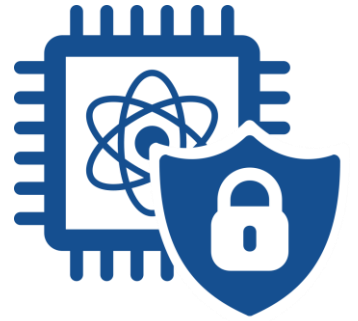


Towards Securing the Quantum Era



Sanjay Deshpande

Department of Electrical & Computer Engineering
Northwestern University

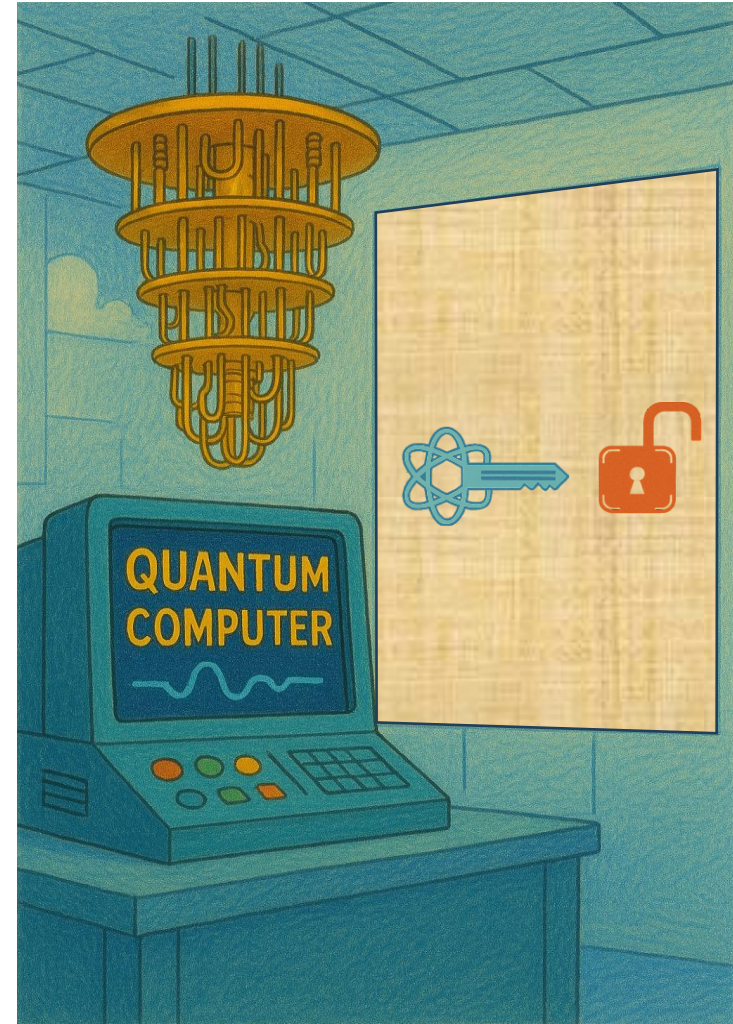


PKIA 2026

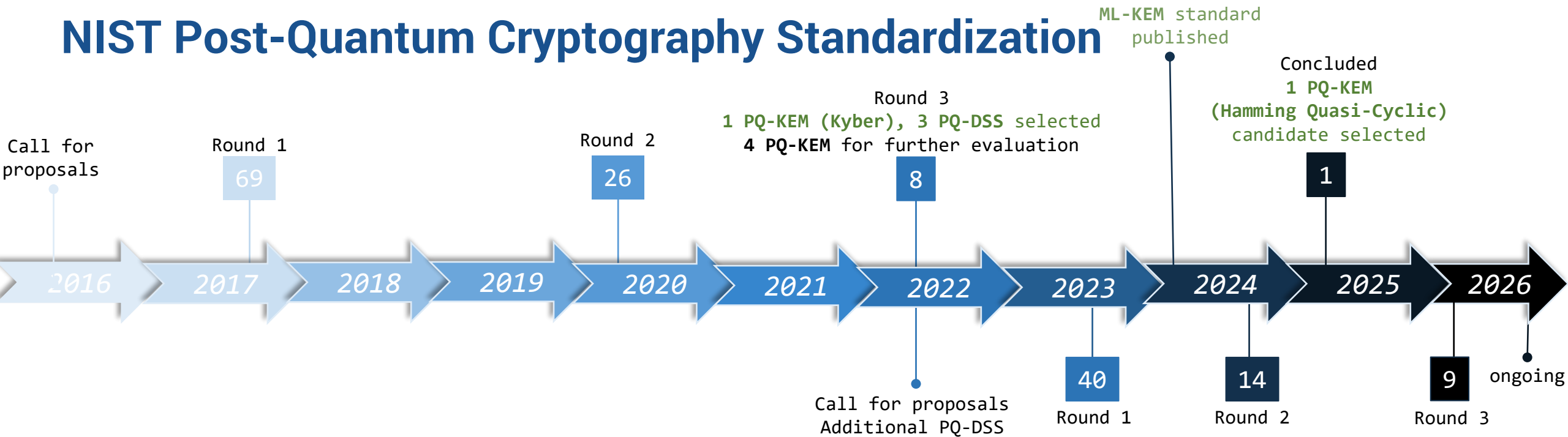
Modern Cryptography

- Symmetric-key cryptography
- Public-key cryptography
 - Shor's algorithm can potentially break existing pre-quantum public-key cryptosystems (e.g., RSA)
- **Quantum-safe Cryptography to the rescue!**
 - Quantum Cryptography
 - Post-quantum cryptography
 - Algorithms run on classical computers
- Post-Quantum Cryptography standardization efforts are in progress around the world

NIST



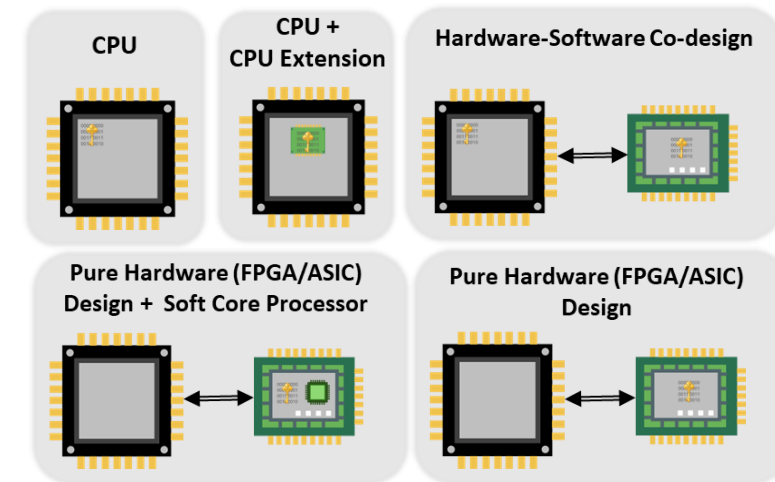
NIST Post-Quantum Cryptography Standardization



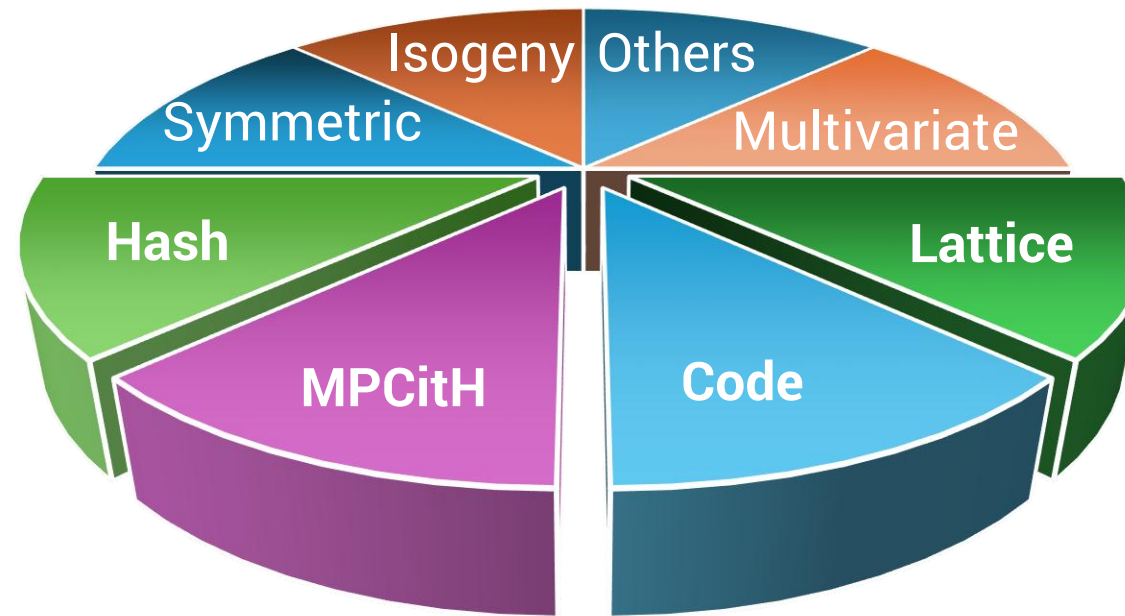
- Submitted candidates belong to two categories
 - **PQ-KEM: Post-Quantum Key Encapsulation Mechanism**
 - PQ-DSS: Post-Quantum Digital Signature Scheme
- Evaluation Criteria
 - Security
 - **Hardware and Software implementation efficiency and security**
 - Cost (key size, signature size, etc.)

Secure and Efficient Implementations of Cryptographic Algorithms

- Target platforms:
 - CPU
 - Microcontrollers
 - **Field-Programmable Gate Array (FPGA)**
 - Application Specific Integrated Circuit (ASIC)

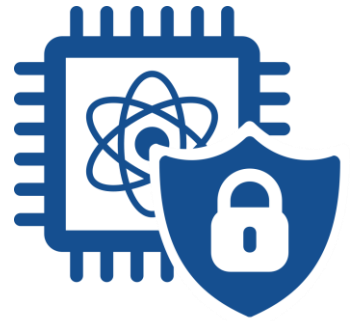


Today's Talk

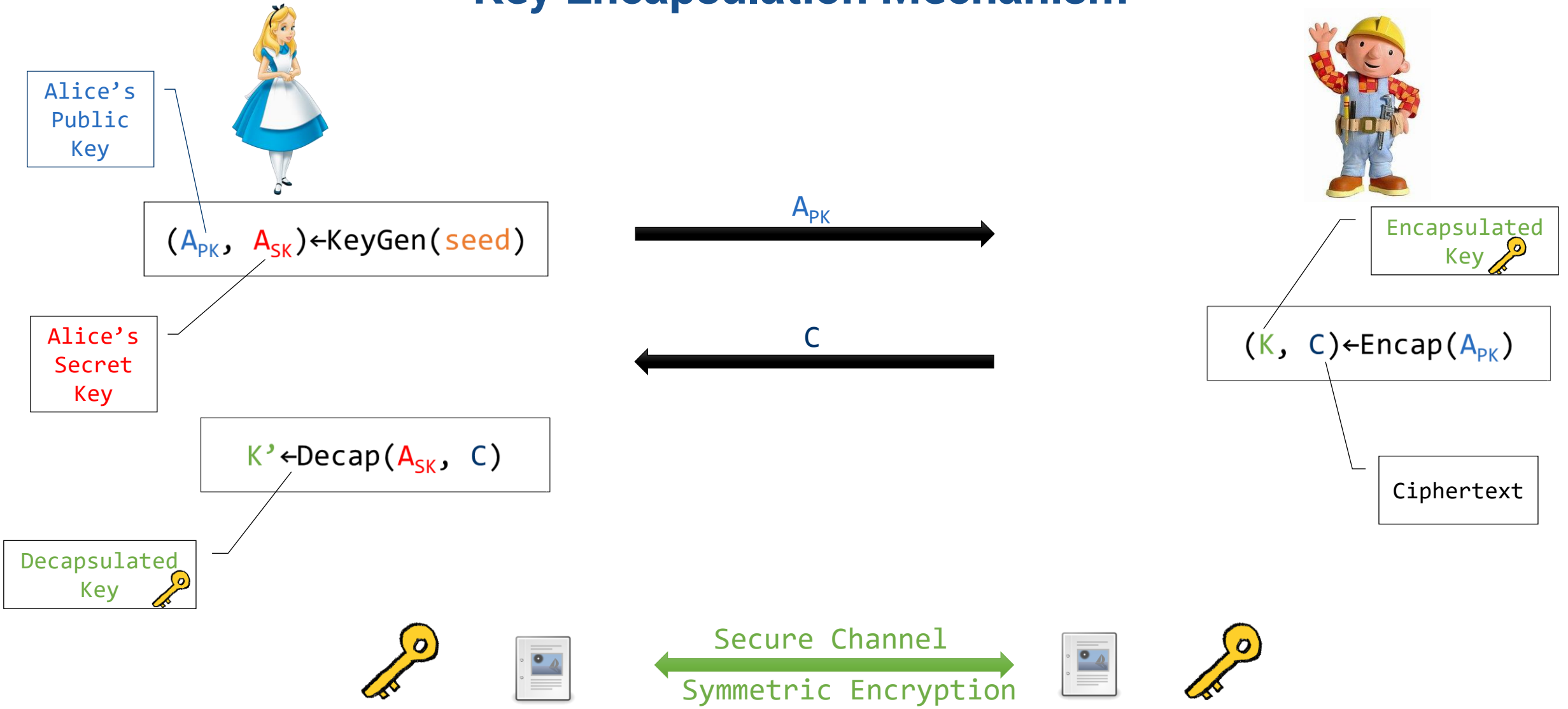


Hamming Quasi-Cyclic (HQC)

Hamming Quasi-Cyclic (HQC)



Key Encapsulation Mechanism



Hamming Quasi Cyclic (HQC)

- A code-based KEM scheme based on the syndrome decoding hard problem.
- Construction is based on concatenated codes:
 - Shortened Reed Solomon code and
 - Duplicated Reed-Muller code

Parameter Set	Security	n	w	$ ek $	$ dk $	$ ct $
HQC-1	128	17,669	66	2,241B	2,321B	4,433B
HQC-3	192	35,851	100	4,514B	4,602B	8,978B
HQC-5	256	57,637	131	7,237B	7,333B	14,421B
X25519	128	–	–	32B	32B	32B

n - degree of the polynomial

w - weight

keypair – (ek,dk)

ct - ciphertext

☐ *Quantum-safe*

☒ *Not Quantum-safe*

Syndrome Decoding Problem

- h, x, y are polynomials in $\mathbb{F}_2[X]/(X^n-1)$
- Setup:
 - h – sampled from a pseudo-random number generator (PRNG)
 - x, y – fixed weight vectors (sparse polynomials) whose locations are sampled from a PRNG

$$\begin{array}{ccccccc} & x & & & h & & y \\ \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare & + & \square \square \square \square \square \square \square \square & \times & \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare & = & S = x + h.y \\ \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare & & & & & & \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \end{array}$$

-  pseudorandom polynomial
-  Sparse fixed-weight polynomial
-  Sparse fixed-weight polynomial

h and S are public
 x and y are secret
Given h and S , finding x and y is
hard!

