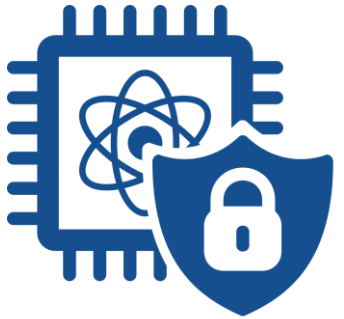
Performance Analysis of Parameterizable HQC Hardware Architecture

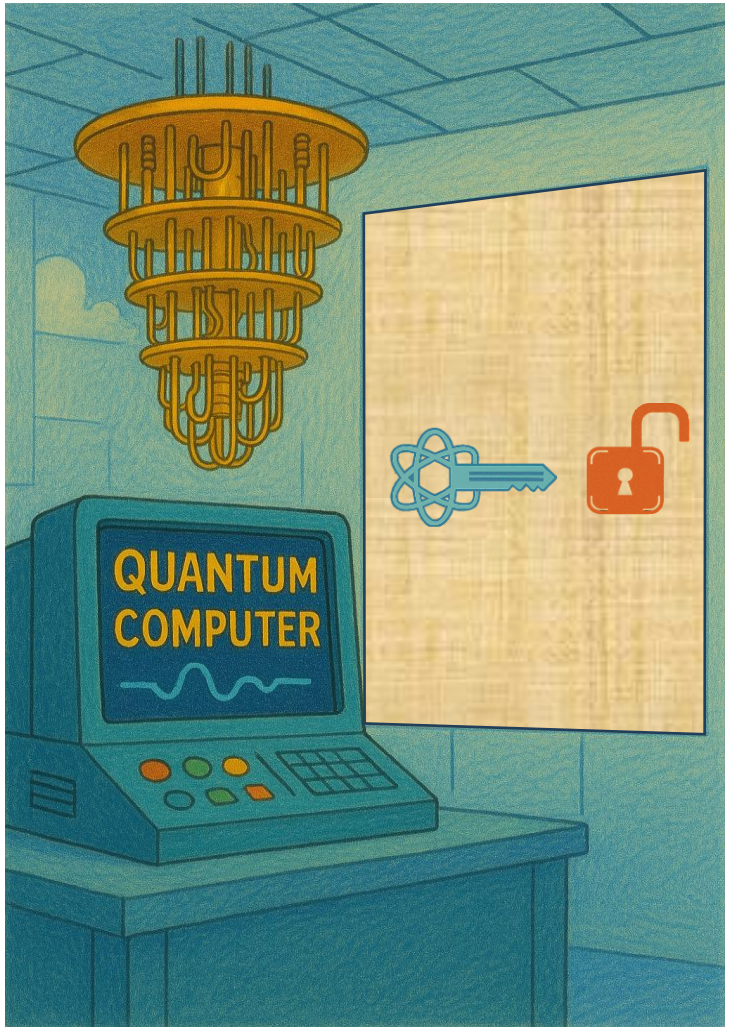


Nishant Pandey, **Sanjay Deshpande**, Dixit Dutt Bohra, Debapriya Basu Roy, Dip Sankar Banerjee, and Jakub Szefer

