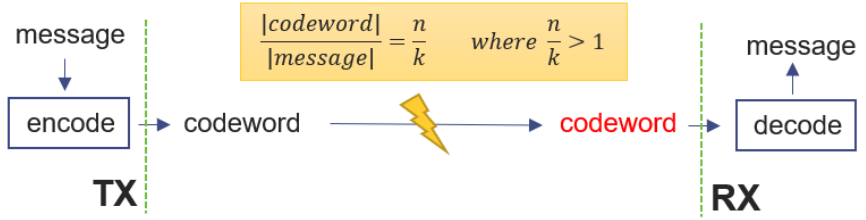
ML-KEM : *Module-Learning with Error,* lattice problem

public $\boldsymbol{A}$, uniform — secret $\boldsymbol{s}$, $\boldsymbol{e}$ 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 $\mathbf{t} = \boldsymbol{As} + \boldsymbol{e}$ and $\boldsymbol{A}$, it is difficult to find $\boldsymbol{s}$

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

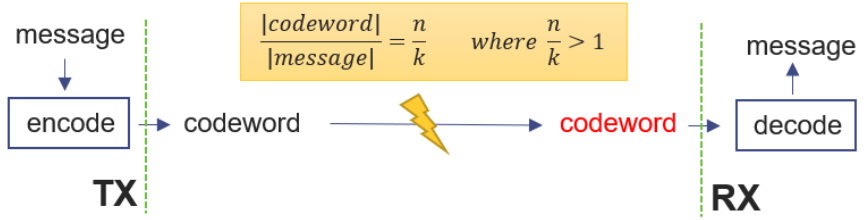
ML-KEM : *Module-Learning with Error,* lattice problem

public $\boldsymbol{A}$, uniform — secret $\boldsymbol{s}$, $\boldsymbol{e}$ 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 $\mathbf{t} = \boldsymbol{As} + \boldsymbol{e}$ and $\boldsymbol{A}$, it is difficult to find $\boldsymbol{s}$

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

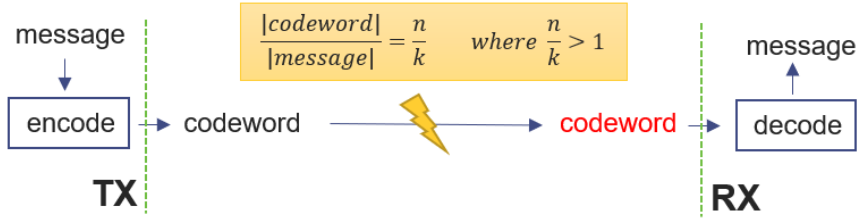
ML-KEM : *Module-Learning with Error,* lattice problem

public $\boldsymbol{A}$, uniform — secret $\boldsymbol{s}$, $\boldsymbol{e}$ 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 $\mathbf{t} = \boldsymbol{As} + \boldsymbol{e}$ and $\boldsymbol{A}$, it is difficult to find $\boldsymbol{s}$

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

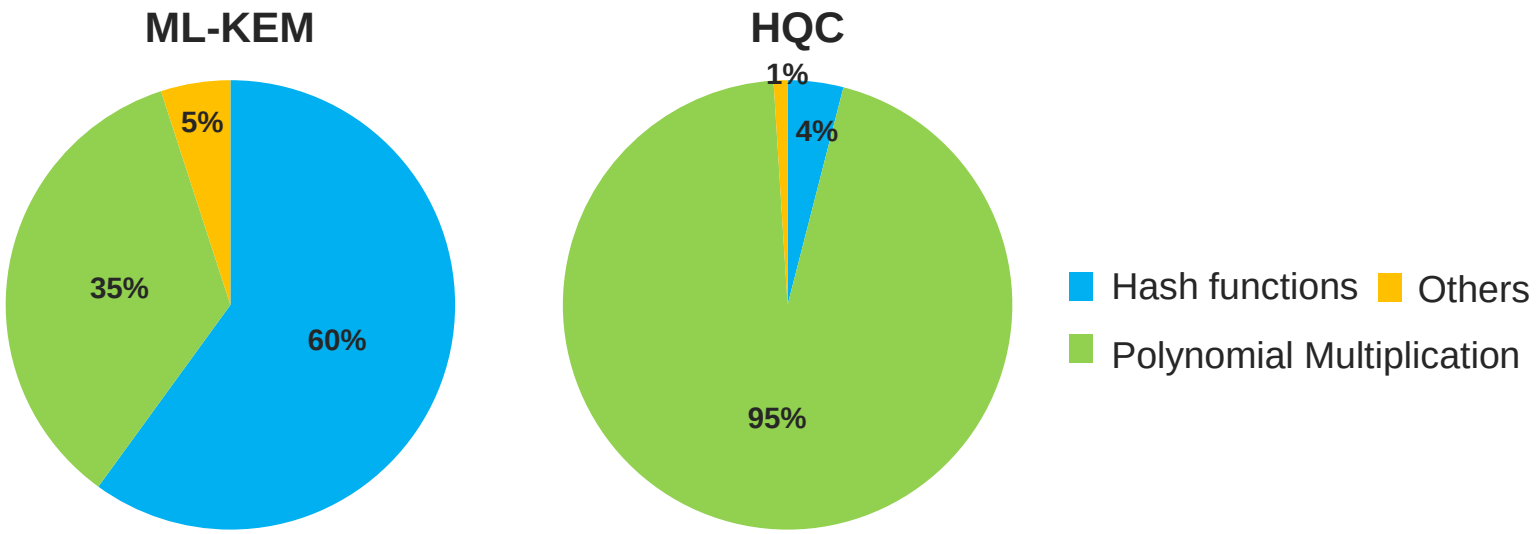
Instance	n_1	n_2	n	w	$w_r = w_e$	security
hqc-128	46	384	17,669	66	75	128
hqc-192	56	640	35,851	100	114	192
hqc-256	90	640	57,637	131	149	256

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

How to implement Crypto-Agility efficiently ?

Sharing strategy

➢ Software profiling



KEM Operations Profiling

➢ Sharing strategy

Purpose : Identify common / similar operations

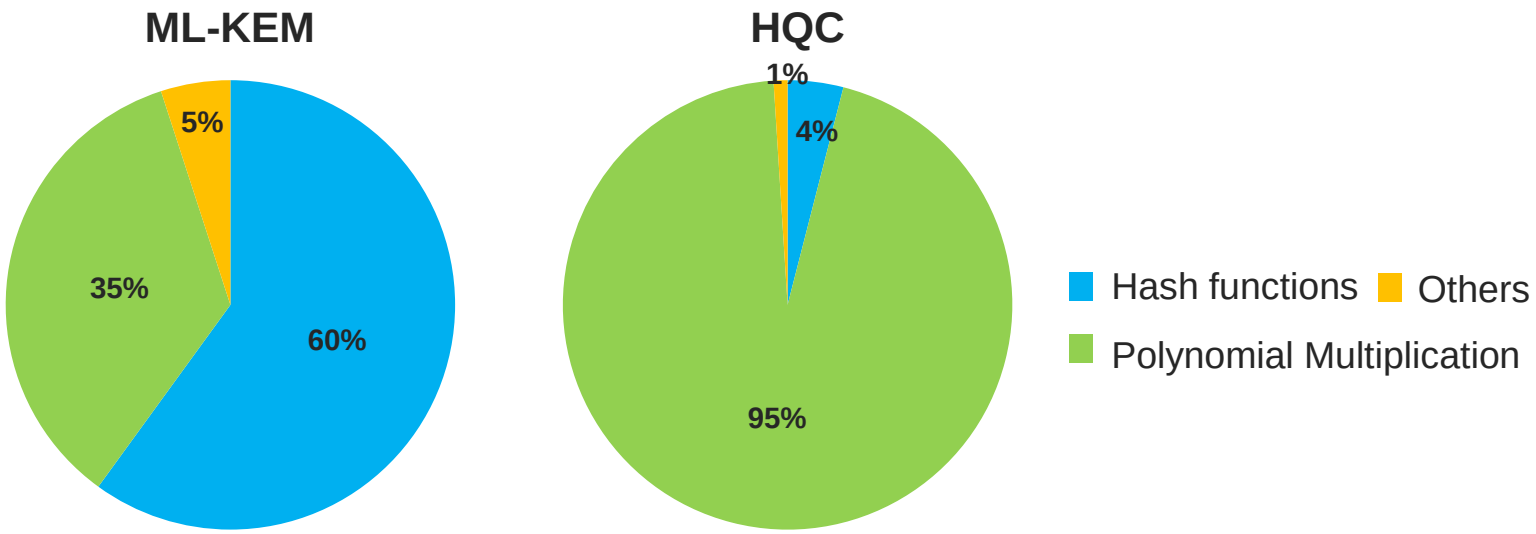
Possible sharing strategies :

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

approach

Sharing strategy

➢ Software profiling



KEM Operations Profiling

➢ Sharing strategy

Purpose : Identify common / similar operations

Possible sharing strategies :

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

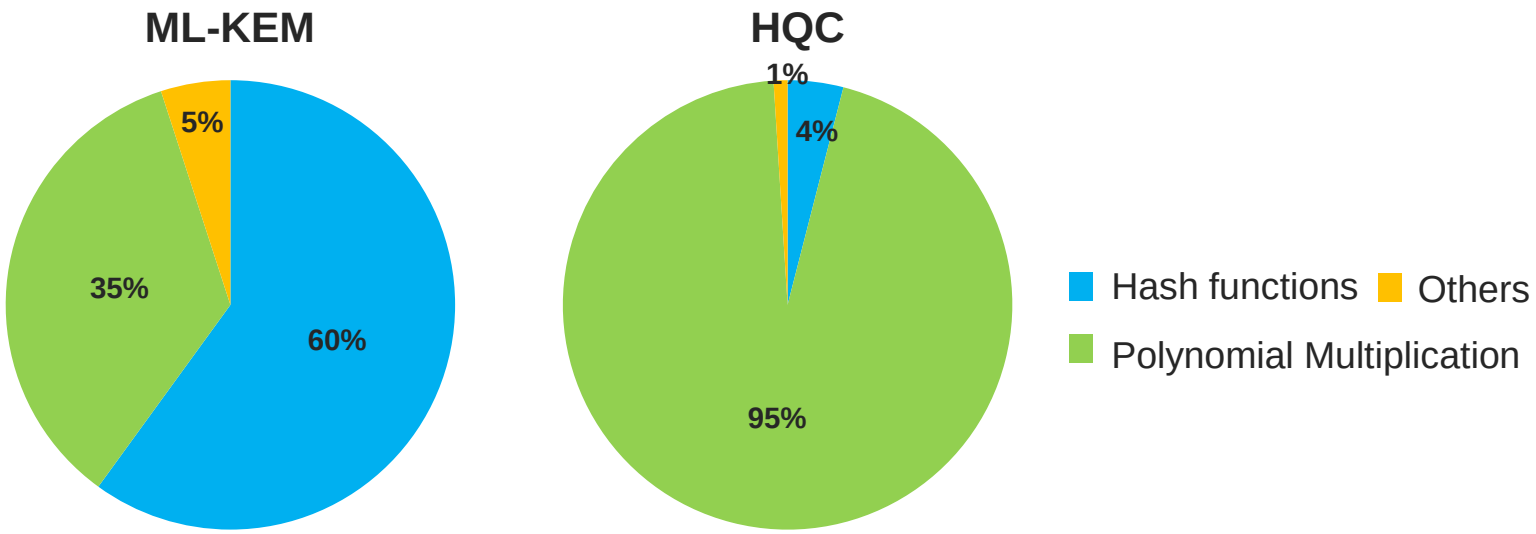
approach

Which multiplication strategy to mutualize

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

Sharing strategy

➢ Software profiling



KEM Operations Profiling

➢ Sharing strategy

Purpose : Identify common / similar operations

Possible sharing strategies :