Key Generation

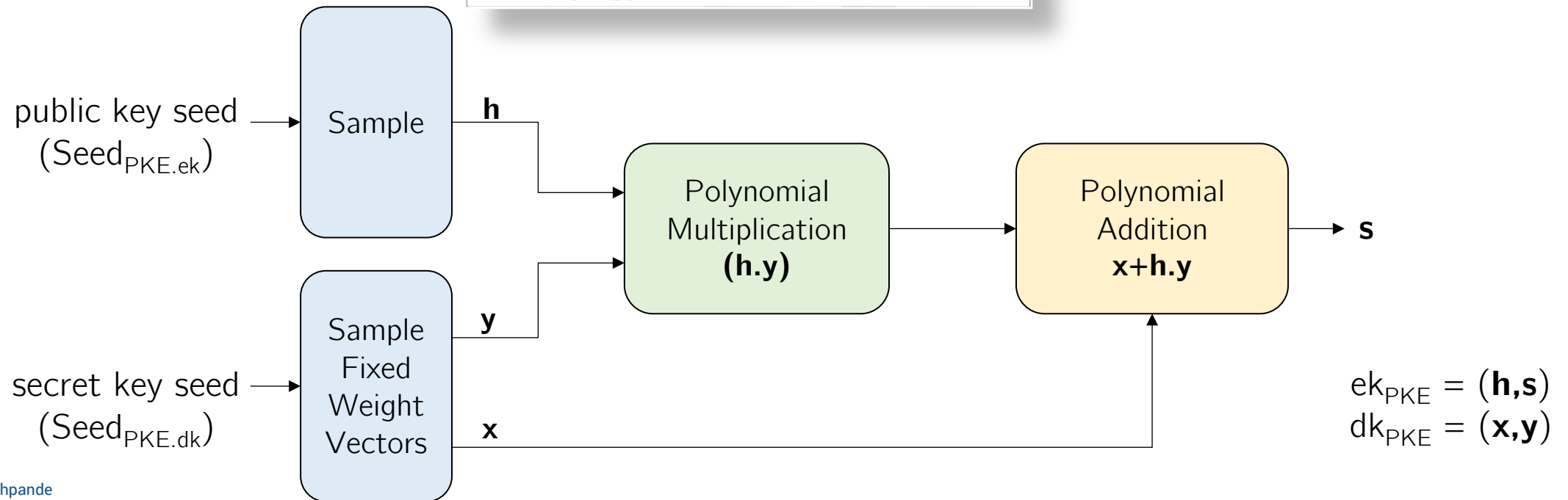
```
HQC-KEM.Keygen()
Input: None.
Output: Encapsulation key  $ek_{KEM}$ , decapsulation key  $dk_{KEM}$ .

▷ Sample  $seed_{KEM}$ 
1.  $seed_{KEM} \leftarrow \mathcal{B}^{|seed|}$ 

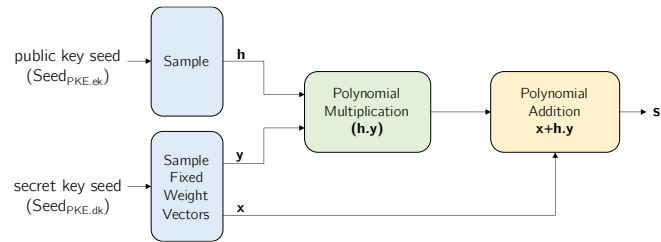
▷ Compute  $seed_{PKE}$  and randomness  $\sigma$ 
2.  $ctx_{KEM} \leftarrow \text{XOF.Init}(seed_{KEM})$ 
3.  $(ctx_{KEM}, seed_{PKE}) \leftarrow \text{XOF.GetBytes}(ctx_{KEM}, |seed|)$ 
4.  $(ctx_{KEM}, \sigma) \leftarrow \text{XOF.GetBytes}(ctx_{KEM}, |k|)$ 

▷ Compute HQC-PKE keypair
5.  $(ek_{PKE}, dk_{PKE}) \leftarrow \text{HQC-PKE.Keygen}(seed_{PKE})$ 

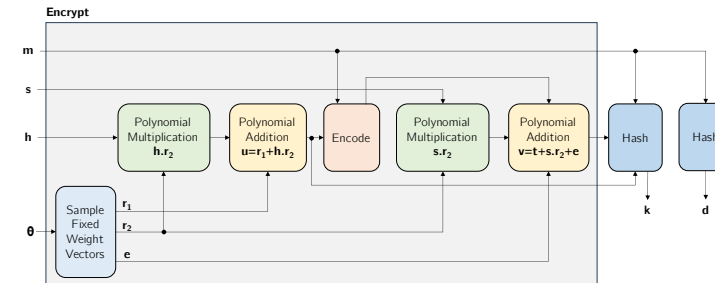
▷ Compute HQC-KEM keypair
6.  $ek_{KEM} \leftarrow ek_{PKE}$ 
7.  $dk_{KEM} \leftarrow (ek_{KEM}, dk_{PKE}, \sigma, seed_{KEM})$ 
8. return  $(ek_{KEM}, dk_{KEM})$ 
```



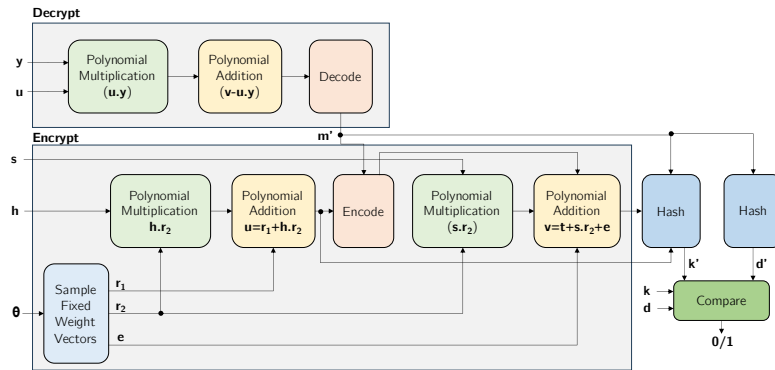
HQC Primitives



Key Generation

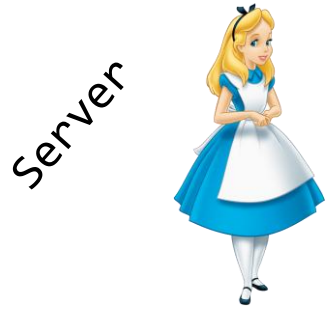


Encapsulation



Decapsulation

Performance Comparison HQC vs Elliptic Curve Cryptography



Server



Client

$$K' \leftarrow \text{Decap}(A_{SK}, C)$$

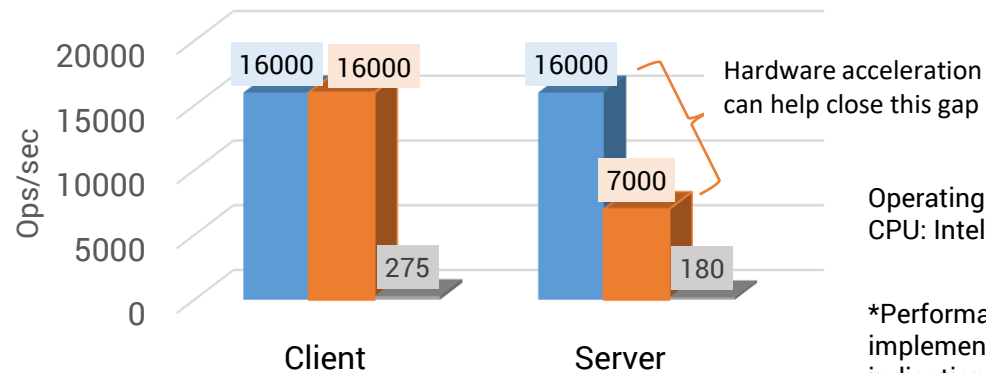
 **K' - Decapsulated Key**

*Key generation is performed offline on the server side

$$(K, C) \leftarrow \text{Encap}(A_{PK})$$

 **K - Encapsulated Key
C - Ciphertext**

Operations Per Second Comparison
on a CPU

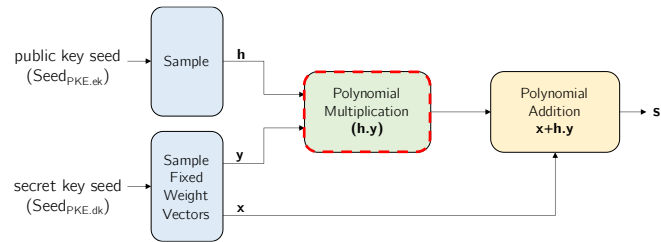


Operating frequency:
CPU: Intel Core i7-11850H – 2.5 GHz

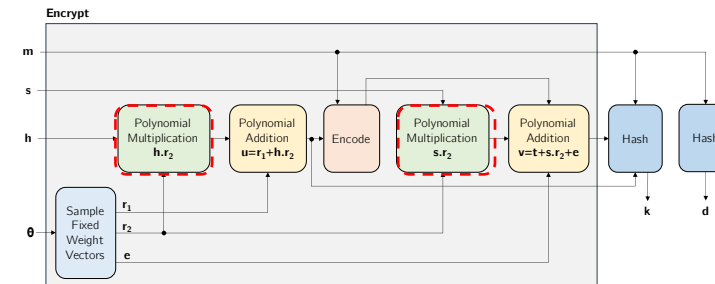
*Performance varies considerably by hardware platform and implementation constraints, and should be taken as a rough indication only.

■ X25519 [SAC'15] ■ HQC - Optimized Software [NIST'25] ■ HQC - Reference Software [NIST'25]

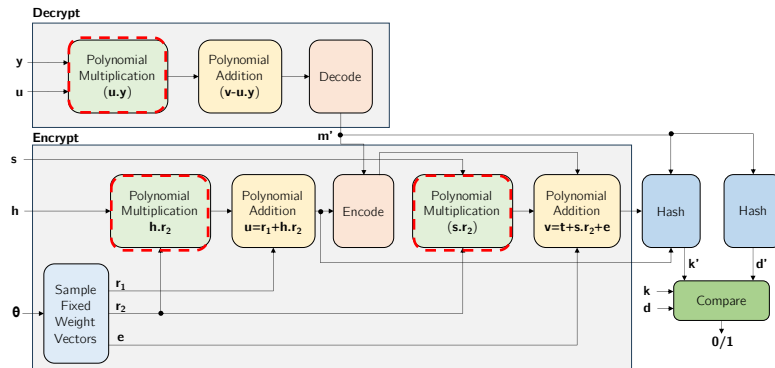
HQC Primitives



Key Generation



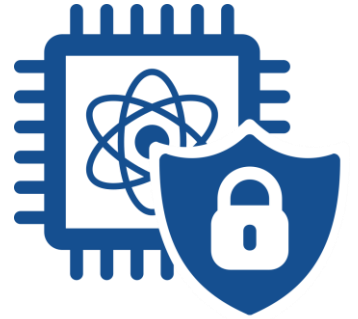
Encapsulation



Decapsulation

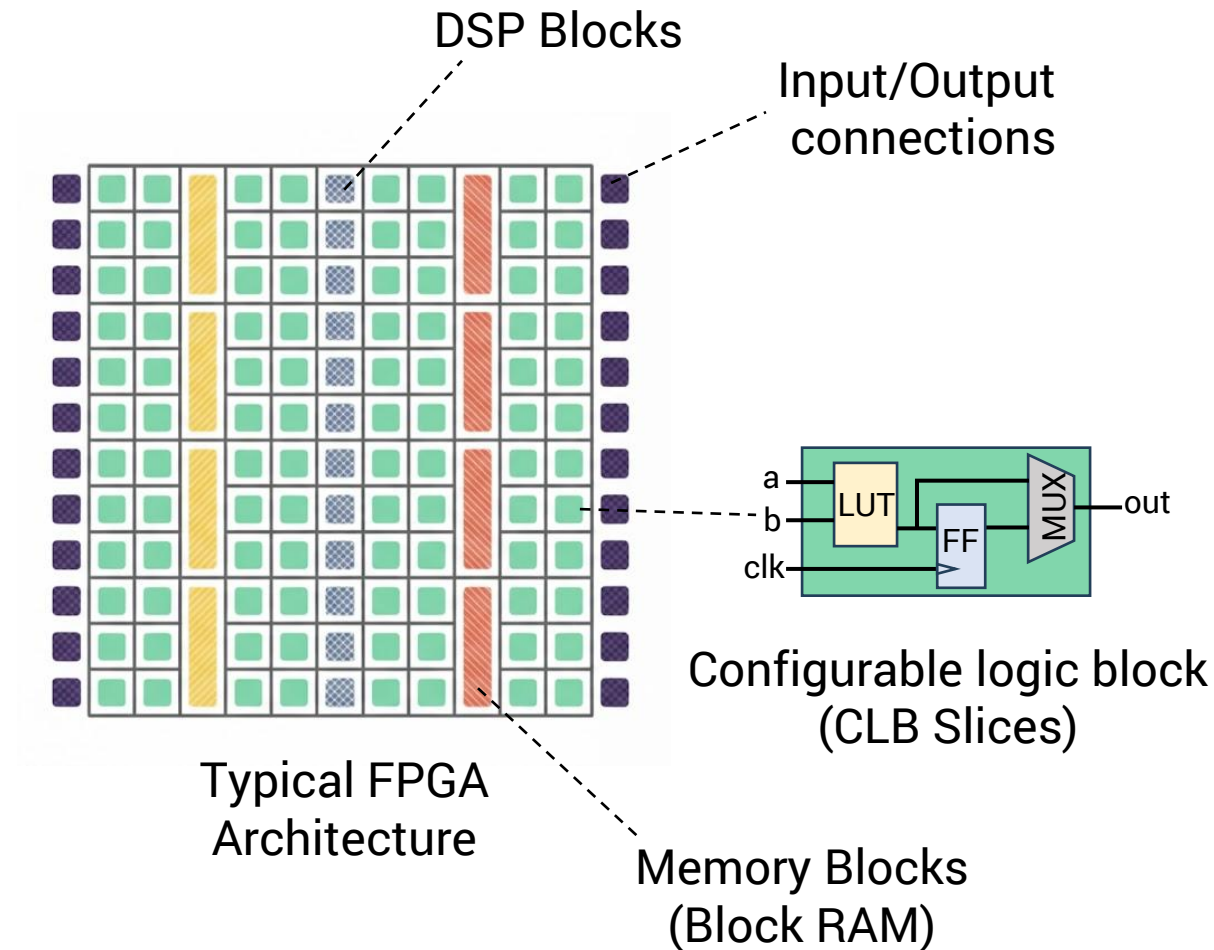
Performance Bottleneck.
95-96% across different parameter sets