Computer Architecture and Security Lab (CASLAB)
https://caslab.io
Northwestern

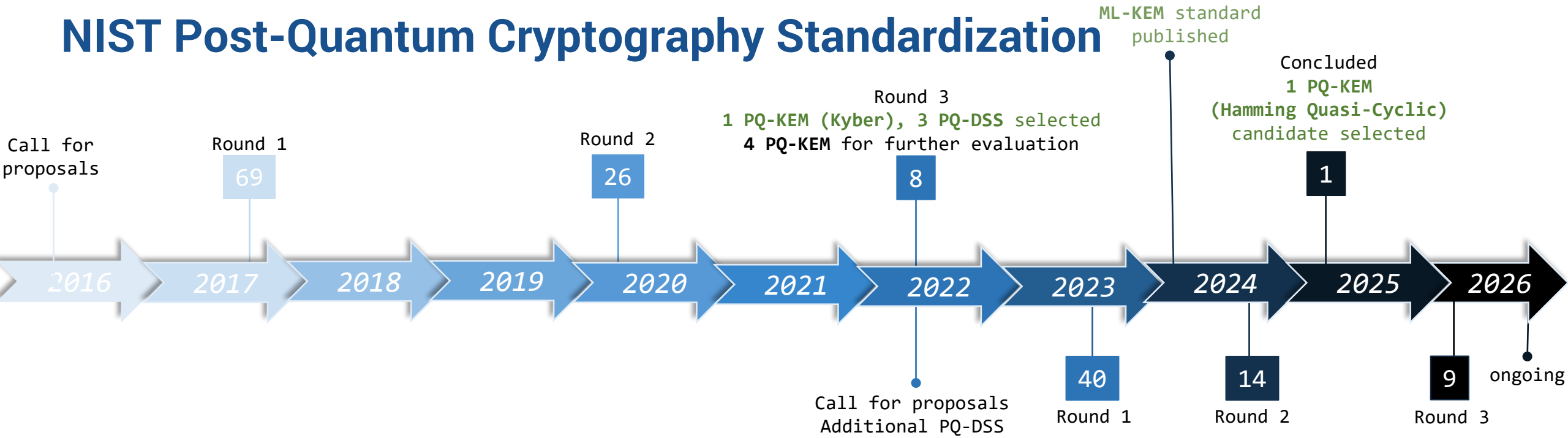
Quantum Threat and Post-Quantum Cryptography

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



NIST ISO IETF

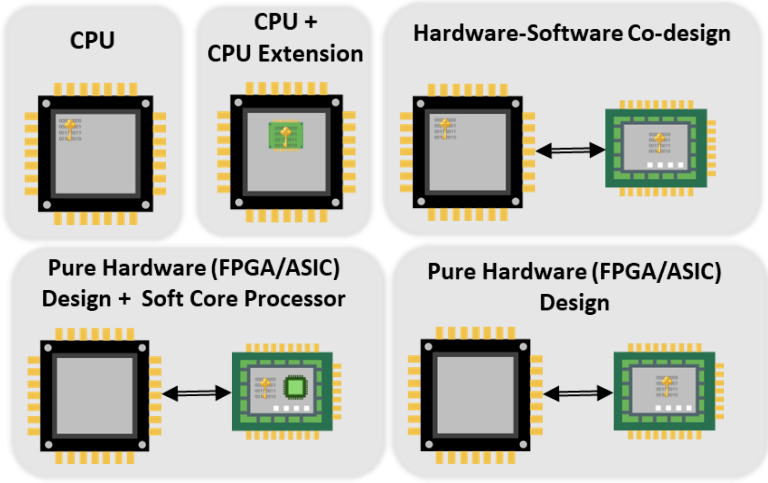
NIST Post-Quantum Cryptography Standardization



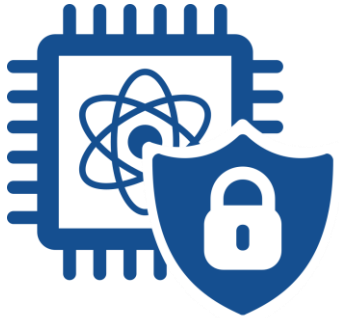
- Submitted candidates belong to two categories
  - **PQ-KEM: Post-Quantum Key Encapsulation Mechanism**
  - PQ-DSS: Post-Quantum Digital Signature Scheme
- Evaluation Criteria
  - Security
  - **Hardware and Software implementation efficiency and security**
  - Cost (key size, signature size, etc.)

Secure and Efficient Implementations of Cryptographic Algorithms

- Target platforms:
  - CPU
  - Microcontrollers
  - **Field-Programmable Gate Array (FPGA) : Artix 7**
  - Application Specific Integrated Circuit (ASIC)



Hamming Quasi-Cyclic (HQC)



Key Encapsulation Mechanism

Alice's Public Key

$(A_{PK}, A_{SK}) \leftarrow KeyGen(seed)$

Alice's Secret Key

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

Decapsulated Key

A_{PK}

C

Encapsulated Key

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

Ciphertext

Secure Channel

Symmetric Encryption

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

n - degree of the polynomial
w - weight
keypair – (ek,dk)
ct - ciphertext

Key Generation

HQC-KEM.Keygen()

Input: None.

Output: Encapsulation key $\mathsf{ek}_{\mathsf{KEM}}$, decapsulation key $\mathsf{dk}_{\mathsf{KEM}}$.