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

approach

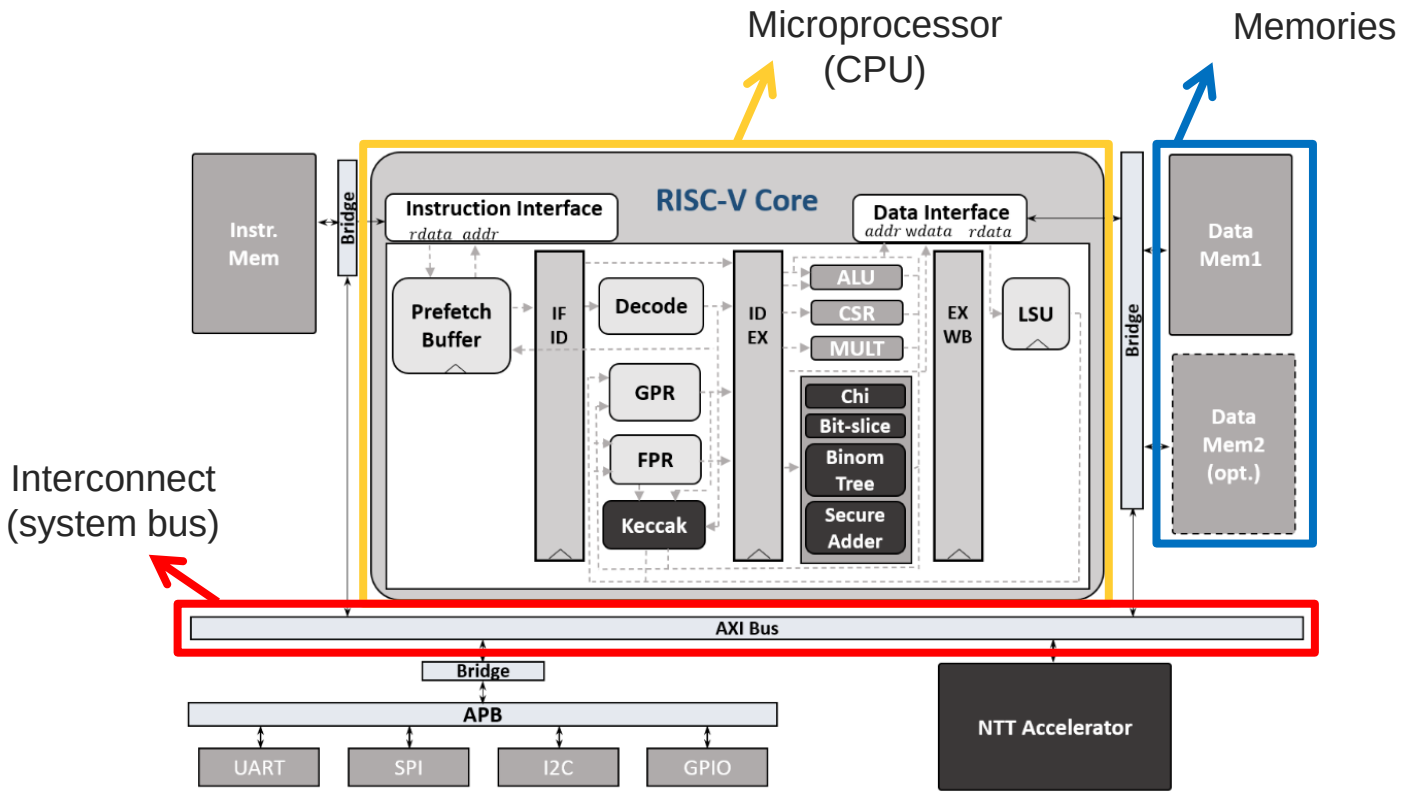
Which multiplication strategy to mutualize

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

What about hardware Crypto-Agility ?

System on Chip and hardware agility

➢ System-on-Chip



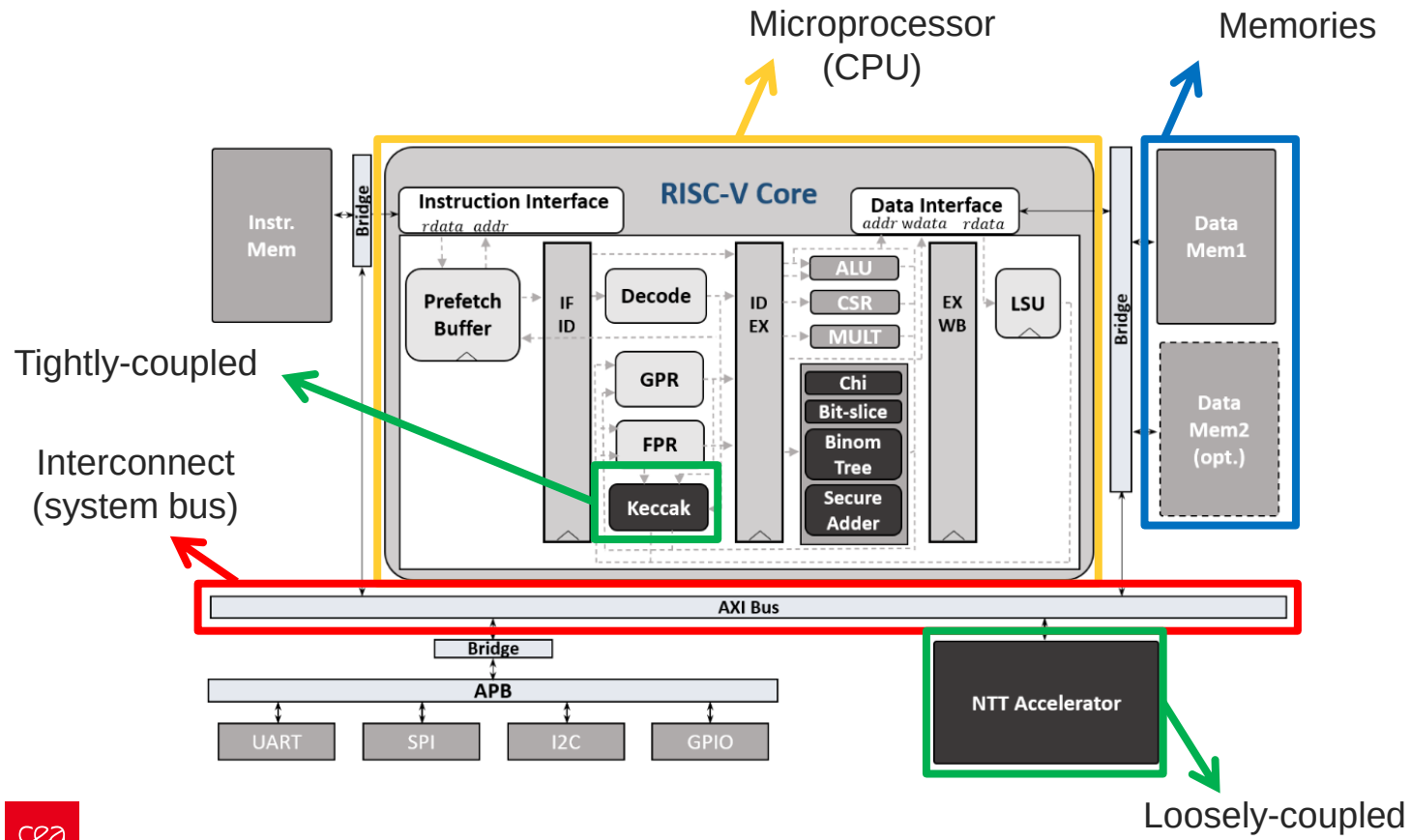
[FBR+22] Tim Fritzmann et al, *"Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography", 2022*

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

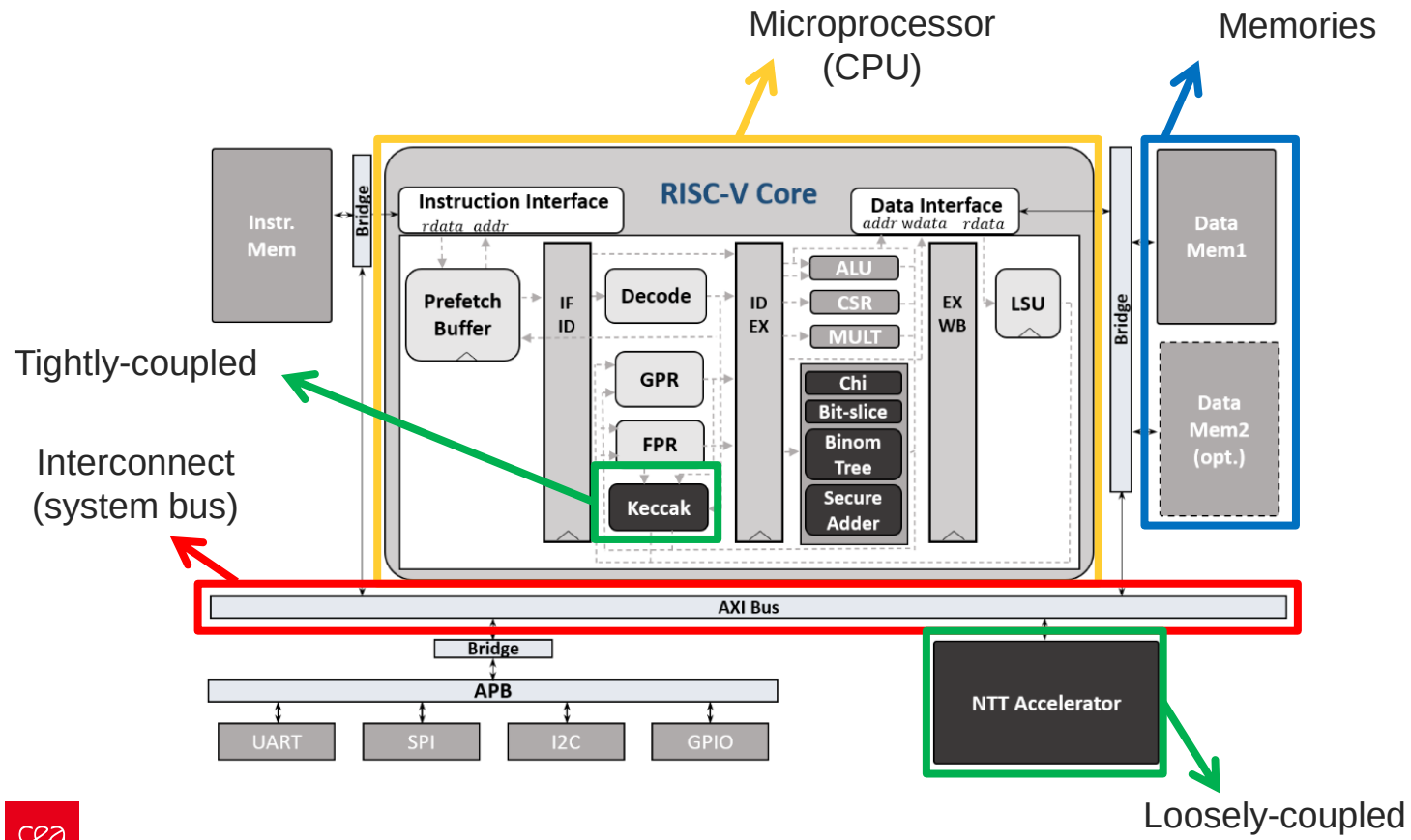


[FBR+22] Tim Fritzmann et al, "Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography", 2022