Hardware Acceleration Using FPGAs



Field Programmable Gate Array (FPGA) Implementation Performance and Evaluation Metrics

- FPGA is a reconfigurable substrate
- Area
- Time
- Time-Area product
 - $\text{Time} * \{\max\{\text{LUT}/4, \text{FF}/8\} + 128 * \text{BRAM} + 100 * \text{DSP}\}$
- For reference (AMD Artix 7 200 T FPGA):
 - Look up Tables (LUTs) = 215,360
 - Digital Signal Processing Blocks (DSPs) = 740
 - Flipflops (FFs) = 269,200
 - Block RAM (BRAM) = 365 (each block can store 36 Kb)



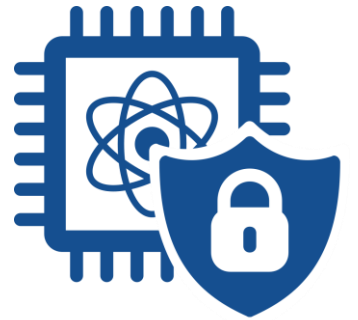
Hardware vs Software Development

	CPU	GPU	FPGA	ASIC
Engineering time	short	medium	long	very long
Compilation time	short	short	long	very long
Debugging	easy	medium	hard	very hard
Maintainability	easy	medium	hard	very hard

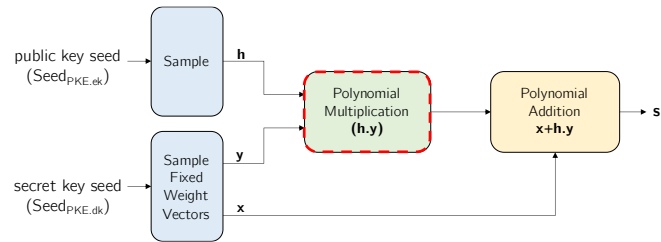


Hardware Implementation

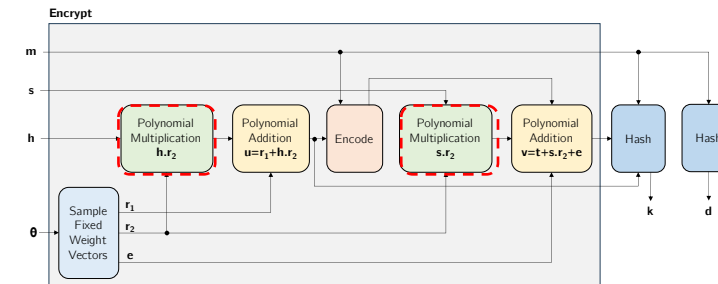
HQC



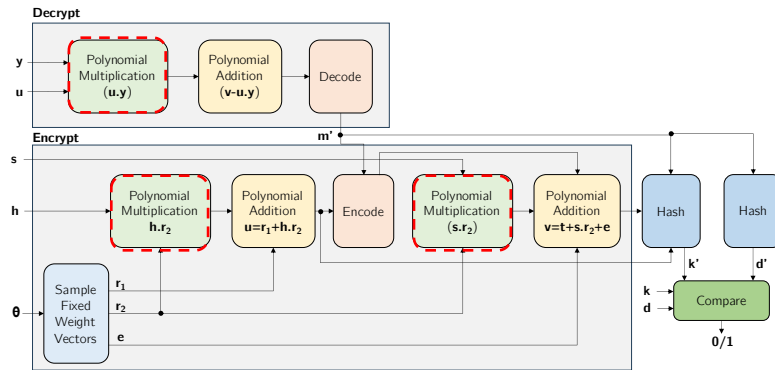
HQC Primitives



Key Generation



Encapsulation

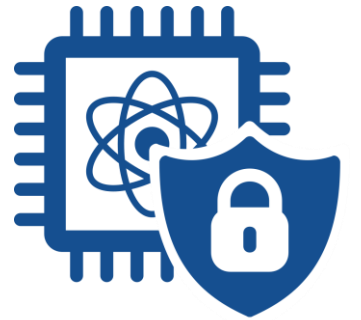


Decapsulation

Performance Bottleneck.
95-96% across different parameter sets

Performance Bottleneck

Polynomial Multiplication



Polynomial Multiplication

A & B are polynomials in $\mathbb{F}_2[X]/(X^n-1)$

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_nx^n$$

$$c(x) = A(x) \cdot B(x) = \sum_{k=0}^{n+n} c_k x^k \quad \text{Where each } c_k = \sum_{i=0}^k a_i b_{k-i}$$

Polynomial Multiplication – Schoolbook Technique

A & B are polynomials in $\mathbb{F}_2[X]/(X^n-1)$

$$A(x)=a_0+a_1x$$

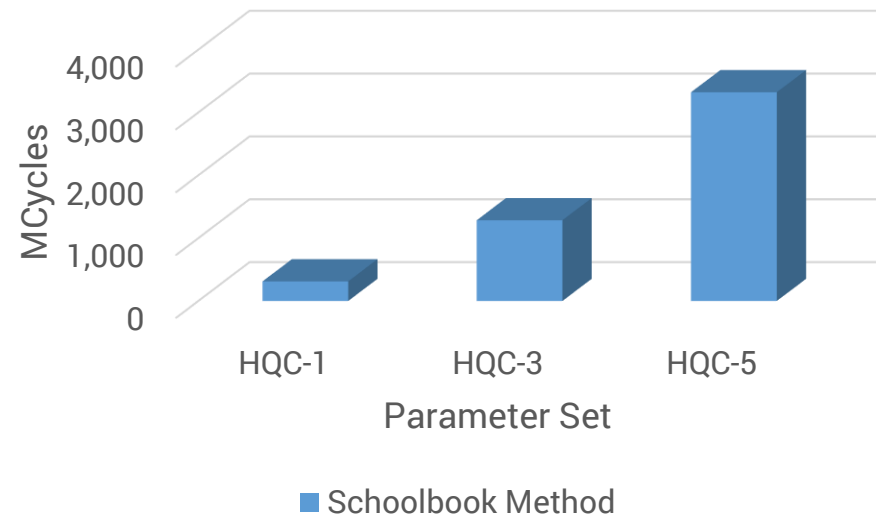
$$B(x)=b_0+b_1x$$

$$C(x)=a_1b_1x^2 + (a_1b_0 + a_0b_1)x + a_0b_0$$

Overall complexity = $O(n^2)$

Complexity of polynomial multiplication

Parameter Set	n	w
HQC-1	17,669	66
HQC-3	35,851	100
HQC-5	57,637	131



Polynomial Multiplication – Karatsuba Algorithm

A & B are polynomials in $\mathbb{F}_2[X]/(X^n-1)$

$$A(x)=a_0+a_1x$$

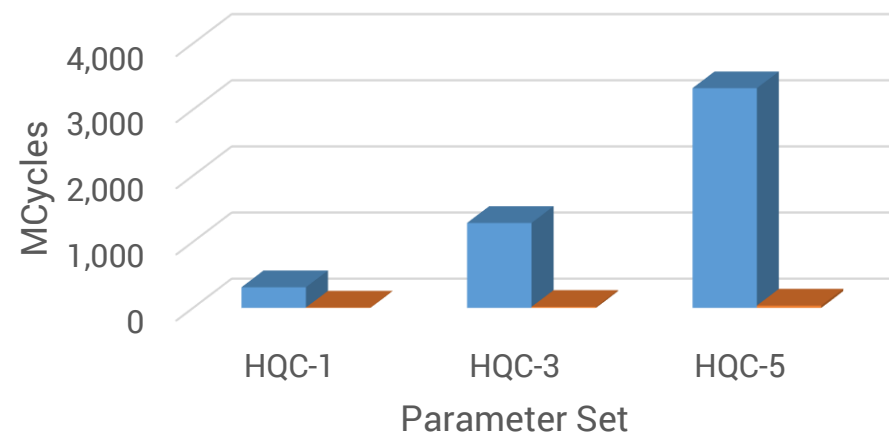
$$B(x)=b_0+b_1x$$

$$C(x)=\underbrace{a_1b_1}_{c_2}x^2 + \underbrace{(a_1b_0 + a_0b_1)}_{c_1}x + \underbrace{a_0b_0}_{c_0}$$

$$c_1=(a_0+a_1)(b_0+b_1)-c_0-c_2$$

Complexity of polynomial multiplication

Parameter Set	n	w
HQC-1	17,669	66
HQC-3	35,851	100
HQC-5	57,637	131



Schoolbook = $O(n^2)$
Karatsuba = $O(n^{1.585})$

■ Schoolbook Method ■ Karatsuba

Polynomial Multiplication – Toom-Cook Algorithm

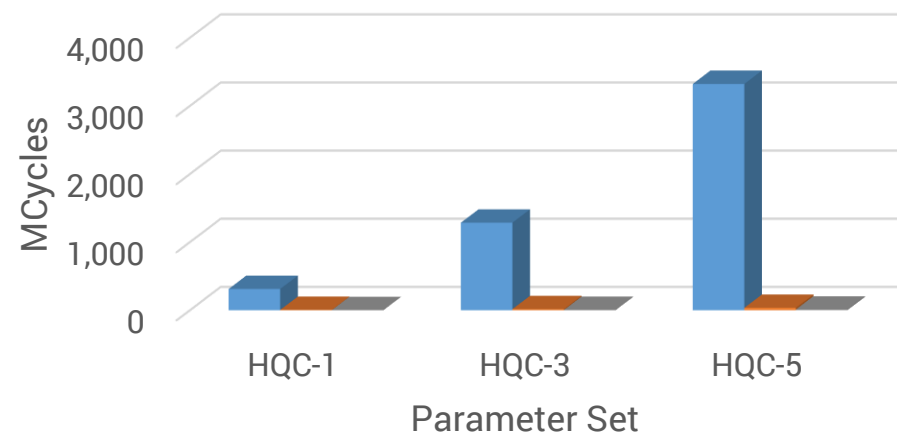
A & B are polynomials in $\mathbb{F}_2[X]/(X^n-1)$

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_nx^n$$

Parameter Set	n	w
HQC-1	17,669	66
HQC-3	35,851	100
HQC-5	57,637	131

Complexity of polynomial multiplication



Schoolbook = $O(n^2)$
 Karatsuba = $O(n^{1.585})$
 Toom-Cook = $O(n^{1.465})$

■ Schoolbook Method ■ Karatsuba ■ Toom-Cook

Polynomial Multiplication - Can we do better?

A & B are polynomials in $\mathbb{F}_2[X]/(X^n-1)$

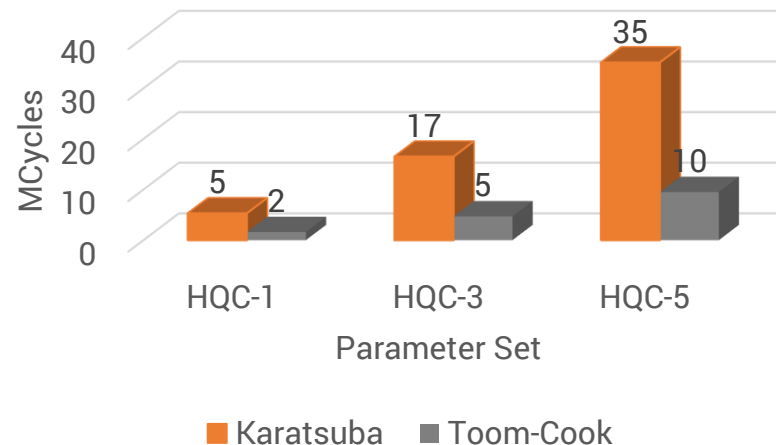
$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

$$B(x) = 0 + b_1x + 0 + \dots + b_mx^m$$

One of the inputs to the polynomial multiplier is a sparse fixed weight vector.