▷ Sample $\mathsf{seed}_{\mathsf{KEM}}$

1. $\mathsf{seed}_{\mathsf{KEM}} \leftarrow\$ \mathcal{B}^{|\mathsf{seed}|}$

▷ Compute $\mathsf{seed}_{\mathsf{PKE}}$ and randomness σ

2. $\mathsf{ctx}_{\mathsf{KEM}} \leftarrow \mathsf{XOF.Init}(\mathsf{seed}_{\mathsf{KEM}})$
3. $(\mathsf{ctx}_{\mathsf{KEM}}, \mathsf{seed}_{\mathsf{PKE}}) \leftarrow \mathsf{XOF.GetBytes}(\mathsf{ctx}_{\mathsf{KEM}}, |\mathsf{seed}|)$
4. $(\mathsf{ctx}_{\mathsf{KEM}}, \sigma) \leftarrow \mathsf{XOF.GetBytes}(\mathsf{ctx}_{\mathsf{KEM}}, |k|)$

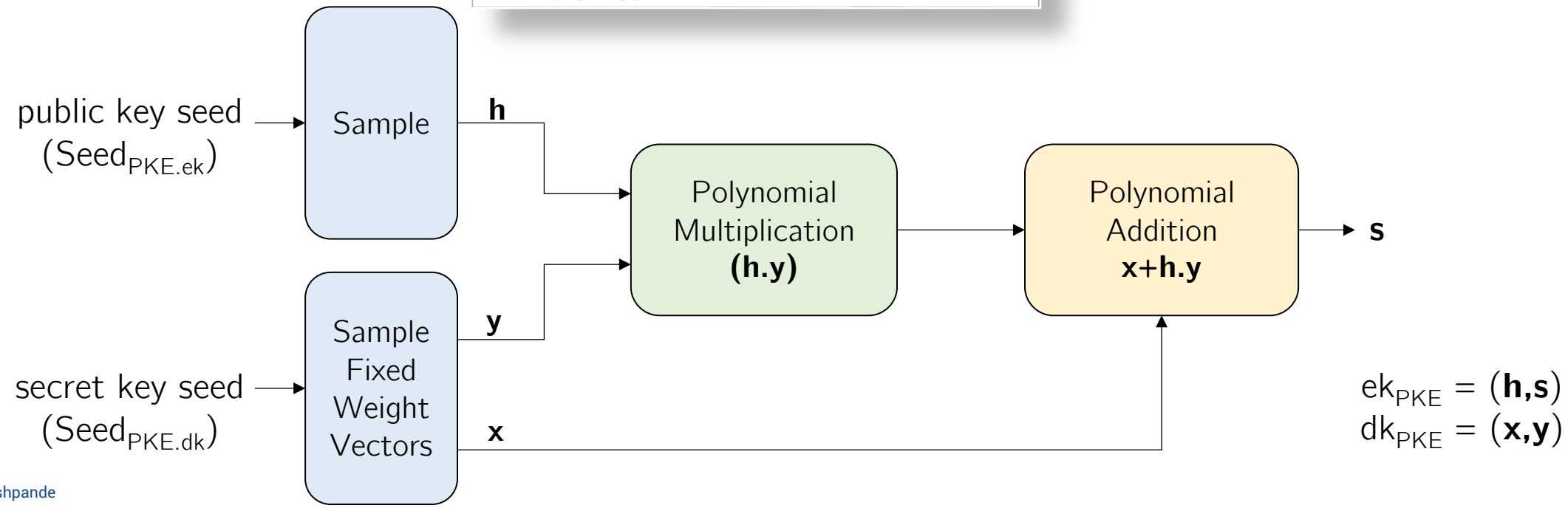
▷ Compute HQC-PKE keypair

5. $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow \mathsf{HQC\text{-}PKE.Keygen}(\mathsf{seed}_{\mathsf{PKE}})$

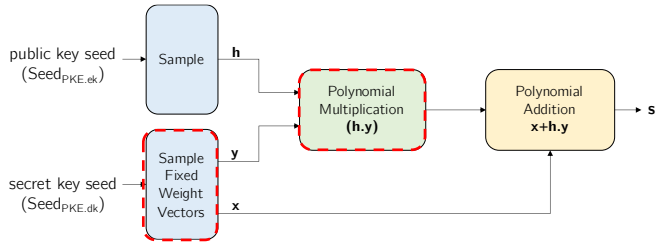
▷ Compute HQC-KEM keypair

6. $\mathsf{ek}_{\mathsf{KEM}} \leftarrow \mathsf{ek}_{\mathsf{PKE}}$
7. $\mathsf{dk}_{\mathsf{KEM}} \leftarrow (\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{PKE}}, \sigma, \mathsf{seed}_{\mathsf{KEM}})$