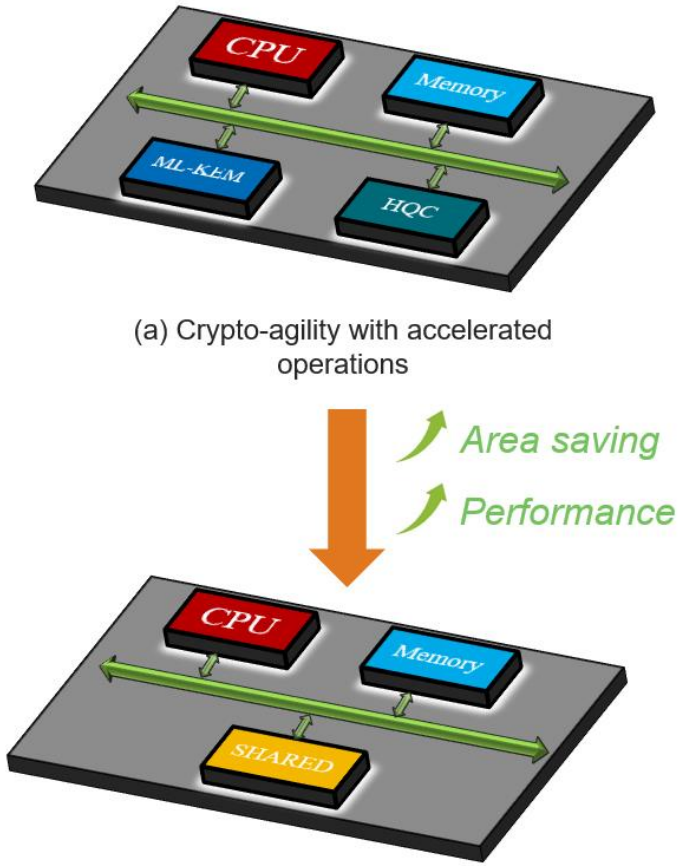
System on Chip and hardware agility

➢ System-on-Chip

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



➢ Hardware resource-sharing



[FBR+22] Tim Fritzmann et al, "Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography", 2022

2. Number-Theoretic Transform

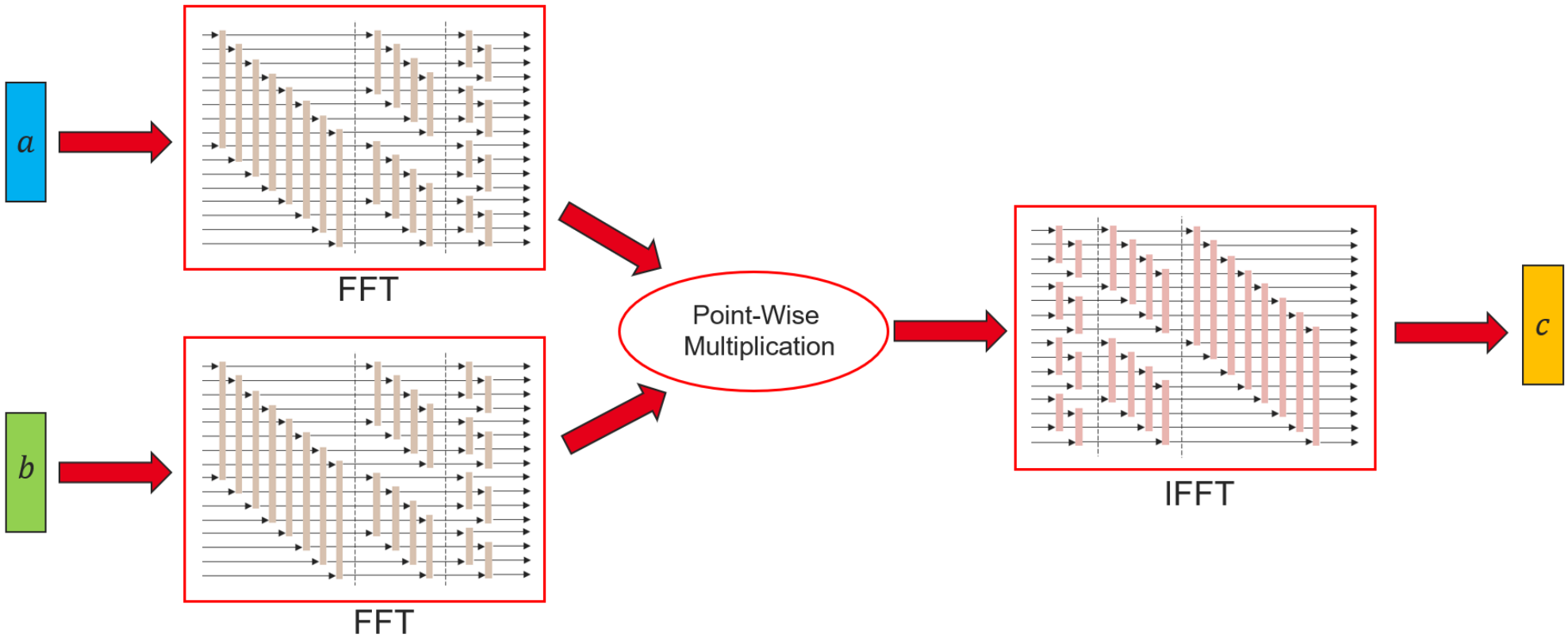
The polynomial multiplication strategy for ML-KEM

FFT-based polynomial multiplication

How to perform multiplication

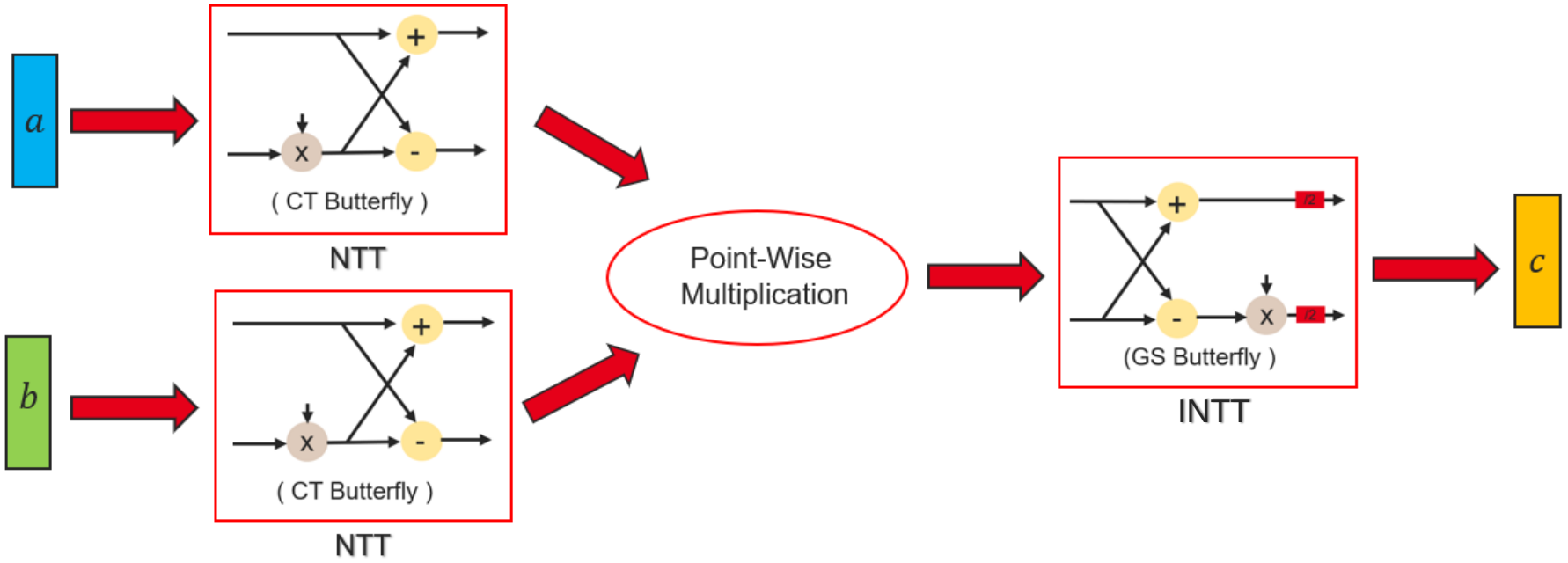
- ❑ Evaluating input polynomial in a different domain using an evaluation points set → **FFT step**
- ❑ Perform multiplication between evaluated points → **PWM step**
- ❑ Interpolate back to get final result → **Inverse FFT (IFFT) step**