Parameter Set	n	w
HQC-1	17,669	66
HQC-3	35,851	100
HQC-5	57,637	131

Complexity of polynomial multiplication



$$\begin{aligned} \text{Karatsuba} &= O(n^{1.585}) \\ \text{Toom-Cook} &= O(n^{1.465}) \end{aligned}$$

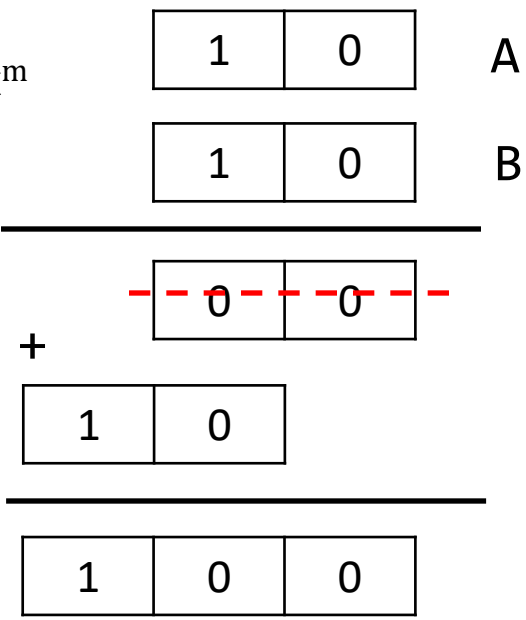
Polynomial Multiplication involving Sparse Polynomials

A & B are polynomials in $\mathbb{F}_2[X]/(X^n-1)$

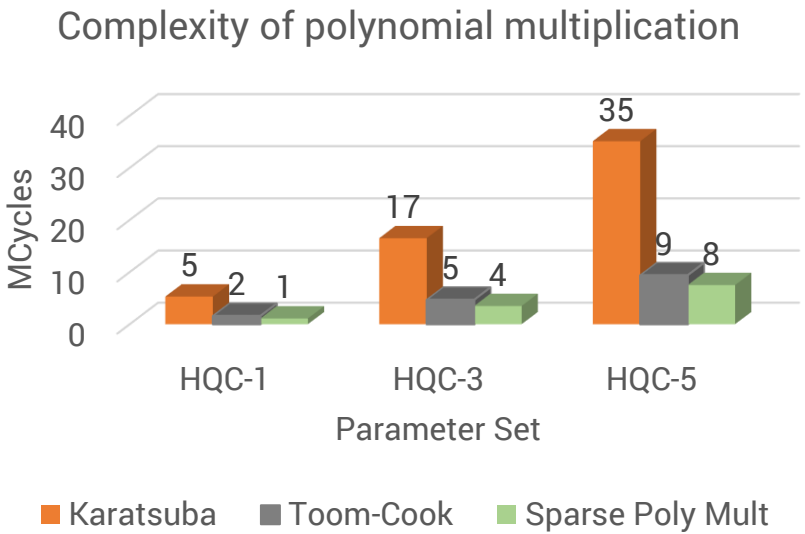
$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

$$B(x) = 0 + b_1x + 0 + \dots + b_mx^m$$

$$c(x) = \sum_{i=0}^{w-1} A(x) \ll \text{index}[i]$$

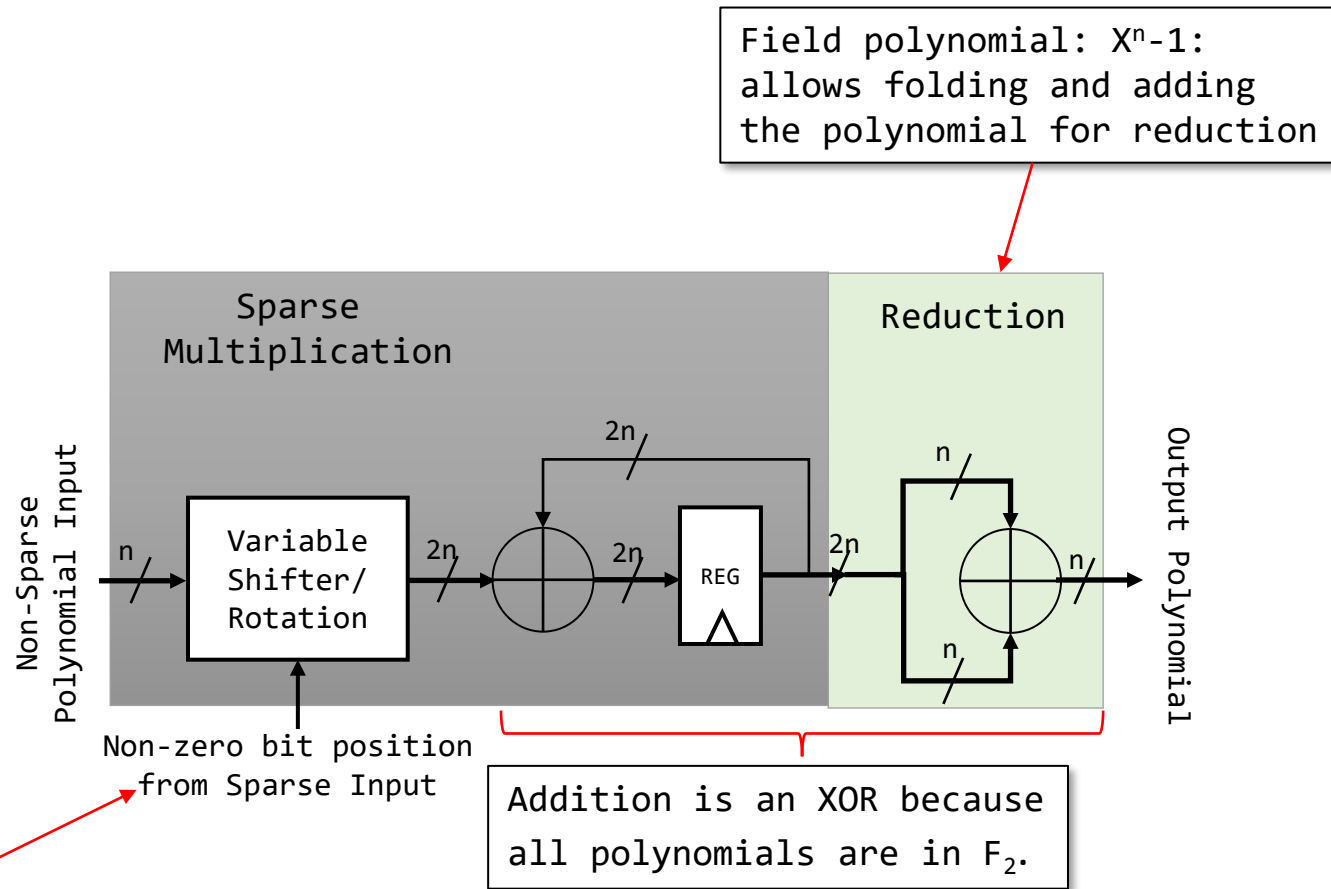


Parameter Set	n	w
HQC-1	17,669	66
HQC-3	35,851	100
HQC-5	57,637	131



Karatsuba = $O(n^{1.585})$
 Toom-Cook = $O(n^{1.465})$
 Sparse poly mult = $O(n.w)$

Sparse F_2 Polynomial Multiplication with Reduction



Indices of non-zero elements are used to shift non-sparse polynomial to imitate the multiplication.

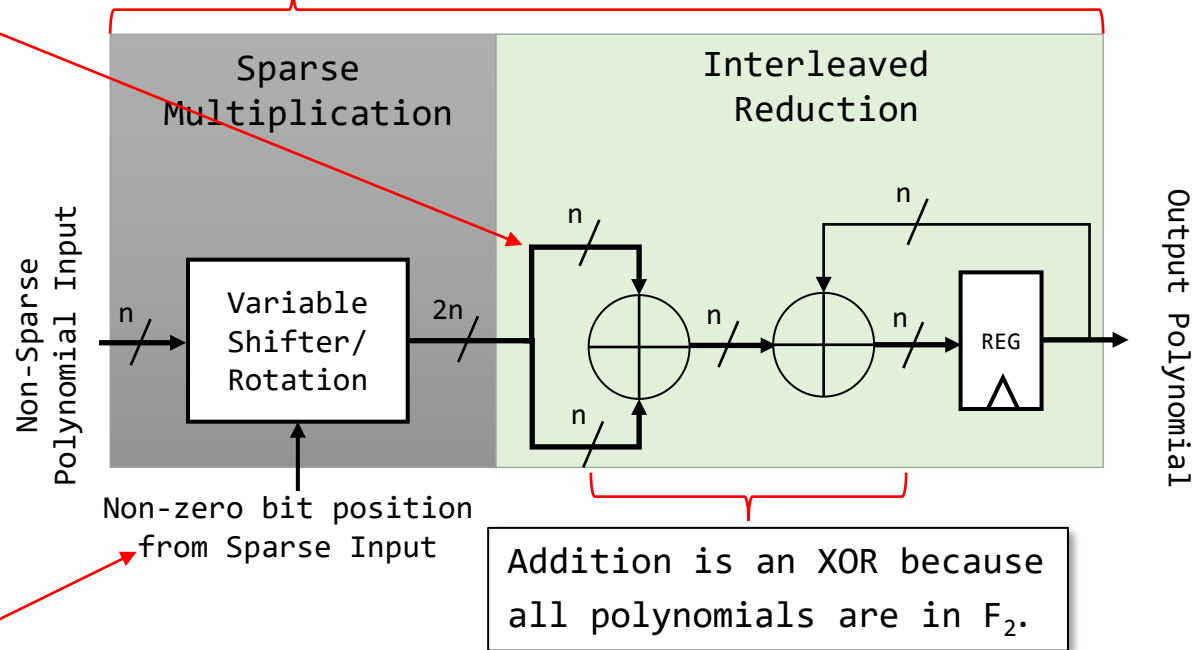
Addition is an XOR because all polynomials are in F_2 .

Sparse F_2 Polynomial Multiplication with Interleaved Reduction

Field polynomial: X^n-1 :
allows folding and adding
the polynomial for reduction

The multiplication and
the modular reduction is
interleaved.

- Reduction in area
- Reduction in memory utilization

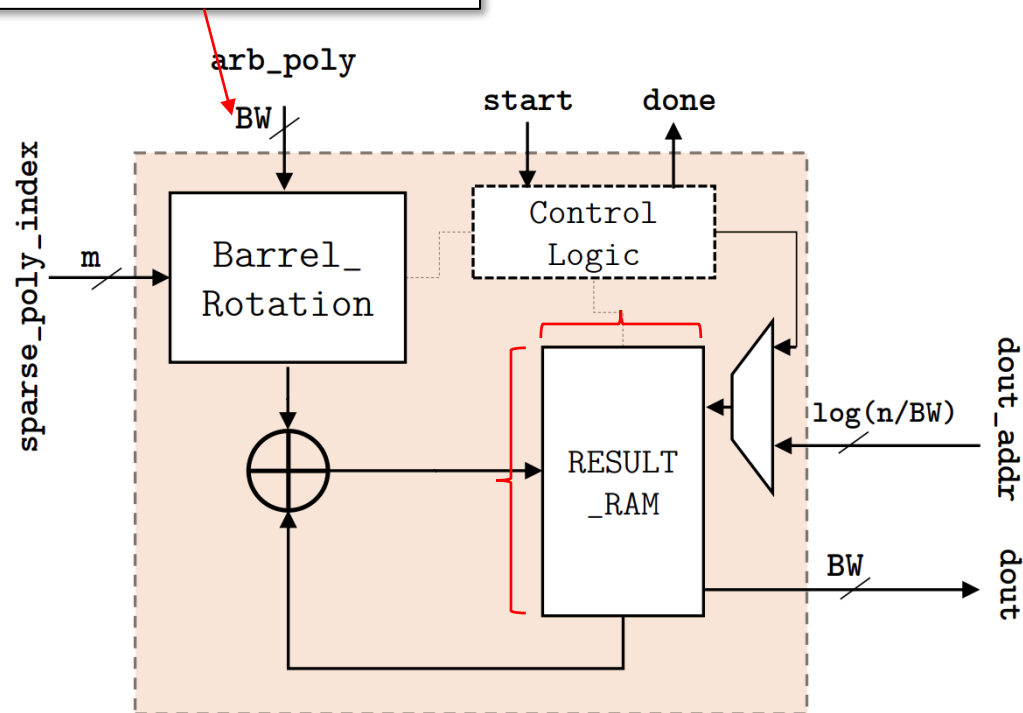


Indices of non-zero elements
are used to shift non-sparse
polynomial to imitate the
multiplication.

Addition is an XOR because
all polynomials are in F_2 .