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



Polynomial multiplication in ML-KEM

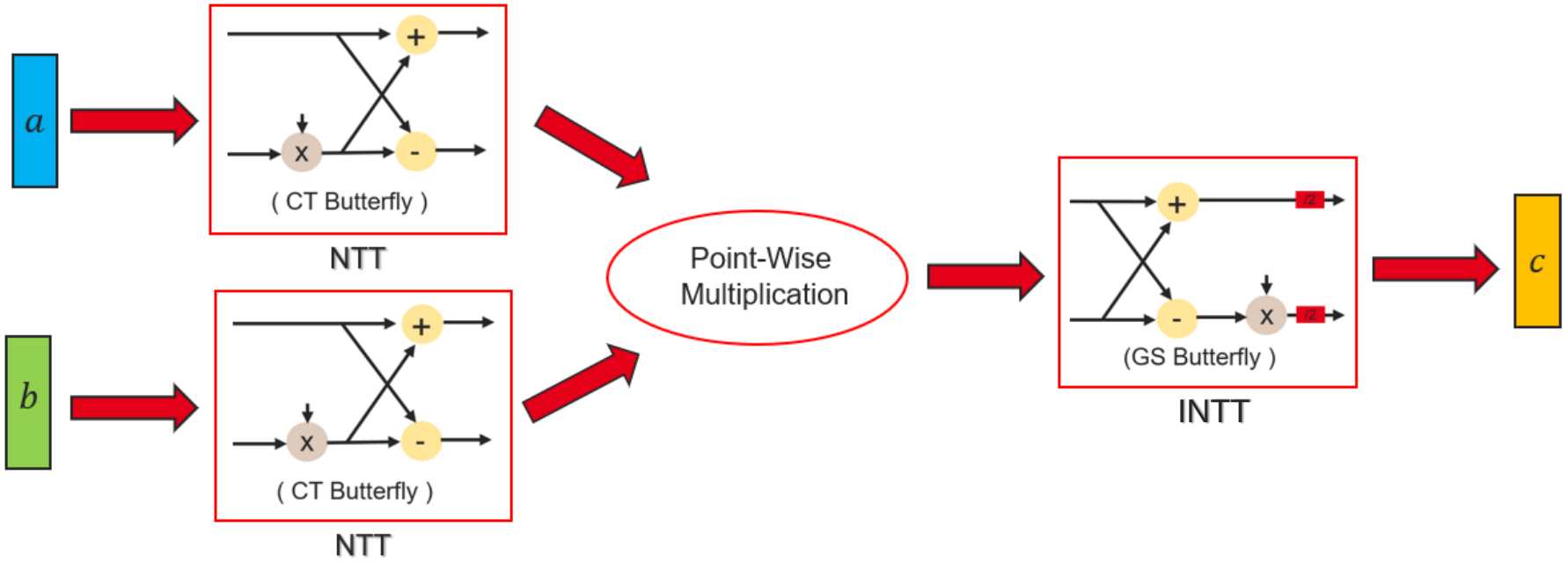
- **Number-Theoretic-Transform :** FFT-like for polynomials in $R_q = \frac{\mathbb{Z}_q}{X^n+1}$, where $q = 3329$ and $\mathrm{n} = 256$.



[XL21] Y. Xing and S. Li, *"A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA", 2021*

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

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

Modular subtraction

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

Modular integer multiplication

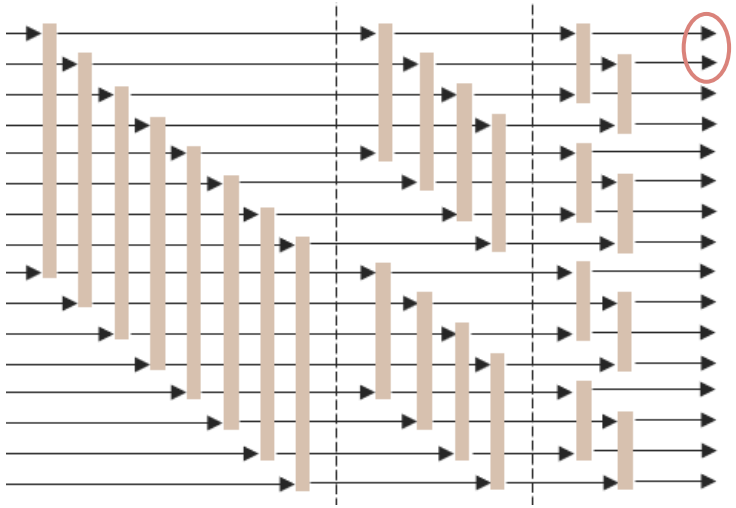
$$h = (f * g)\ mod\ q$$

Barrett or Montgomery reduction

[XL21] Y. Xing and S. Li, *"A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA", 2021*

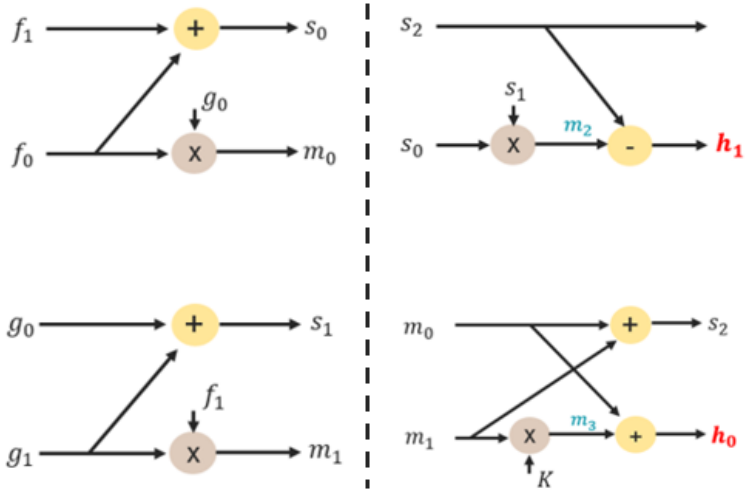
Polynomial multiplication in ML-KEM

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



$(f_0 + f_1 X)$

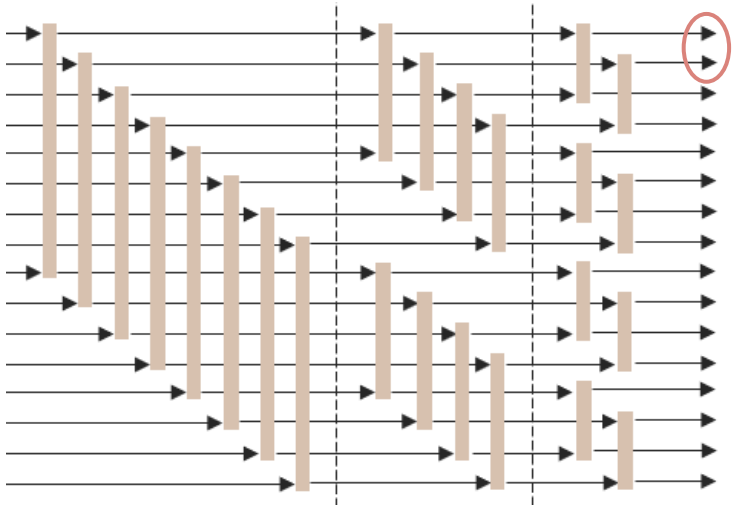
Point-wise Multiplication



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

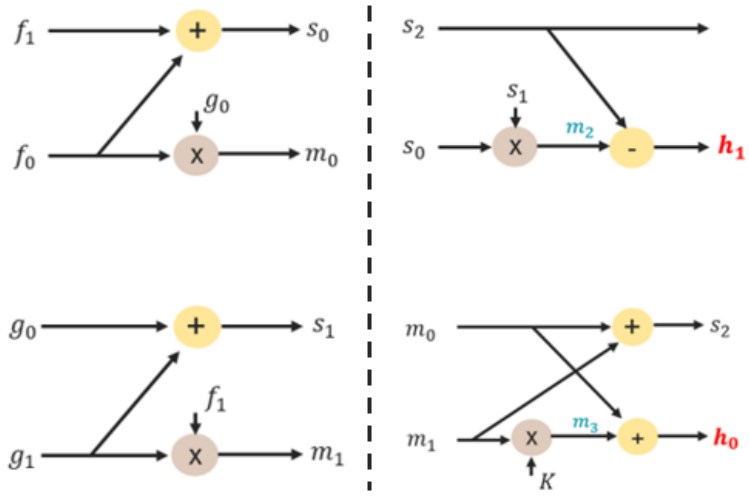
Polynomial multiplication in ML-KEM

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



$(f_0 + f_1 X)$

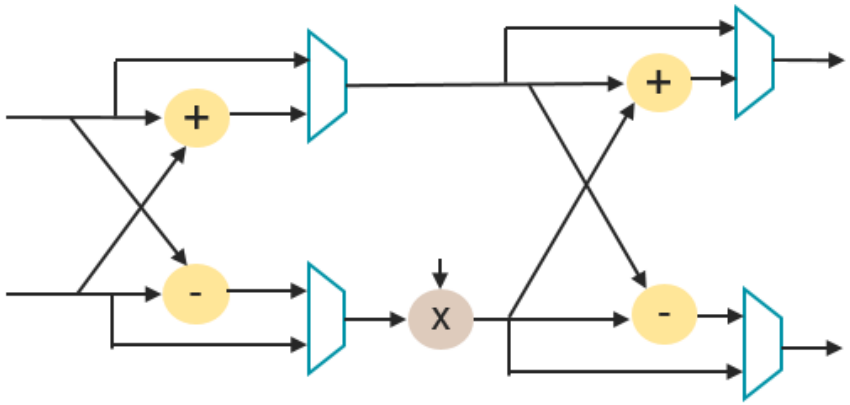
Point-wise Multiplication



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**



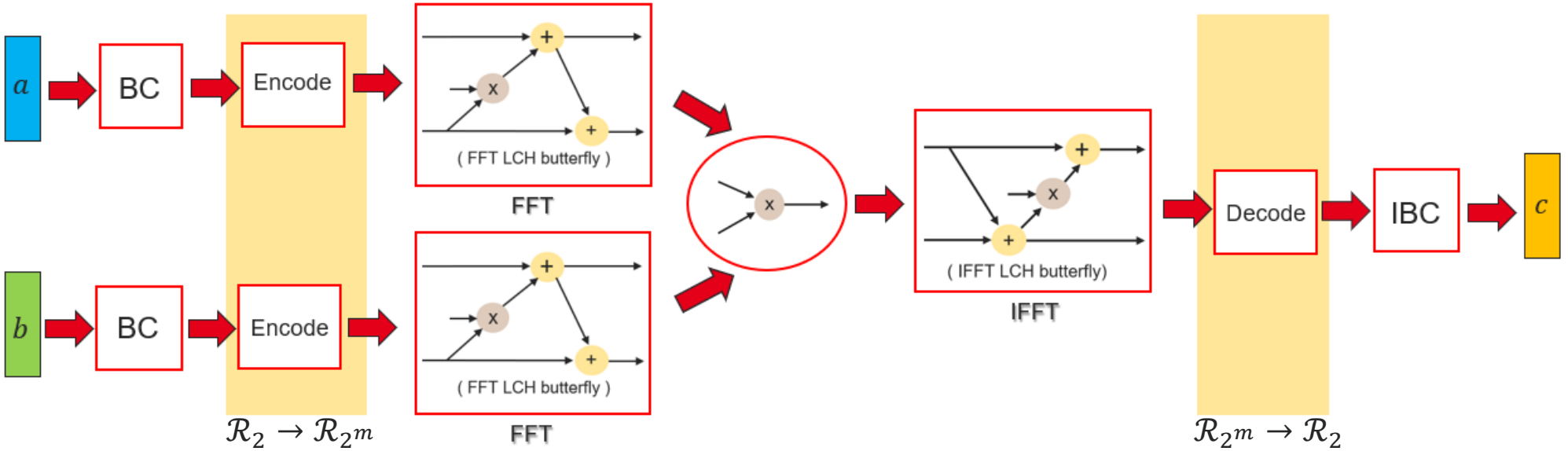
3. Frobenius Additive FFT

The alternative polynomial multiplication strategy for HQC

Frobenius Additive FFT (FAFFT)

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

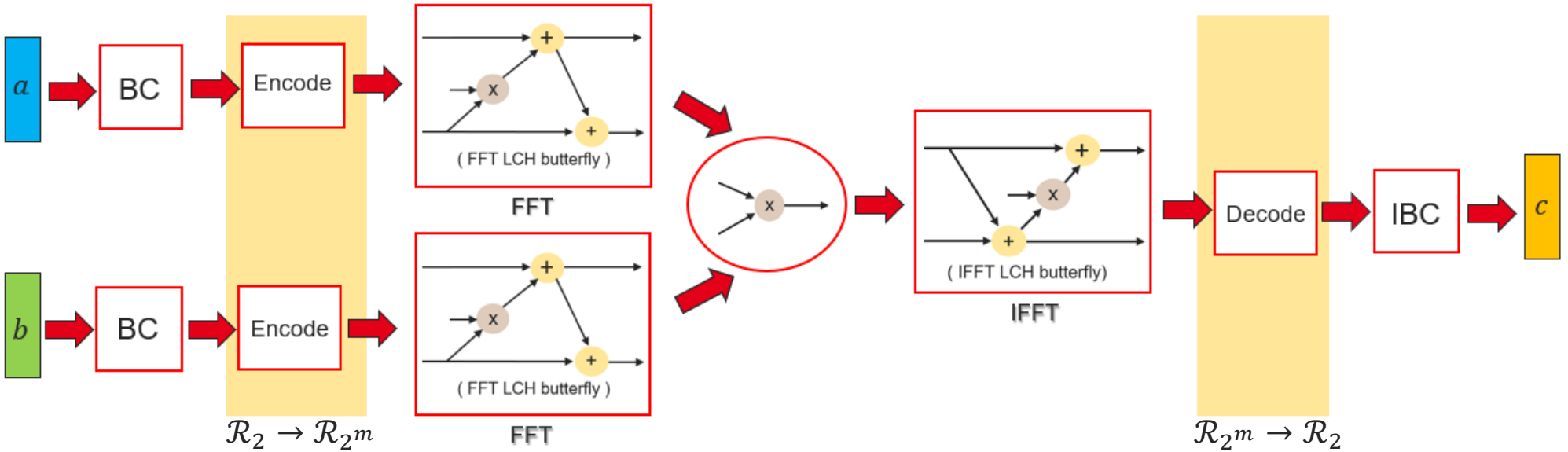
- ❑ New polynomial basis using **Cantor basis**
- ❑ Encode and Decode : **Frobenius map** reduce evaluation points



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

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

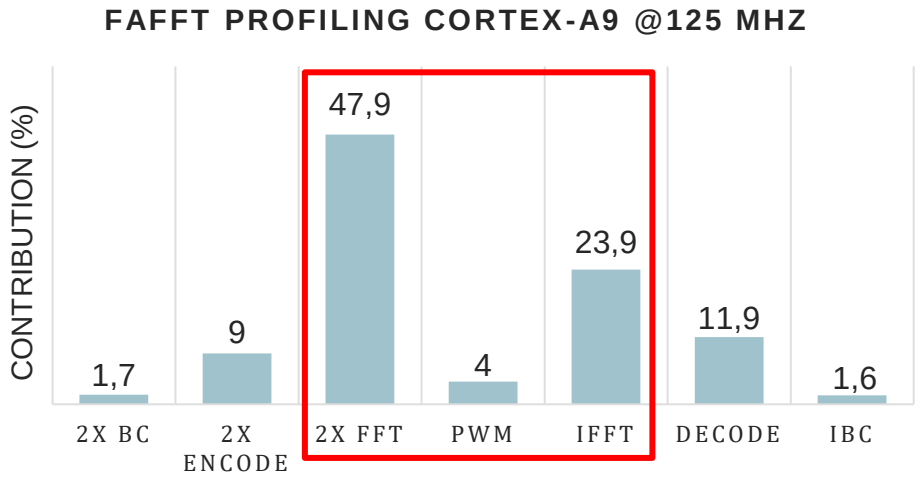
→ **Shift-and-Add strategy**

[CCK+18] Chen, Cheng, Li et al. «Multiplying boolean Polynomials with Frobenius Partitions in Additive Fast Fourier Transform», 2018

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 : $\mathbf{32\,768}$ bits
  - ❑ HQC-192/256 : $\mathbf{65\,536}$ bits

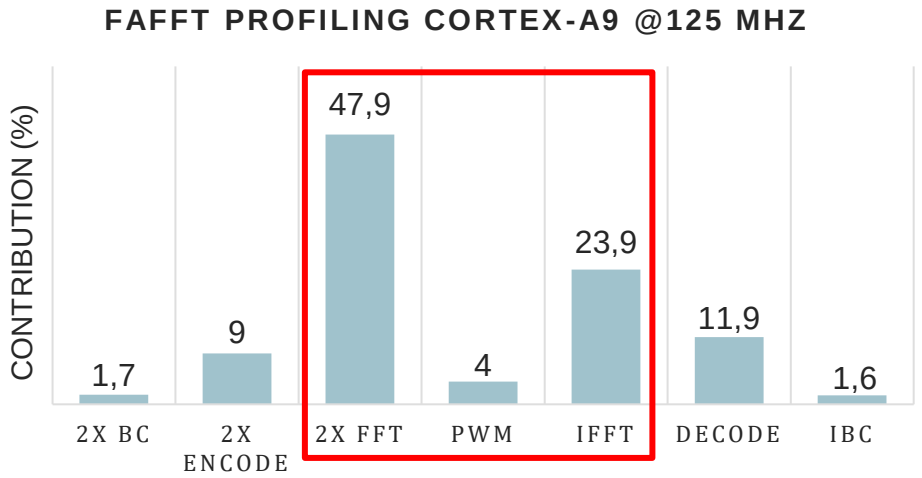


[CCK21] Chen et al, “Optimizing BIKE for the Intel Haswell and ARM Cortex-M4”, 2021

HQC using FAFFT in software

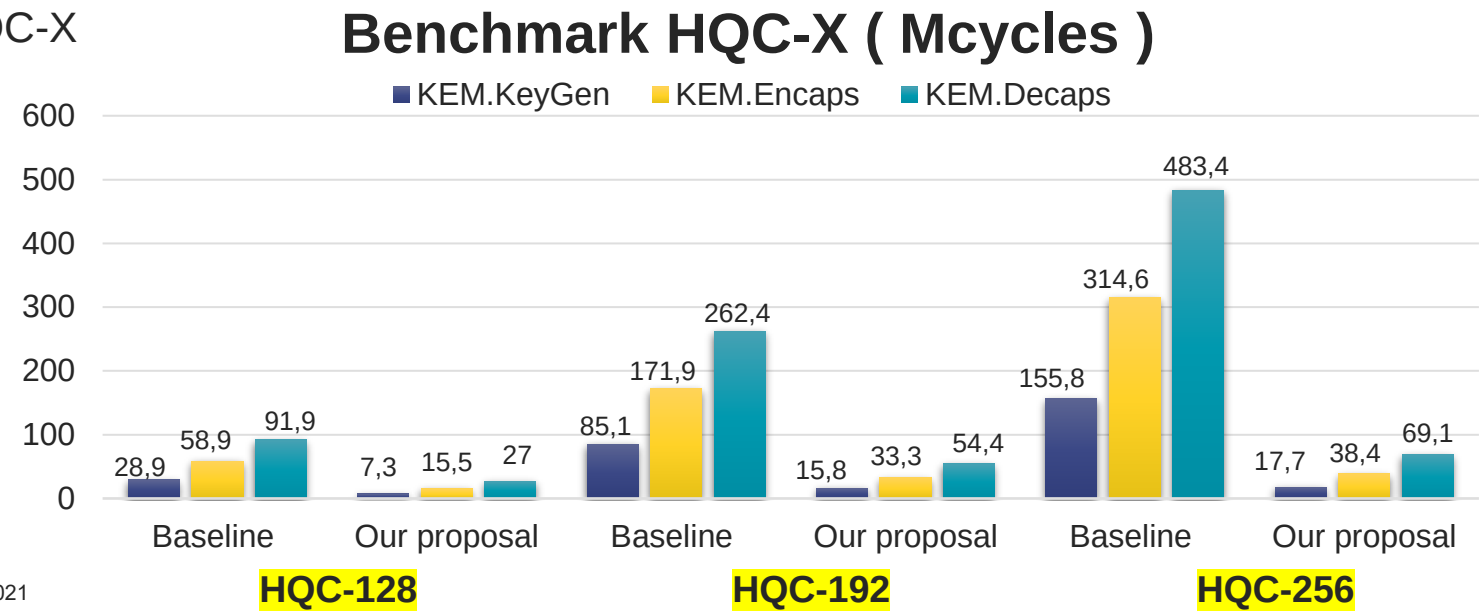
➢ FAFFT profiling

- ❑ 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



[CCK21] Chen et al, "Optimizing BIKE for the Intel Haswell and ARM Cortex-M4", 2021

4. Proposed sharing strategy

The solution to manage different butterfly structures with different mathematical foundations