Sparse F_2 Polynomial Multiplication Hardware Design

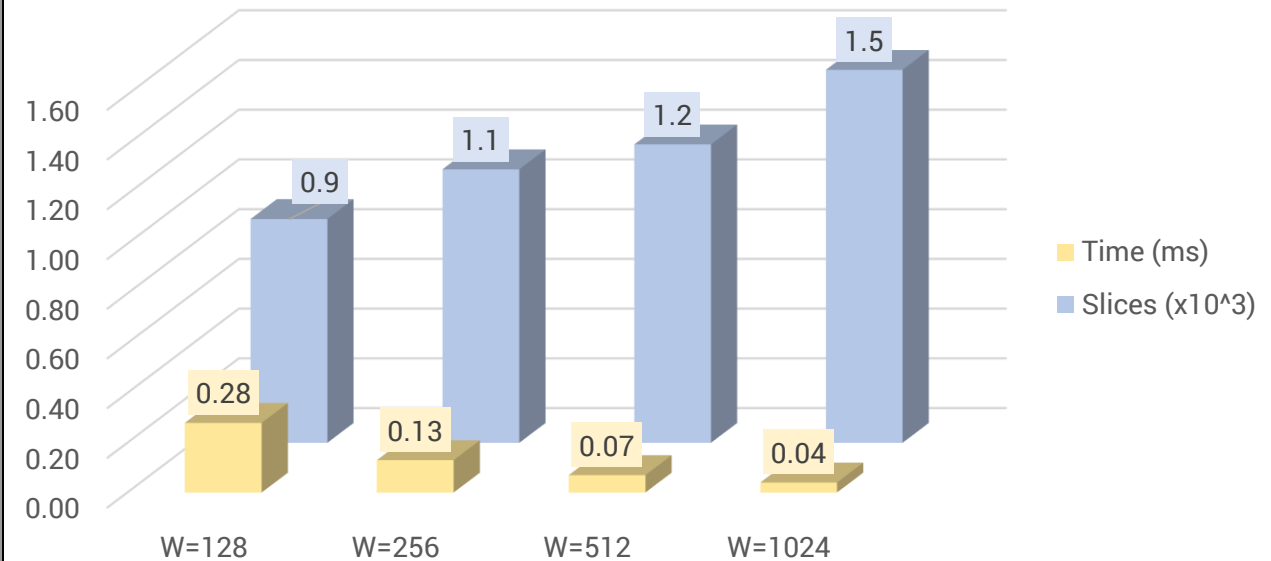
Performance parameter



Hardware design of sparse polynomial multiplication module

HQC-1

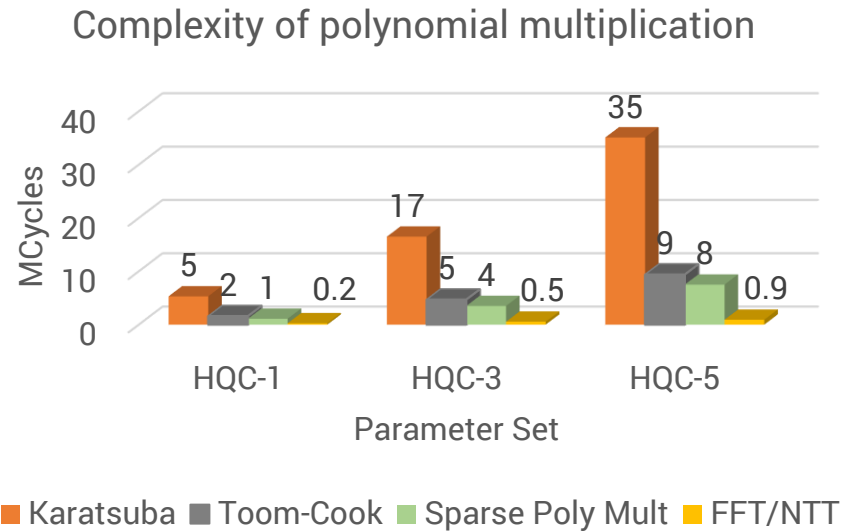
Time and Area Scaling



Parameter Set	n	w
HQC-1	17,669	66
HQC-3	35,851	100
HQC-5	57,637	131

What about NTT/FFT type multiplication for HQC?

Parameter Set	n	w
HQC-1	17,669	66
HQC-3	35,851	100
HQC-5	57,637	131



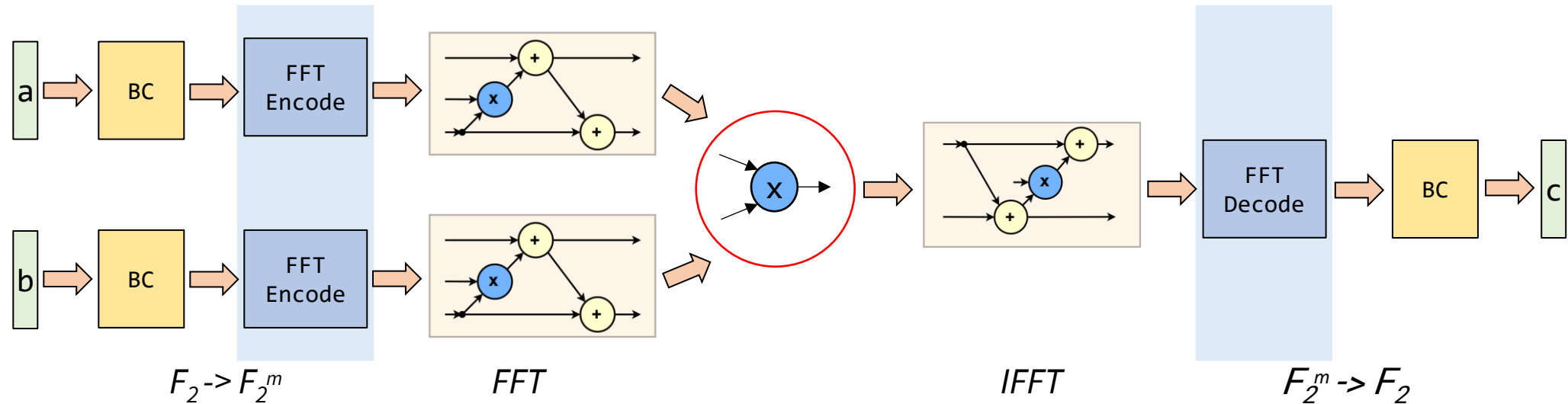
Karatsuba $= O(n^{1.585})$
Toom-Cook $= O(n^{1.465})$
Sparse poly mult $= O(n.w)$
FFT/NTT $= O(n.\log n)$

What about NTT/FFT type multiplication for HQC?

- Cannot directly apply NTT (Number Theoretic Transform) to binary field polynomial multiplication
 - Requires a finite field with special roots of unity
- Chen et. al [CCK21] proposed using **Frobenius Additive FFTs (FAFFT)**.

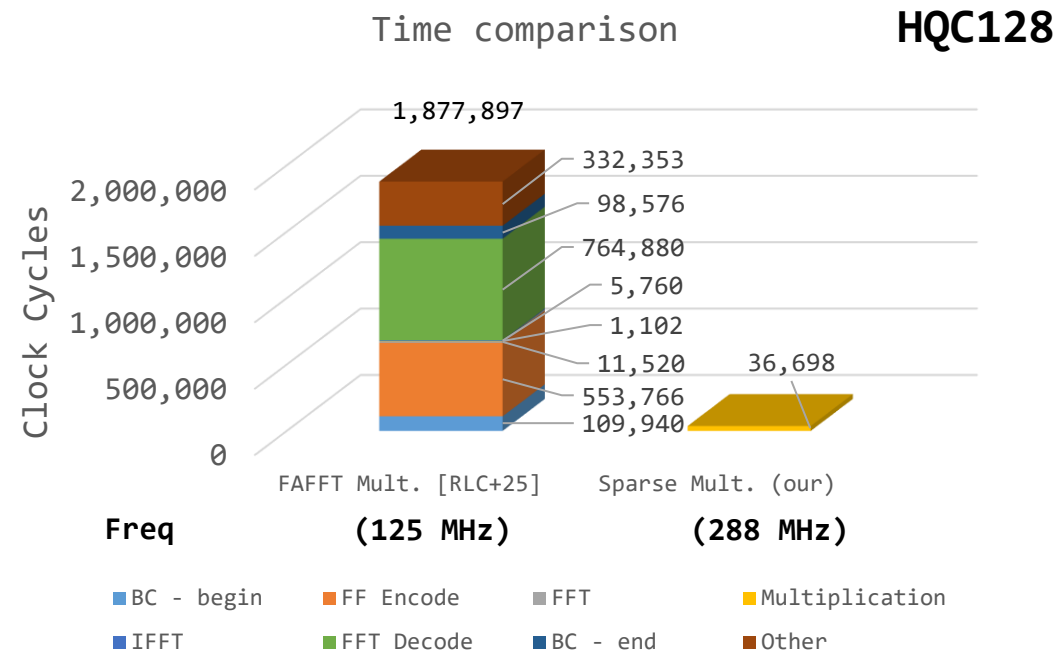
Frobenius Additive FFT (FAFFT)

- Convert to new polynomial basis – **Cantor Basis**.
- **FFT Encode** and **FFT Decode** are used to reduce the number of evaluations.

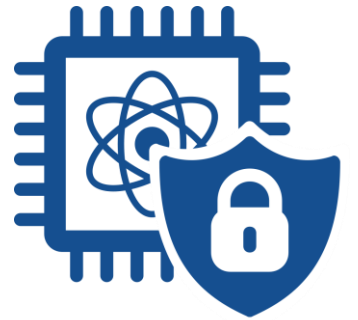


Performance FAFFT vs Sparse Multiplication

- Implemented in hardware by Ras et. al [RLC+25]
- Taking advantage of sparsity doesn't lead to non-constant time behavior on hardware.
 - Weight of the sparse polynomial is fixed.



Key Generation, Encapsulation, and Decapsulation



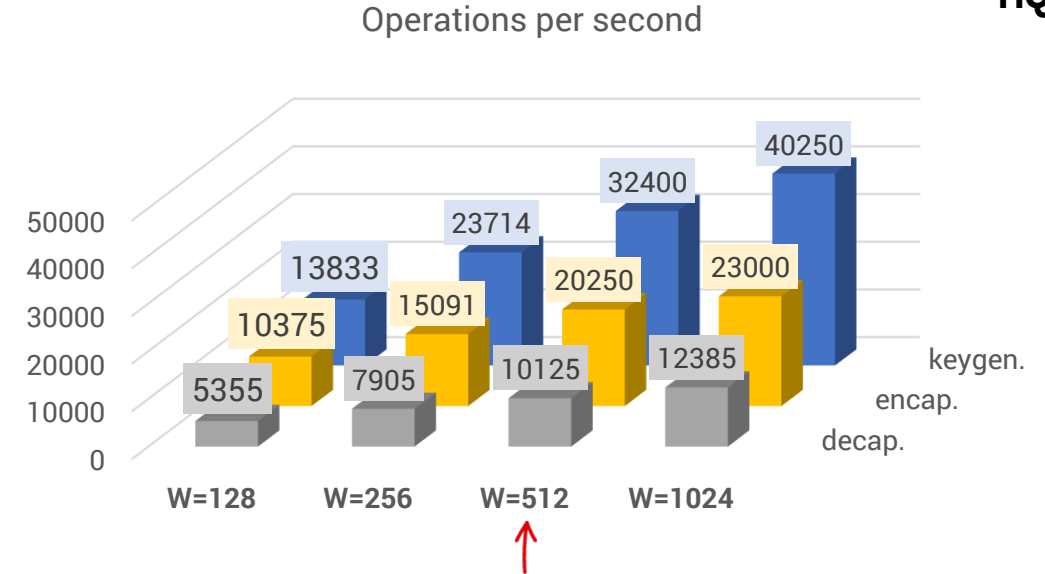
Joint Design – Sweep

FPGA: AMD Artix 7

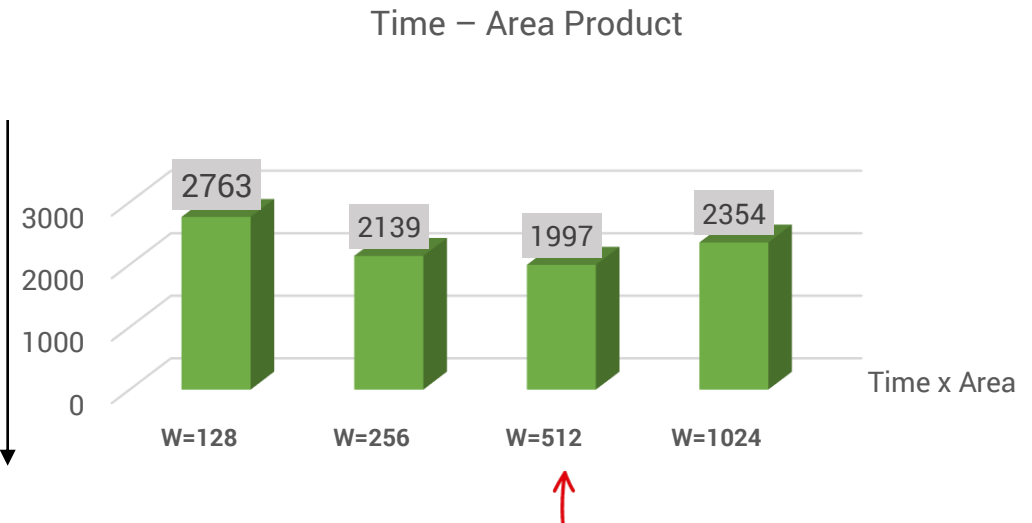
HQC-1

- Joint Design – combines Keygen, Encap, and Decap into one.
- Shared module among different primitives.
 - SHAKE256
 - Polynomial Multiplication
 - Encapsulation
 - Polynomial Addition
- W – performance parameter
- Time-Area product
 - $\text{Time} * \max\{\text{LUT}/4 + 128 * \text{BRAM} + 100 * \text{DSP}, \text{FF}/8\}$