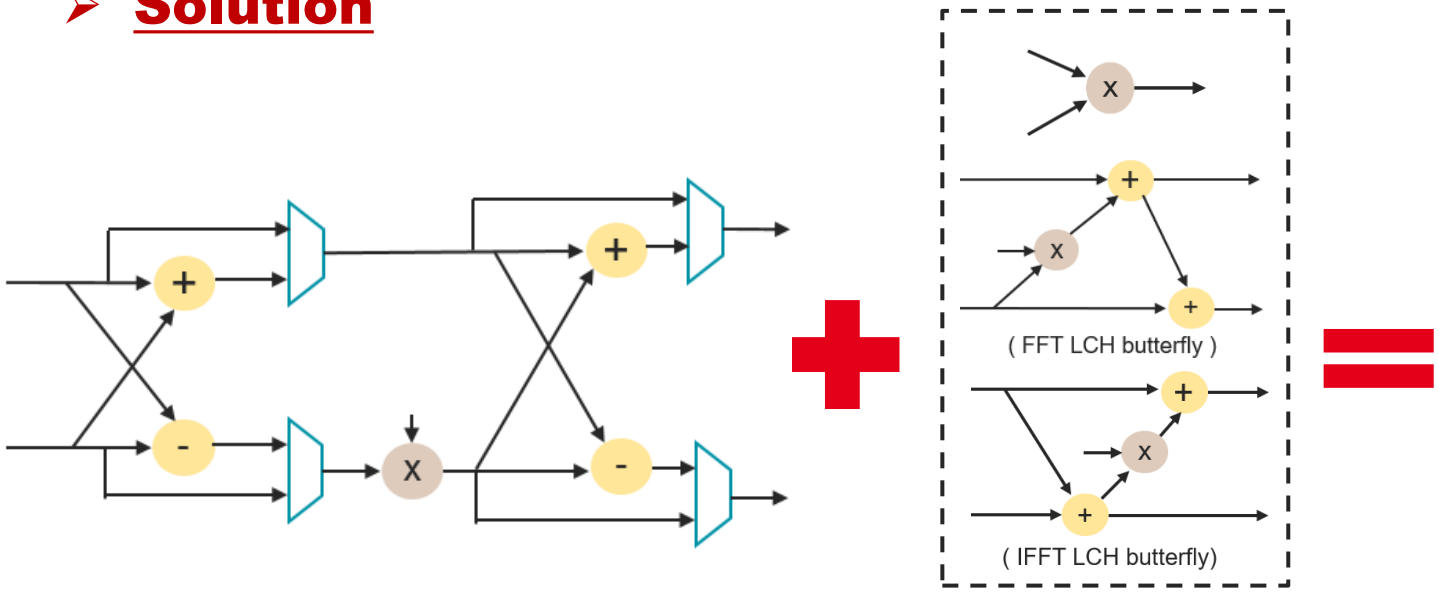
Super-Butterfly

➢ 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
└ *Reduction*	Barrett	Shift-and-Add

➢ Solution



Super-Butterfly

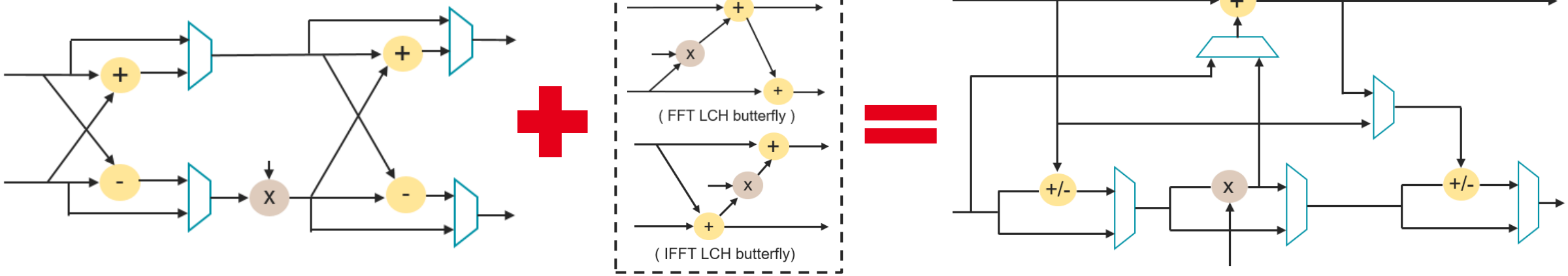
➢ Challenges

- ❑ **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
└ *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**

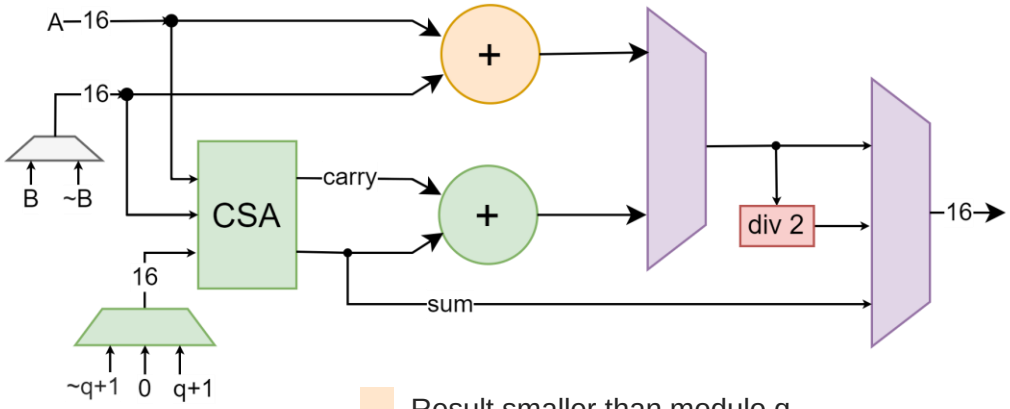
Agility on mathematical foundation

➢ Agile Modular Arithmetics

- ❑ Carryless addition → XOR operation
- ❑ Modular addtion and subtraction
  - ❖ Two's complement to have only additions

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

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



Result smaller than modulo q

Result bigger than modulo q

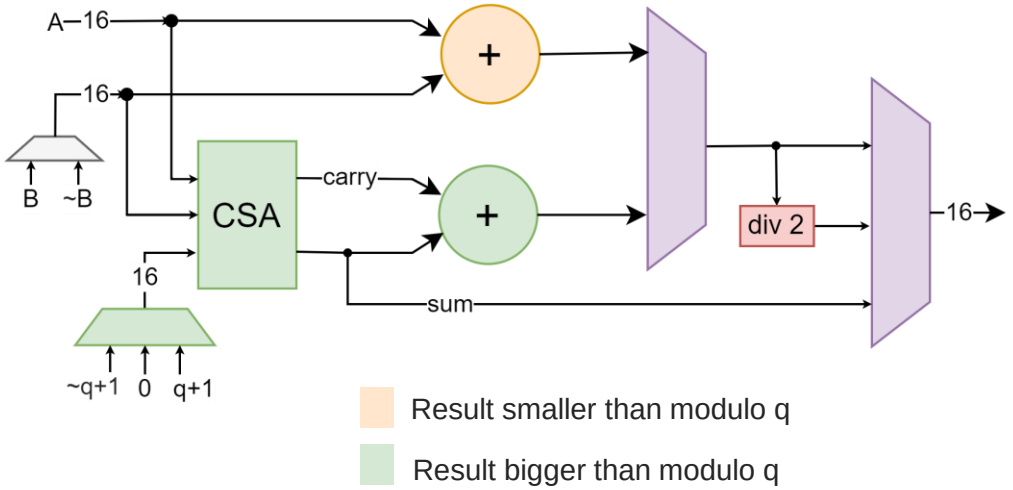
Agility on mathematical foundation

➢ Agile Modular Arithmetics

- ❑ Carryless addition → XOR operation
- ❑ Modular addtion and subtraction
  - ❖ Two's complement to have only additions

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

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



➢ Agile Modular Multiplier

- ❑ Hybrid 2-way Karatsuba
  - ❑ *Schoolbook-based* M_x multiplier

$a = (a_1, a_0)$

$b = (b_1, b_0)$

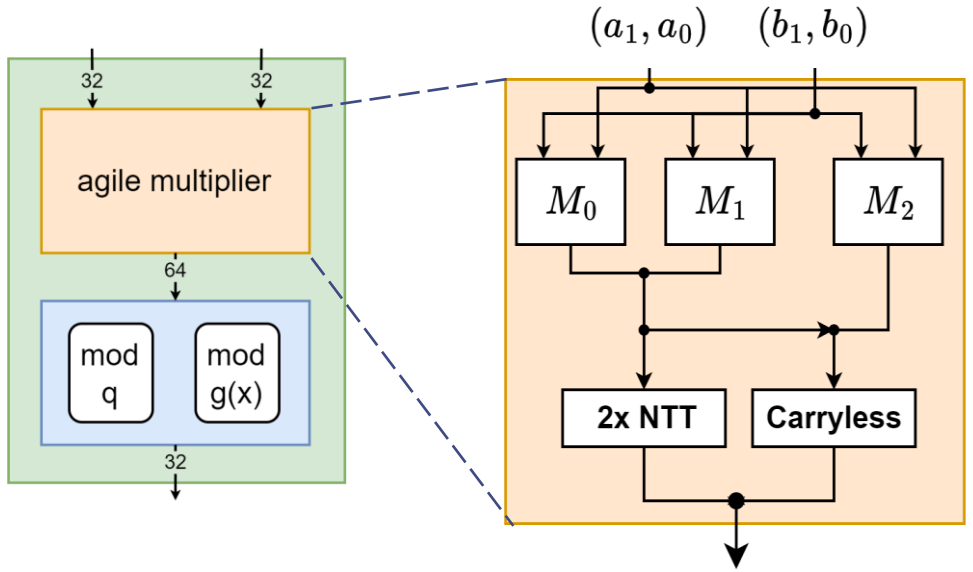
$$z_0 = a_0 \times_{M_0} b_0$$

$$z_1 = a_1 \times_{M_1} b_1$$

NTT butterfly multiplication

$$z_2 = [(a_1 \oplus a_0) \times_{M_2} (b_1 \oplus b_0)] \oplus z_0 \oplus z_1$$

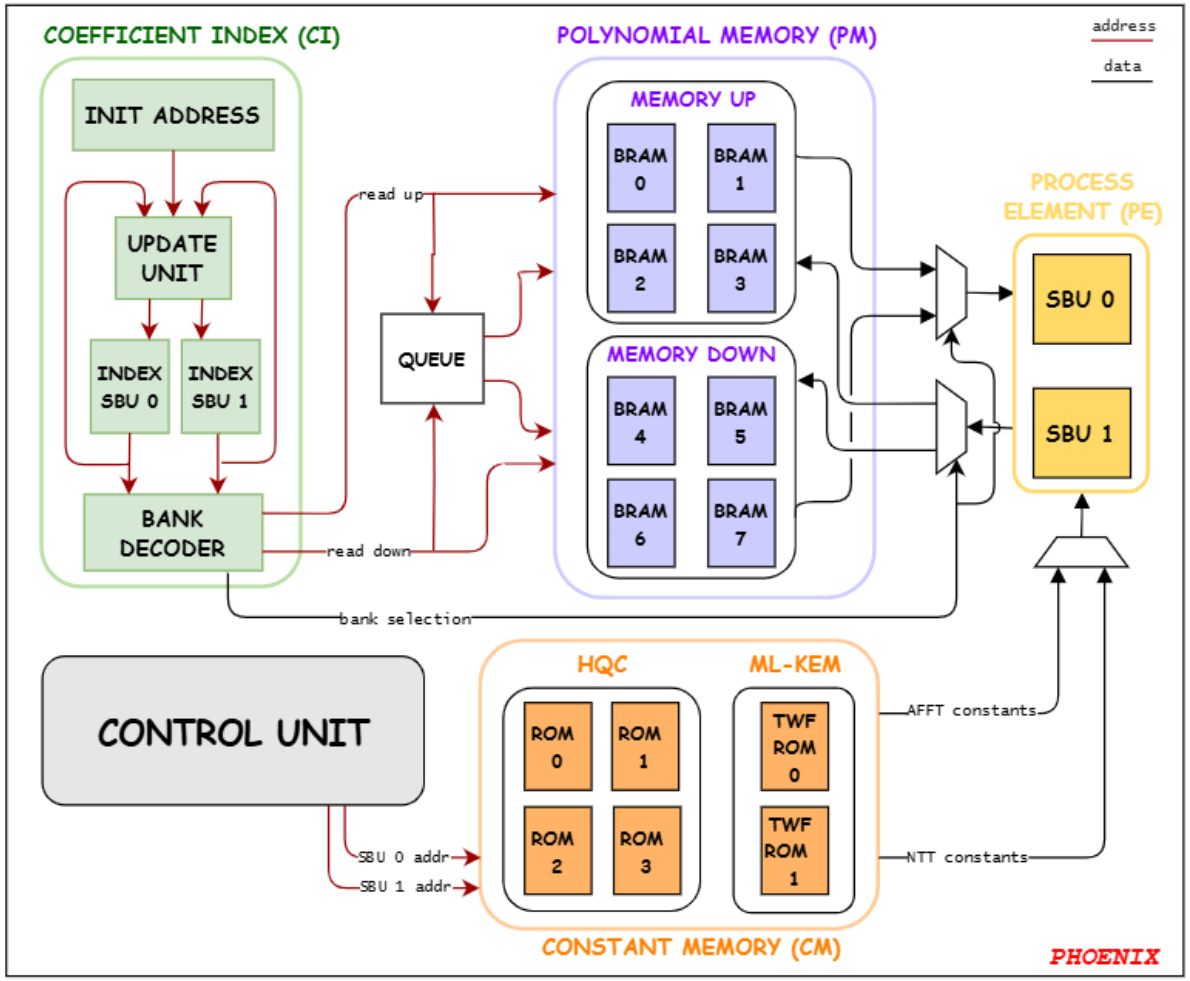
M_0, M_1: agile multiplier (integer + carryless)
M_2: carryless multiplier



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

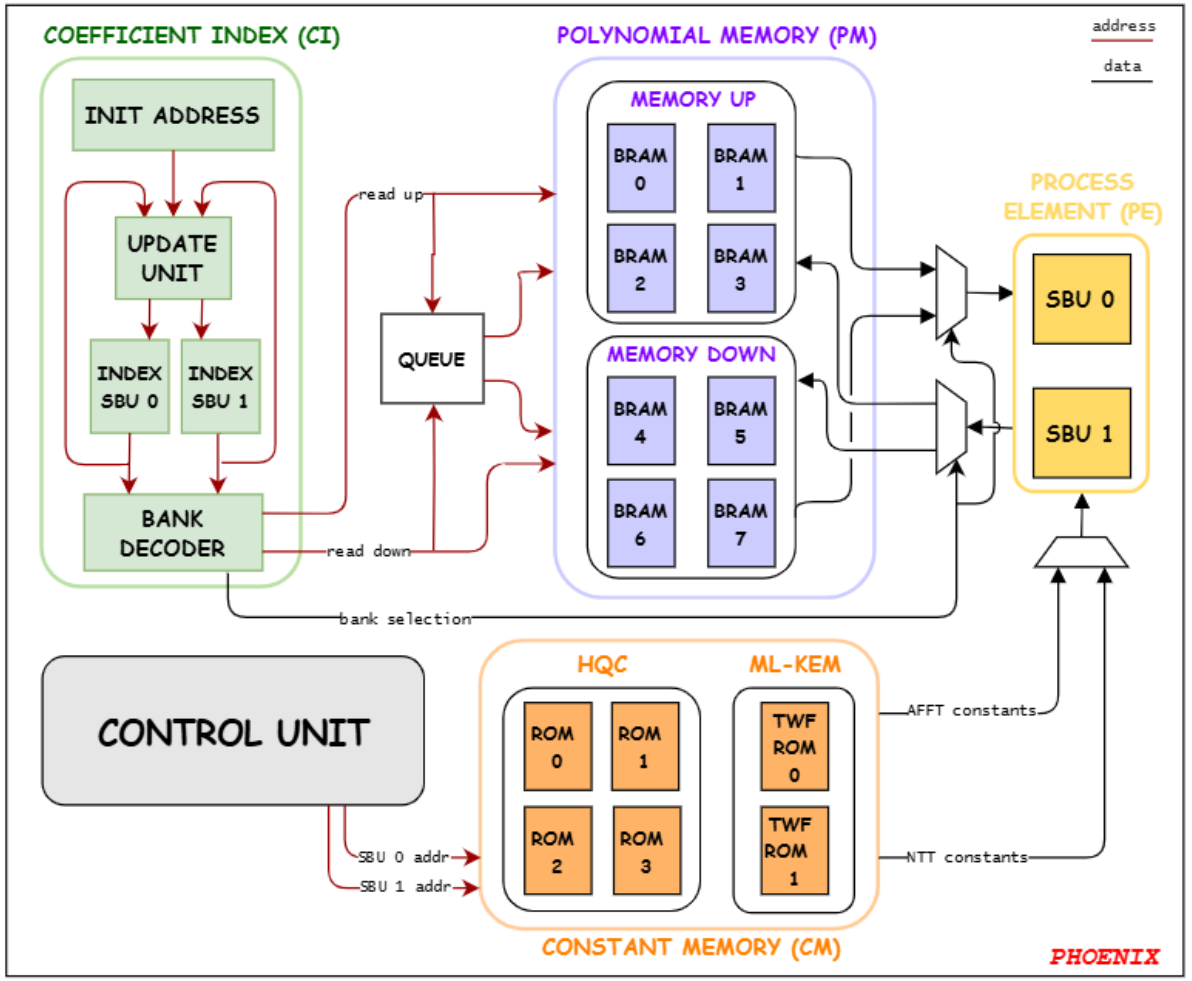
- **Loosely-coupled accelerator**
- **Process Element :** Super-Butterfly valorisation
- **Coefficient Generation :** Generate coefficient pairs to be processed
- **Polynomial Memory :** Conflict-free memory
  Constant Memory : Different constant values
- **Control Unit :** Orchestrate everything

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**

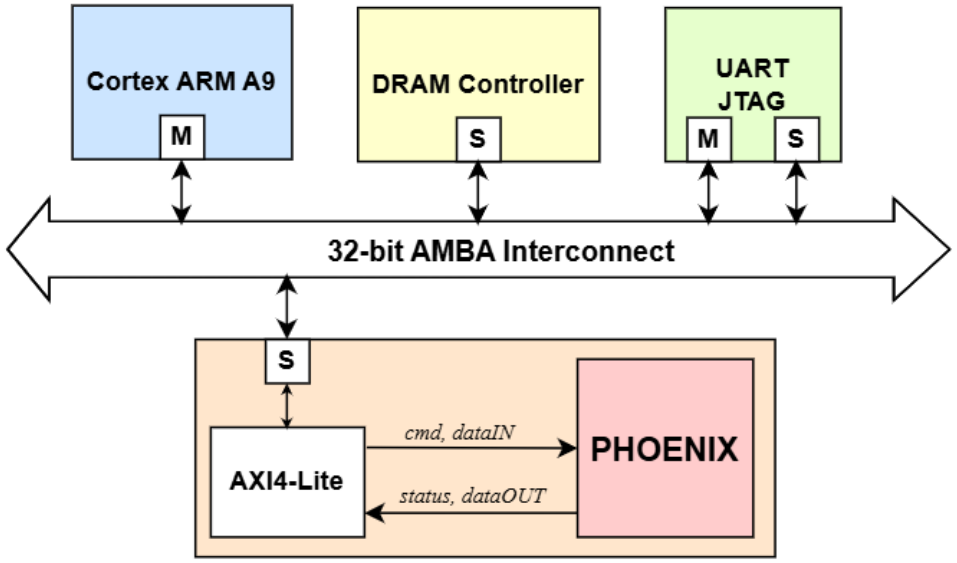


5. Results

Crypto-Agility using PHOENIX for all NIST security levels

Resource utilization on FPGA

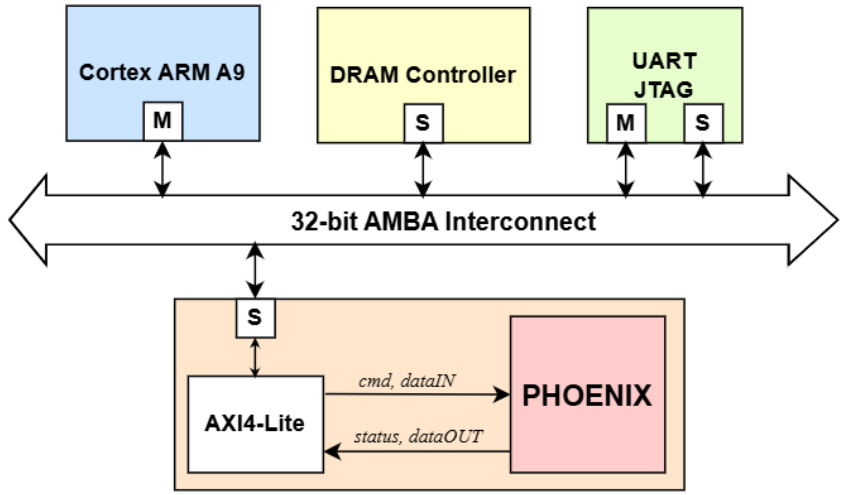
- **System-on-Chip integration**
  - Zybo-Z7 @125 MHz, Artix-7 FPGA
  - AXI4-Lite system bus
  - All NIST security levels



Resource utilization on FPGA

➢ System-on-Chip integration

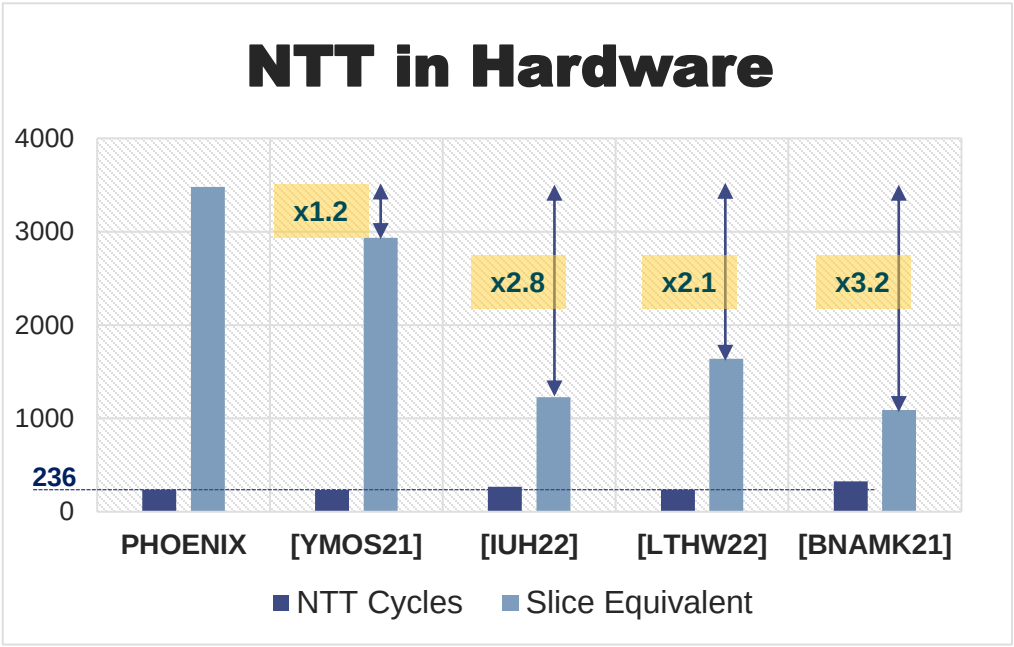
- ❑ Zybo-Z7 @125 MHz, Artix-7 FPGA
- ❑ AXI4-Lite system bus
- ❑ All NIST security levels