Higher is better



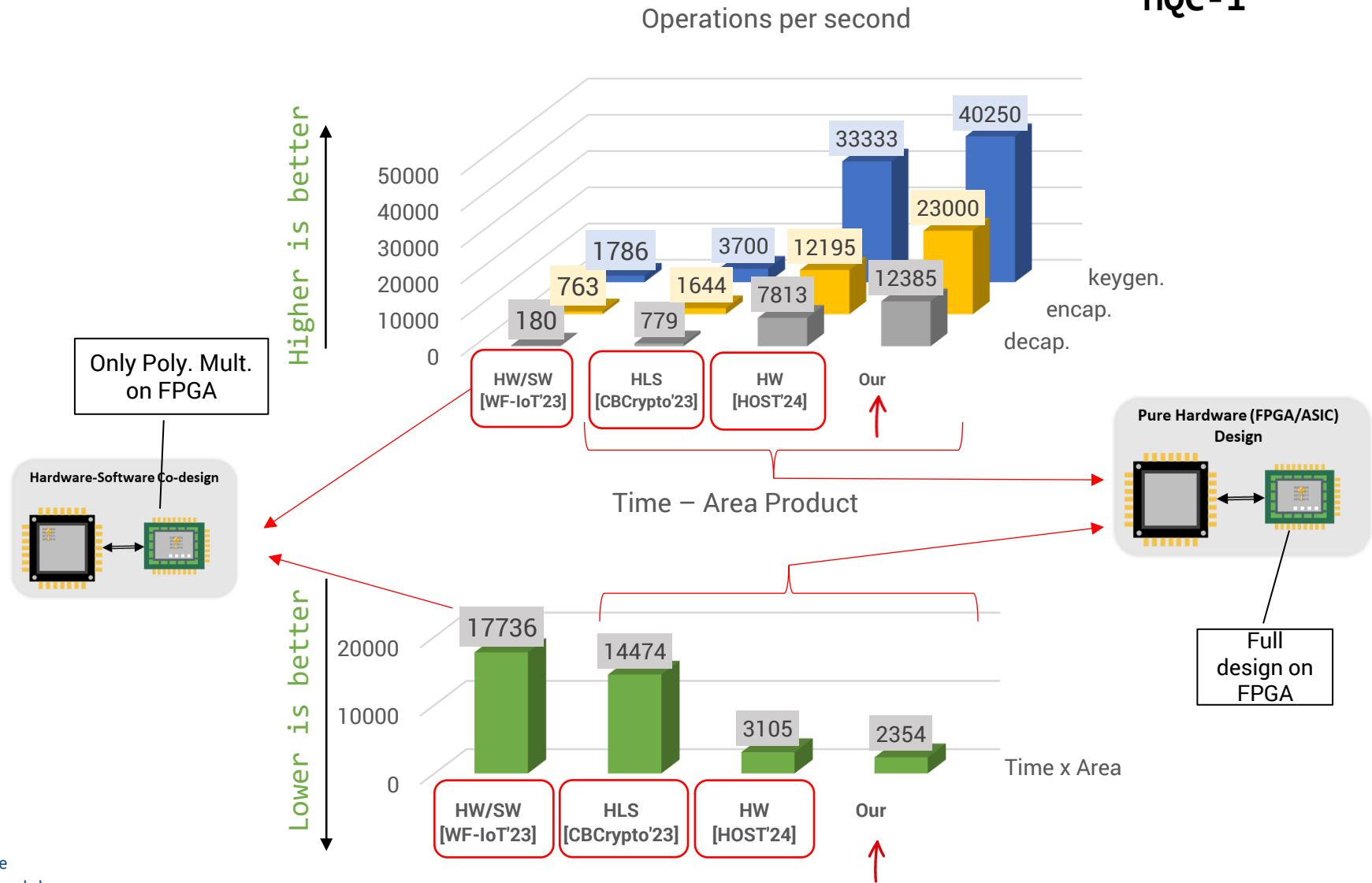
Lower is better



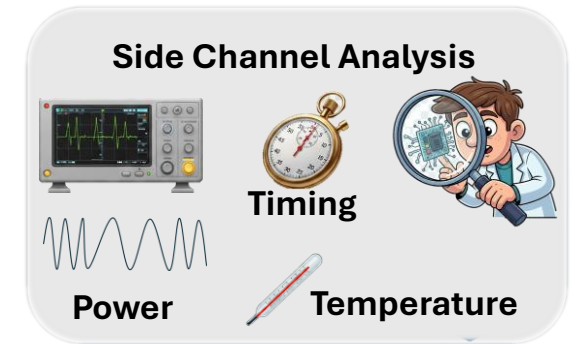
Joint Design – Comparison with other HQC Hardware Designs

FPGA: AMD Artix 7

HQC-1

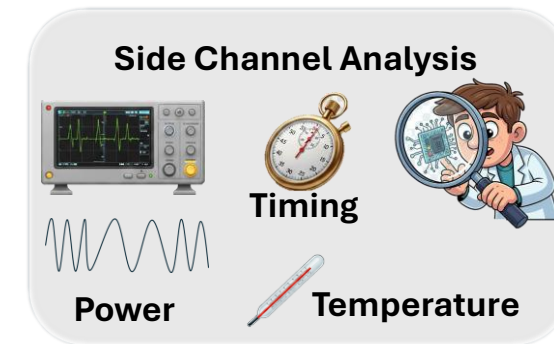


Side-Channel Evaluation

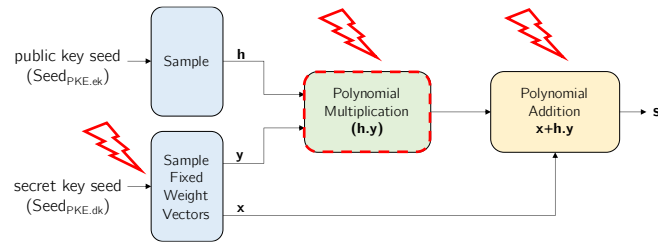


Side Channel Evaluation

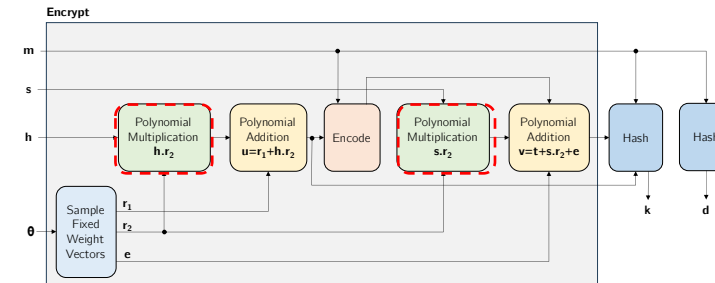
- **Side-channel** attacks are based on the **implementation** of a system
- It is not a brute-force attack or a theoretical weakness based attack
- Observe **unintentional features** such as timing, power, etc



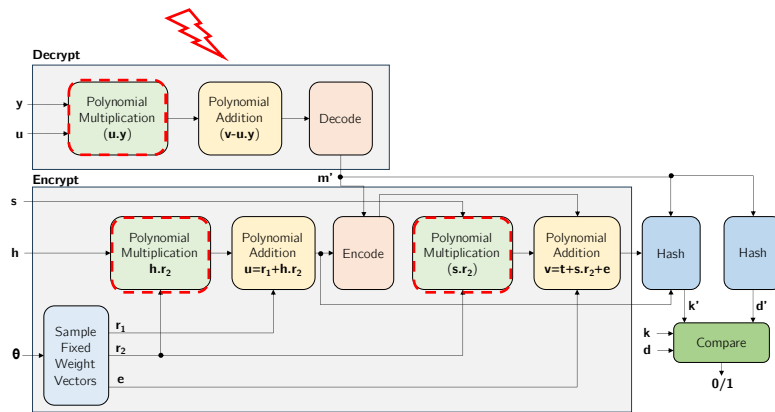
HQC Primitives



Key Generation



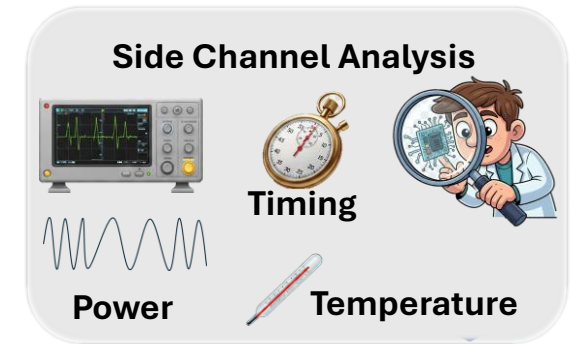
Encapsulation



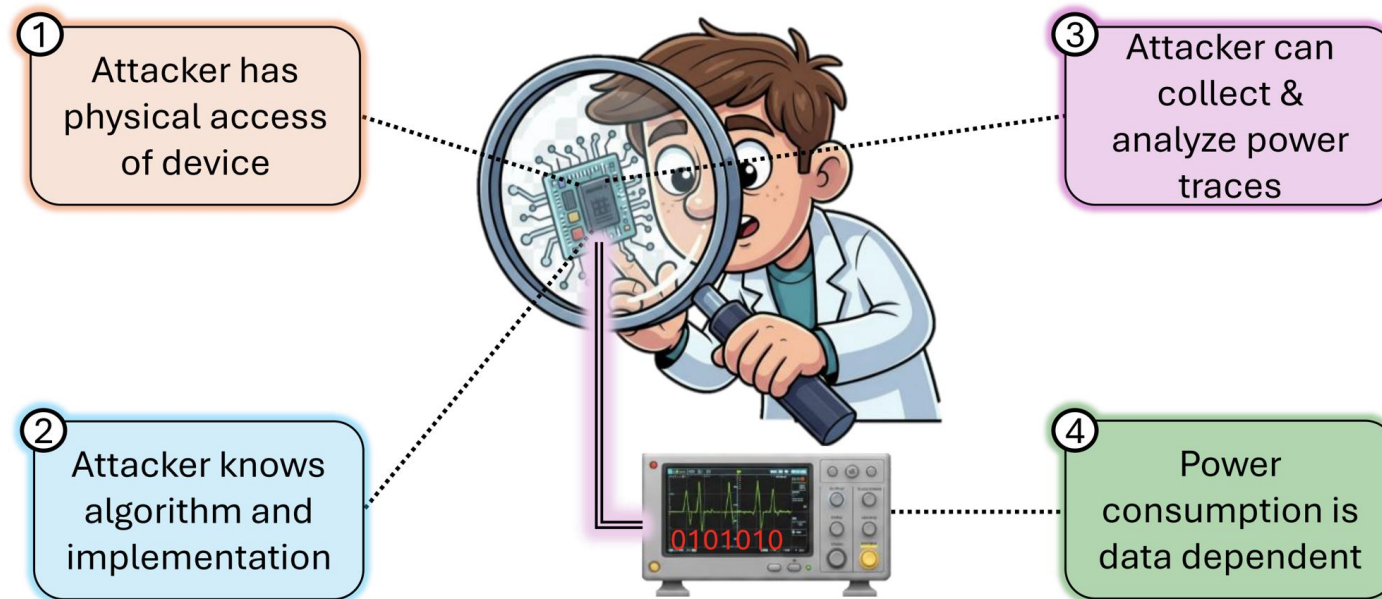
Decapsulation

- Performance Bottleneck
- ⚡ Side-channel attack target

Power Side-Channel Evaluation

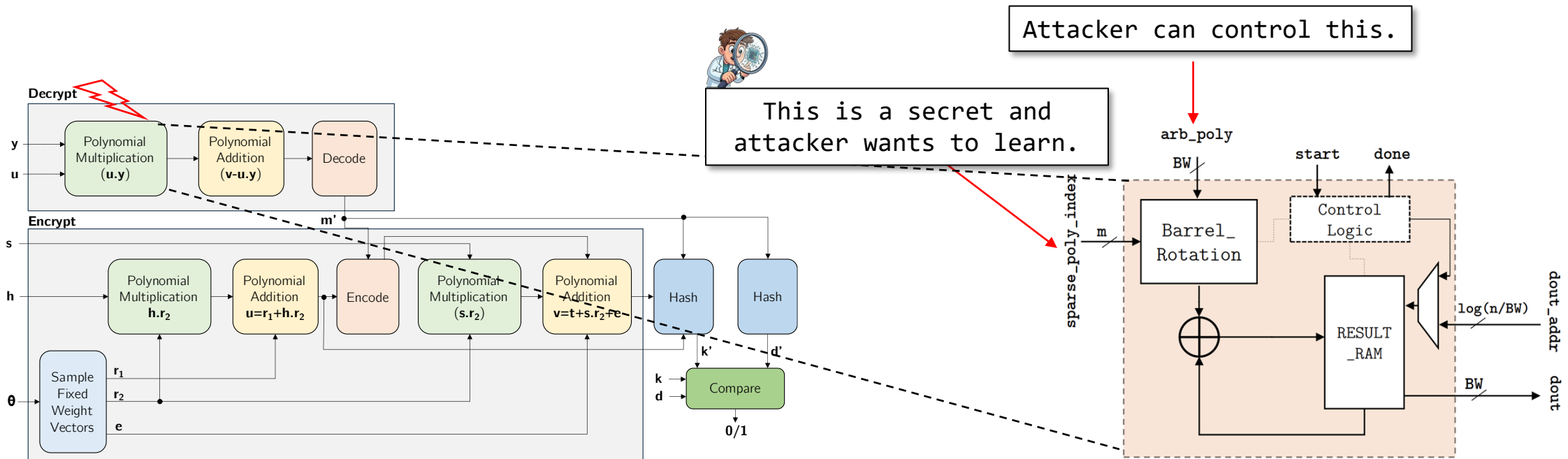


Threat Model

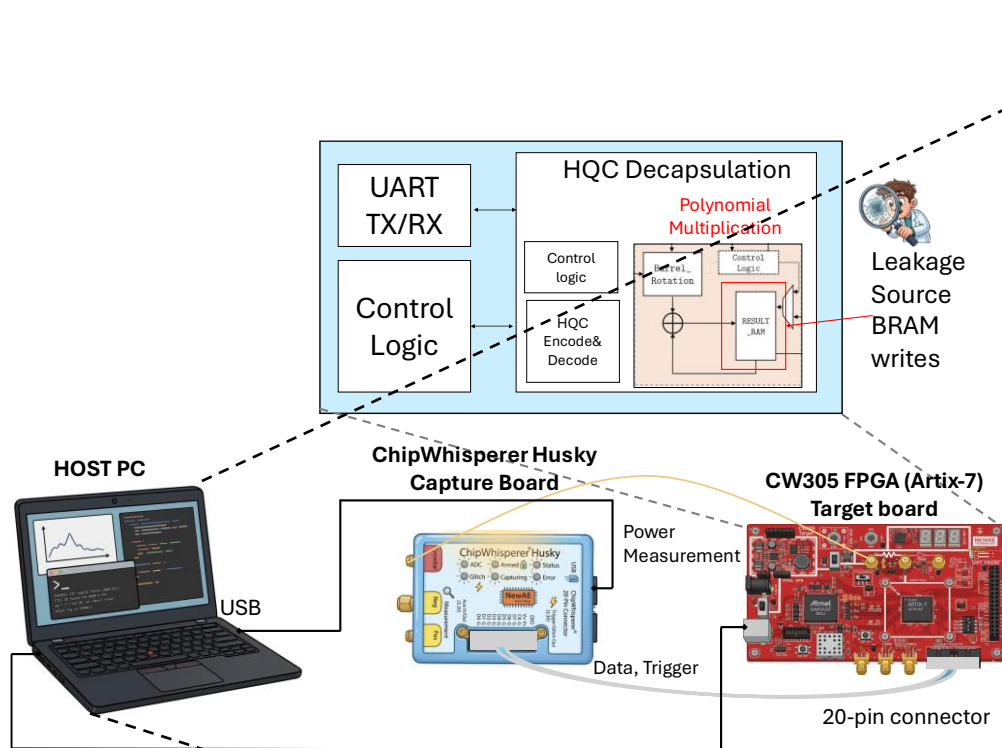


Interesting place to observe for side-channels

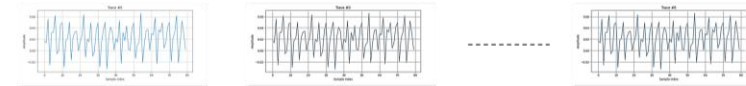
- Attacking the variable rotation in the sparse polynomial multiplication in the decapsulation operation



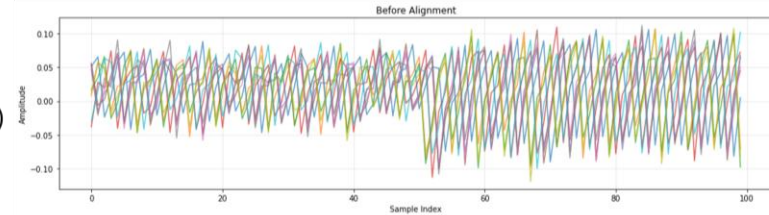
Attack Setup and Working



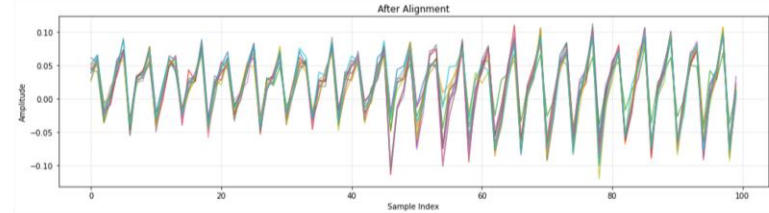
Stage 1:
Trace Acquisition



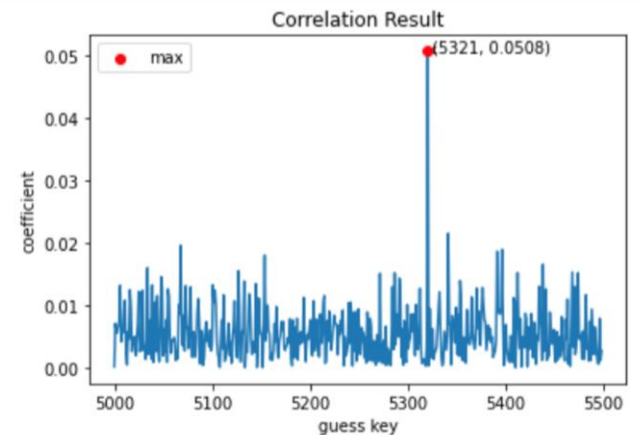
Stage 2:
Preprocessing
(before alignment)



After Alignment



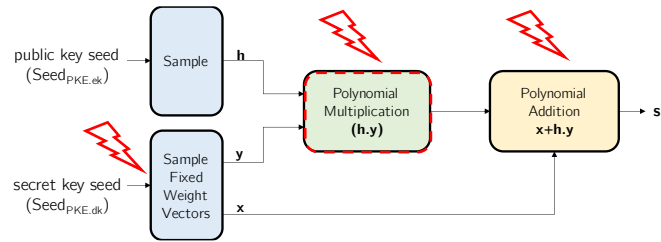
Stage 3: CPA Key
Recovery



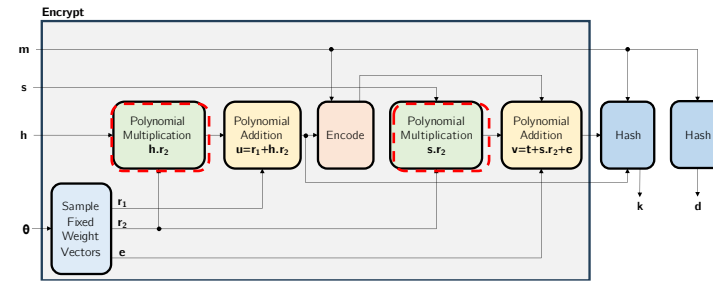
What can we do about it?

- Could be thwarted using robust implementation with **side-channel countermeasures**
- Countermeasures:
 - **Masking**
 - **Shuffling** sparse indices for each run of decapsulation
 - Random **shifting** for non-sparse polynomial before multiplication