➢ 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

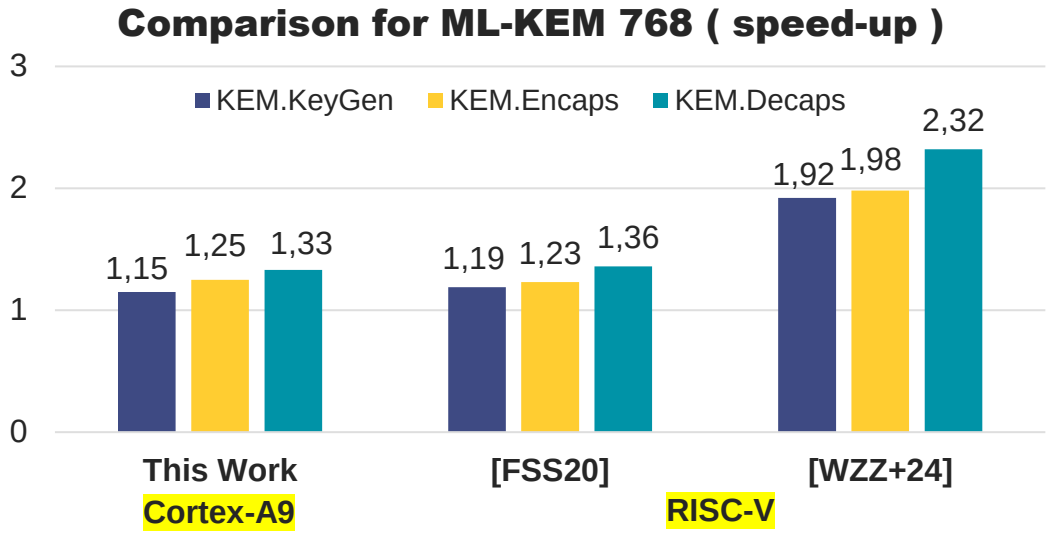


[YMOS21] Bisheh-Niasar et al, "High-Speed NTT-based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography", 2021
[IUH22] Itabashi et al, "Efficient modular poly-nomial multiplier for NTT accelerator of Crystals-Kyber", 2022
[LTHW22] Li et al, "Reconfigurable andhigh-efficiency polynomial multiplication accelerator for Crystals-Kyber.", 2022
[BNAMK21] Bisheh-Niasa 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**

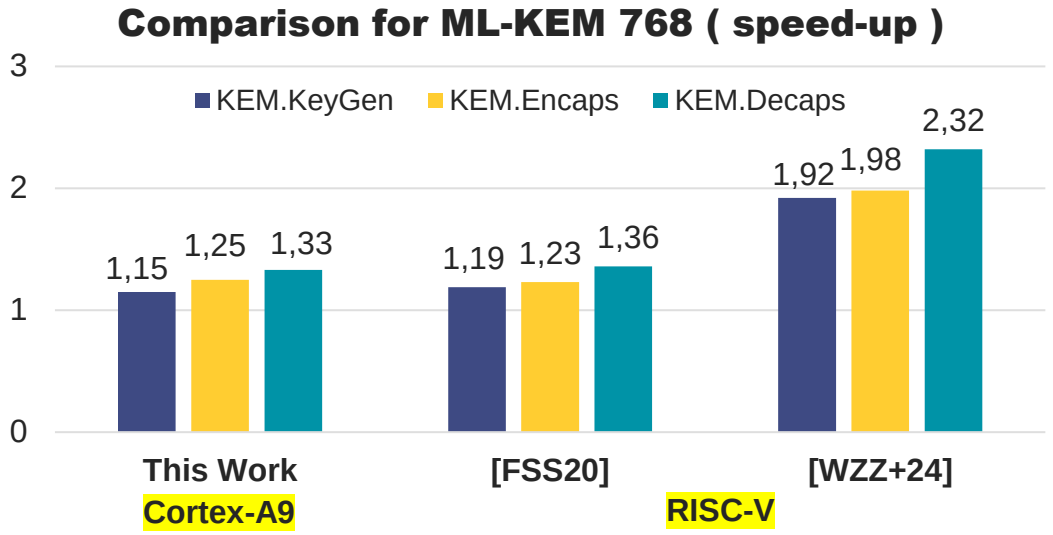


[FSS20] Fritzmann et al, "RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography", 2020
[WZZ+24] Wang et al, "Optimized Hardware-Software Co-Design for Kyber and Dilithium on RISC-V SoC FPGA", 2024

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**

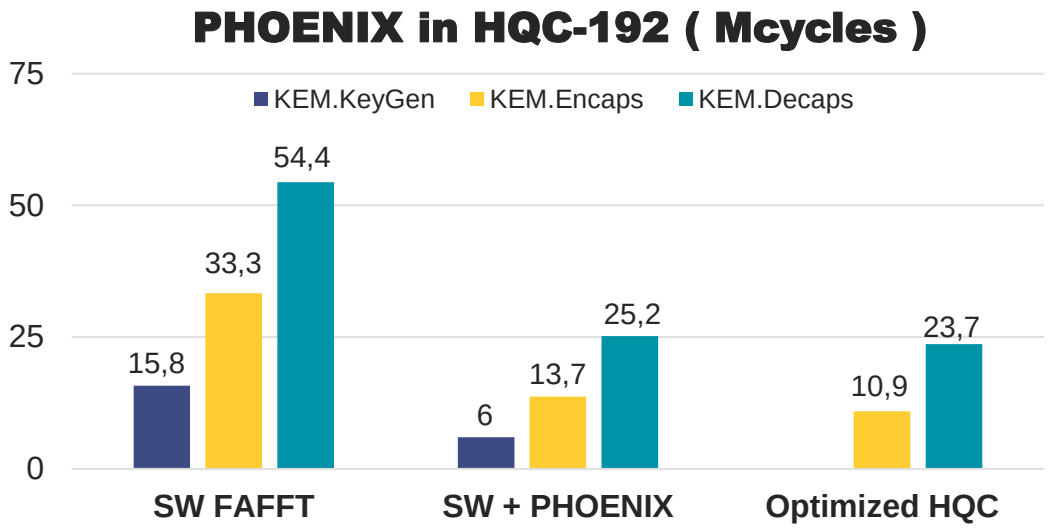


HQC

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

<u>Further optimization for HQC</u> (Optimized HQC)

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



[FSS20] Fritzmann et al, "RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography", 2020
[WZZ+24] Wang et al, "Optimized Hardware-Software Co-Design for Kyber and Dilithium on RISC-V SoC FPGA", 2024

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

- **PHOENIX < NTT + 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

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

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 pratice, 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 $V_i := span\{\beta_0, \beta_0, \dots, \beta_{i-1}\}$. 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) $s_i(x) = s_{i-1}^2(x) + s_{i-1}(x)$

M. Chen, W. Li et al, “Faster Multiplication for Long Binary Polynomials”, 2017

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

HQC

		Standard HQC					*Optimized* HQC			
		Ref.	**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

Alice		Bob
$\mathsf{seed_h} \xleftarrow{\$} \{0,1\}^\lambda$, $\mathbf{h} \xleftarrow{\mathsf{seed_h}} \mathbb{F}_2^n$ $\mathbf{x}, \mathbf{y} \xleftarrow{\$} \mathcal{S}_w^n(\mathbb{F}_2)$, $\mathbf{s} \leftarrow \mathbf{x} + \mathbf{hy}$	$\xrightarrow{\mathsf{seed_h}, \mathbf{s}}$	$\mathbf{r_1}, \mathbf{r_2} \xleftarrow{\$} \mathcal{S}_w^n(\mathbb{F}_2)$, $\mathbf{e} \xleftarrow{\$} \mathcal{S}_w^n(\mathbb{F}_2)$ $\mathbf{u} \leftarrow \mathbf{r_1} + \mathbf{hr_2}$, $\mathbf{v} \leftarrow \mathbf{mG} + \mathbf{sr_2} + \mathbf{e}$
$\mathbf{m} \leftarrow \mathcal{C}.\mathsf{Decode}(\mathbf{v} - \mathbf{uy})$	$\xleftarrow{\mathbf{u}, \mathbf{v}}$	

Alice		Bob
$x, y \leftarrow \mathcal{R}_\omega$ $h \leftarrow \mathcal{R}$ $\dot{x}, \dot{y}, \dot{h} \leftarrow BC(x), BC(y), BC(h)$ $\ddot{x}, \ddot{y}, \ddot{h} \leftarrow FFT_{LCH}(\dot{x}), FFT_{LCH}(\dot{y}), FFT_{LCH}(\dot{h})$ $\ddot{s} = \ddot{x} + \ddot{y} \times_{PWM} \ddot{h}$ $sk \leftarrow (x, \ddot{y}), pk \leftarrow (\ddot{h}, \ddot{s})$	$\xrightarrow{pk}$	$r_1, r_2, e \leftarrow \mathcal{R}_{\omega_r} \times \mathcal{R}_{\omega_r} \times \mathcal{R}_{\omega_e}$ $\dot{r_1}, \dot{r_2} \leftarrow BC(r_1), BC(r_2)$ $\ddot{r_2} \leftarrow FFT_{LCH}(\dot{r_2})$ $\dot{u} = \dot{r_1} + FFT_{LCH}^{-1}(\ddot{r_2} \times_{PWM} \ddot{h})$ $v = mG + BC^{-1}\left(FFT_{LCH}^{-1}(\ddot{r_2} \times_{PWM} \ddot{s})\right) + e$ $ct \leftarrow (\dot{u}, v)$
$\ddot{u} = FFT_{LCH}(BC(u))$ $m = Decode(v - BC^{-1}(FFT_{LCH}^{-1}(\ddot{u} \times_{PWM} \ddot{y})))$	$\xleftarrow{ct}$	