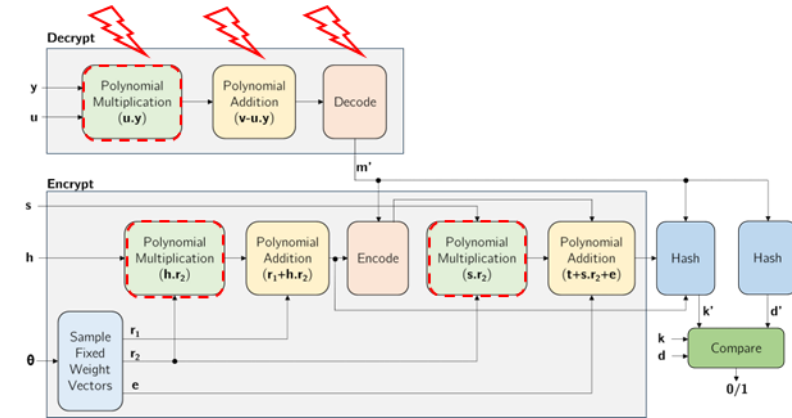
Summary



Key Generation



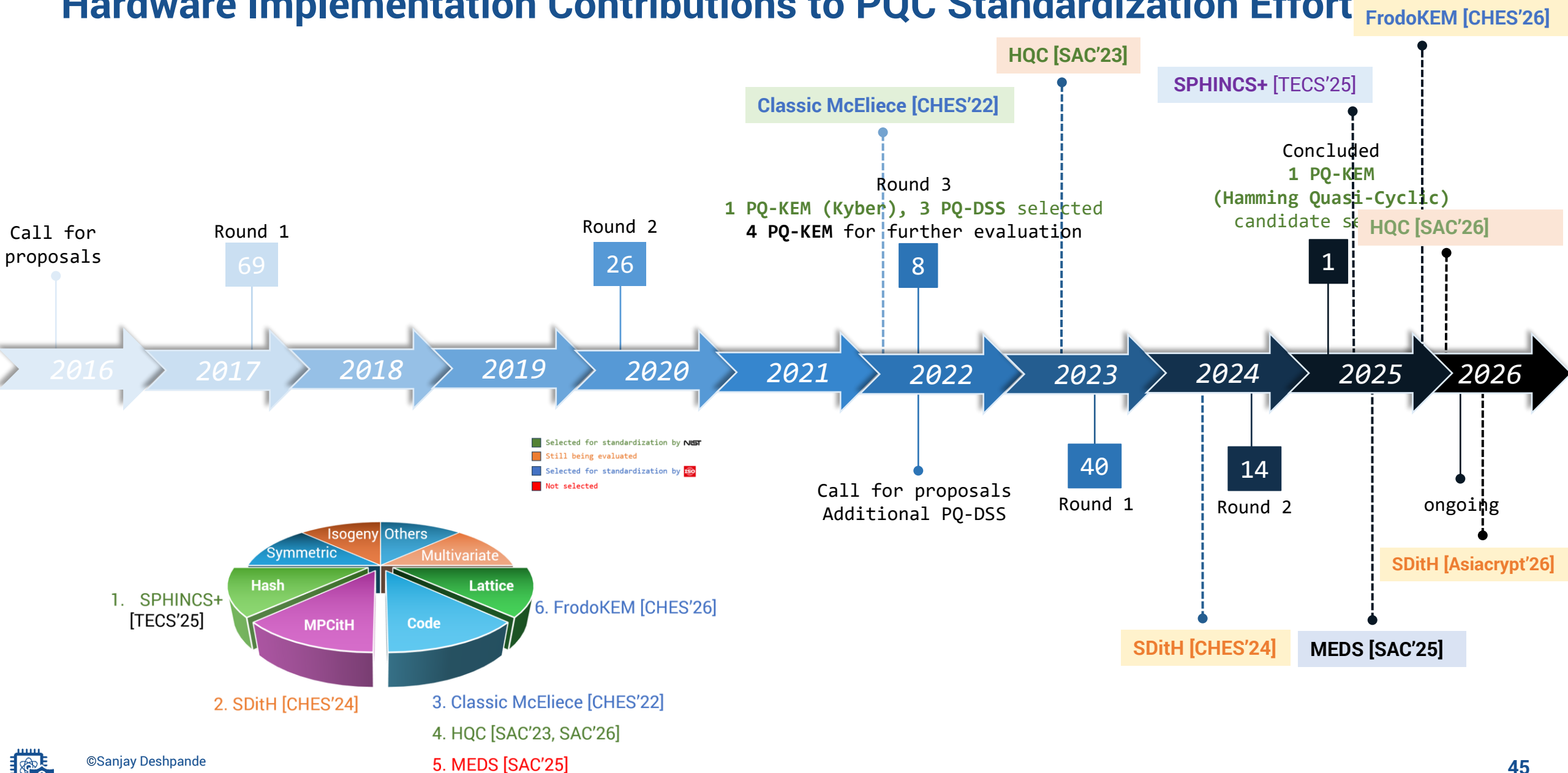
Encapsulation



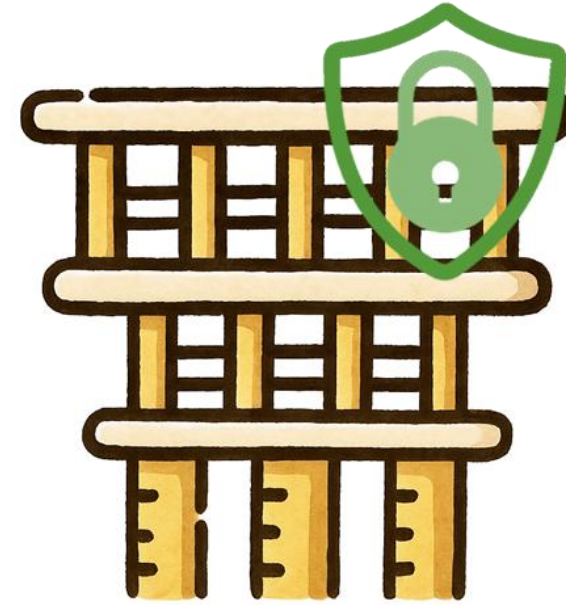
Decapsulation

- Design of performance and security critical aspects of HQC implementation

Hardware Implementation Contributions to PQC Standardization Effort

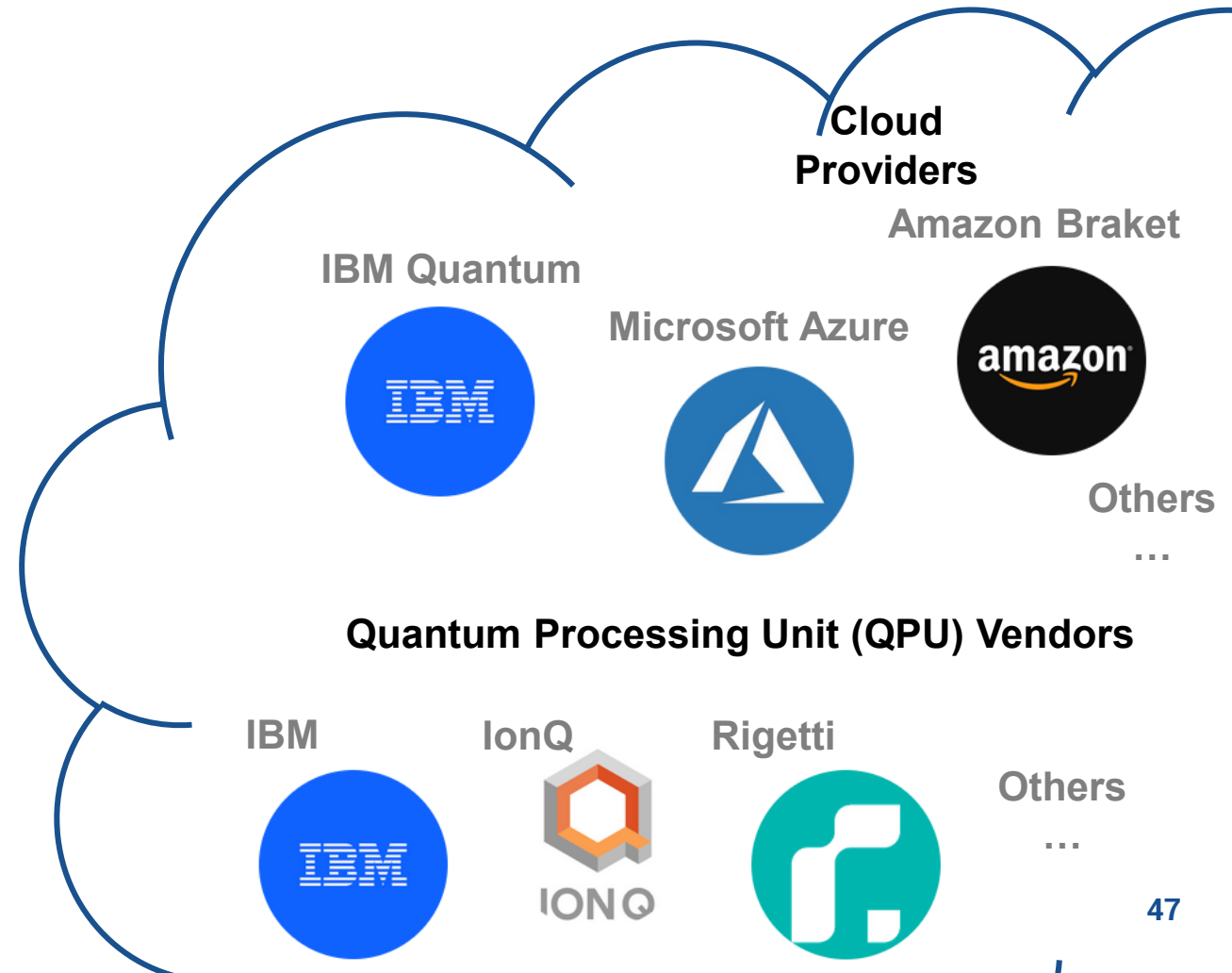


Quantum Computer Cybersecurity



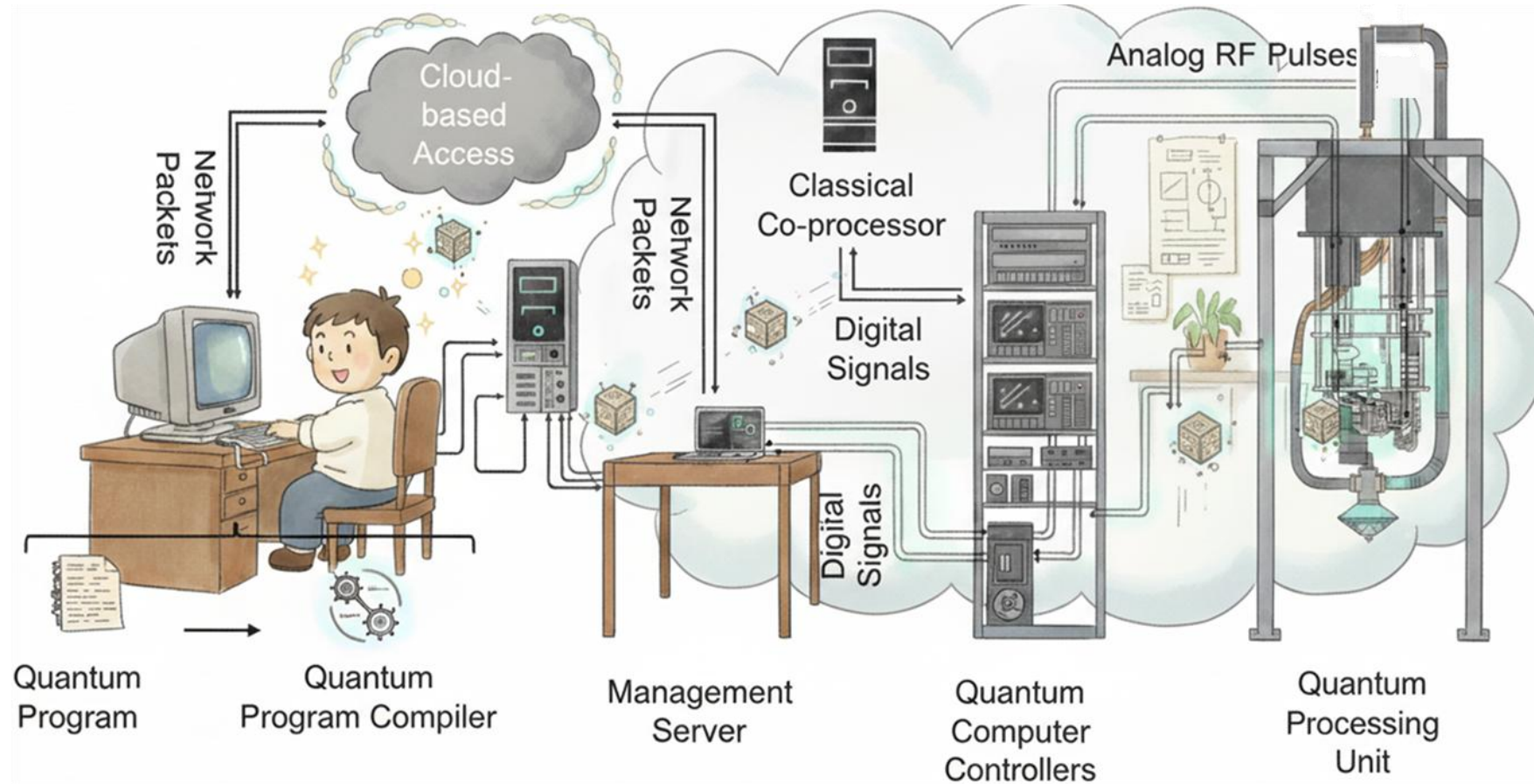
Cloud-based Access to Quantum Computers

- Quantum Computers are rapidly being deployed as cloud-based accelerators today
 - Many companies are pushing for Quantum Computer as a Service (QCaaS) deployments
- But there are no security considerations in QCaaS today
 - Possibly malicious users can now access quantum computing hardware remotely
 - Can attack other users, or try to reverse engineer the infrastructure

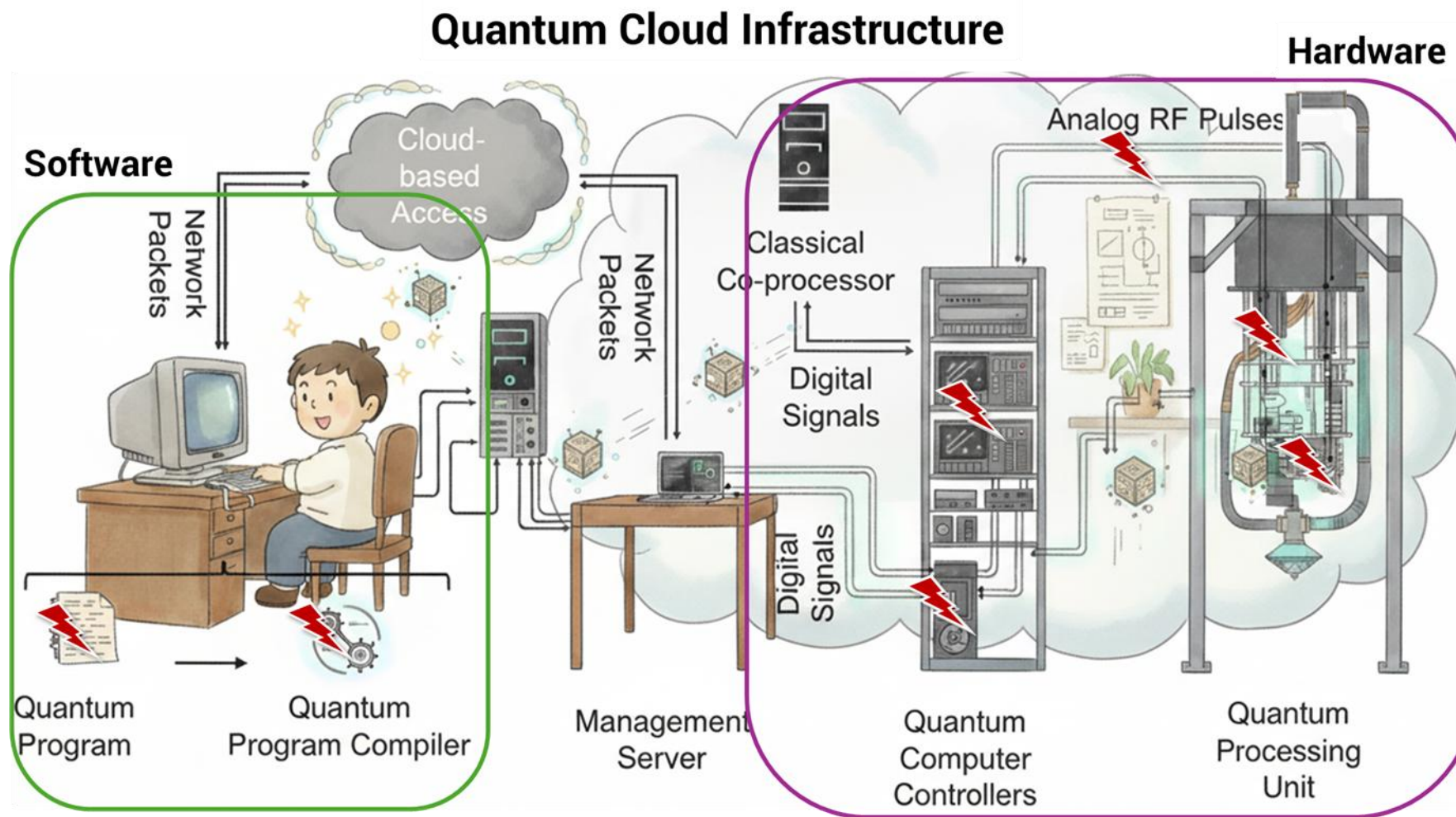


Workflow of Cloud-Based Quantum Computers

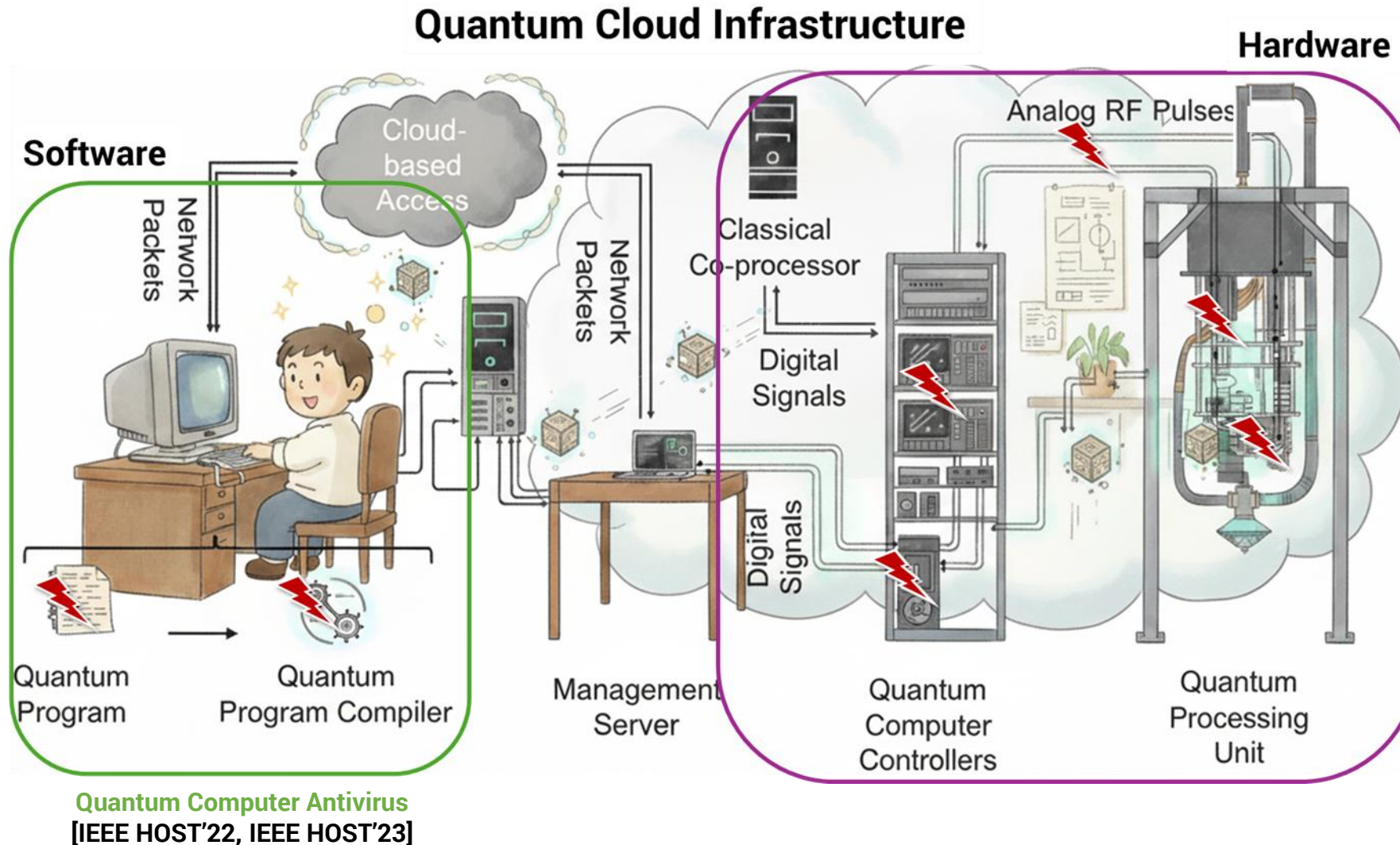
Quantum Cloud Infrastructure



Potential Threat Landscape



Some of Our Work on Quantum Computer Cybersecurity (QCC)



A Quantum Computer Trusted Execution environment [IEEE CAL'23]

Dynamic Pulse Switching for Protection of Quantum Computation on Untrusted Clouds [IEEE HOST'24]

Protecting quantum computers with a trusted controller [IEEE QCE'24]

Trusted execution environments for quantum computers [Front. Comp. Sci'25]

CHEQ: Towards Enabling Circuit Integrity Checking in Quantum Controllers [GLSVLSI'25]

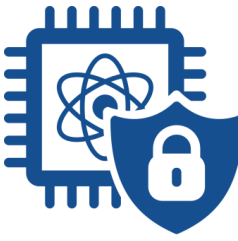
QCCS 2026

4th Annual Quantum Computer Cybersecurity Symposium

To be held November 11 to 13, 2026 at The Guild Lounge in Scott Hall at Northwestern University

<https://caslab.io/events/qccs/>





Thank you!