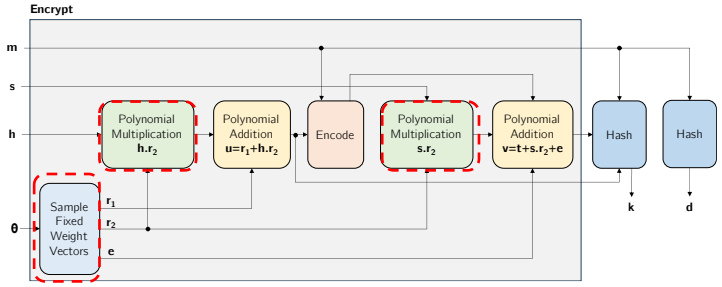
8. **return** $(\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{KEM}})$



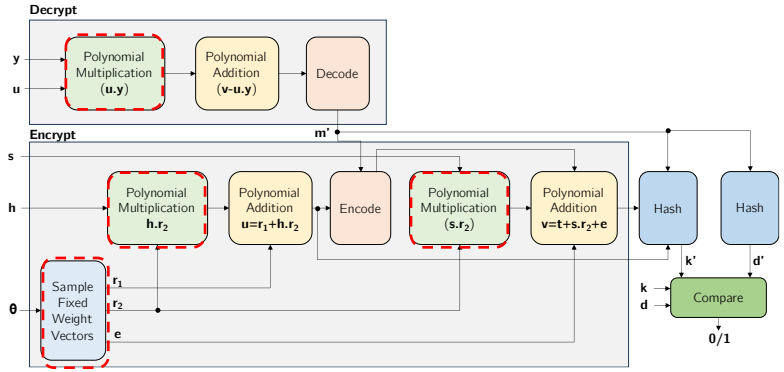
HQC Primitives



Key Generation



Encapsulation

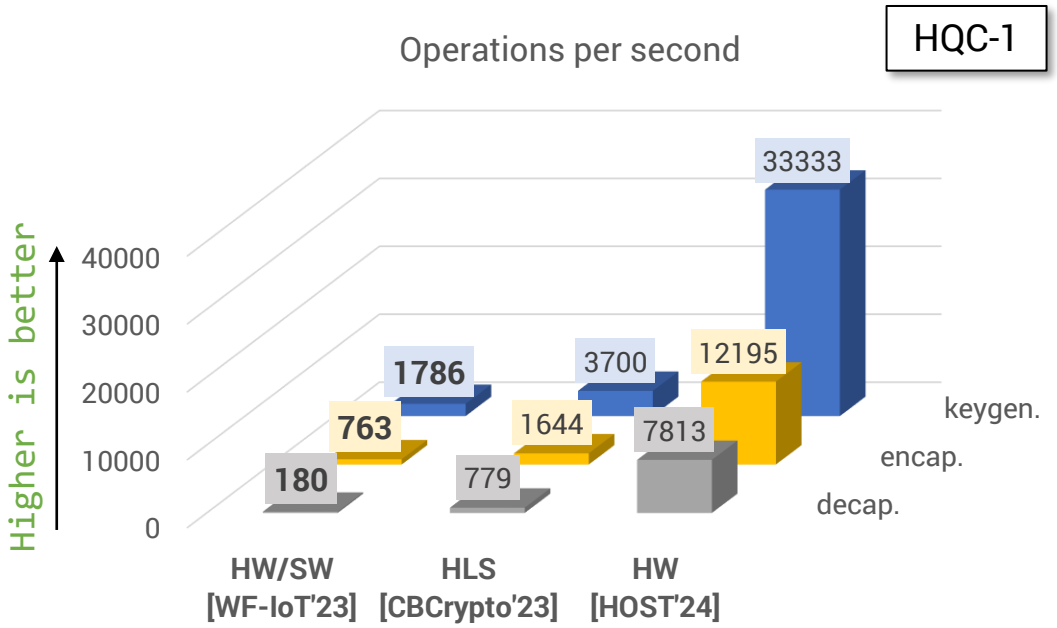


Decapsulation

Performance Bottleneck.
95-96% across different parameter sets

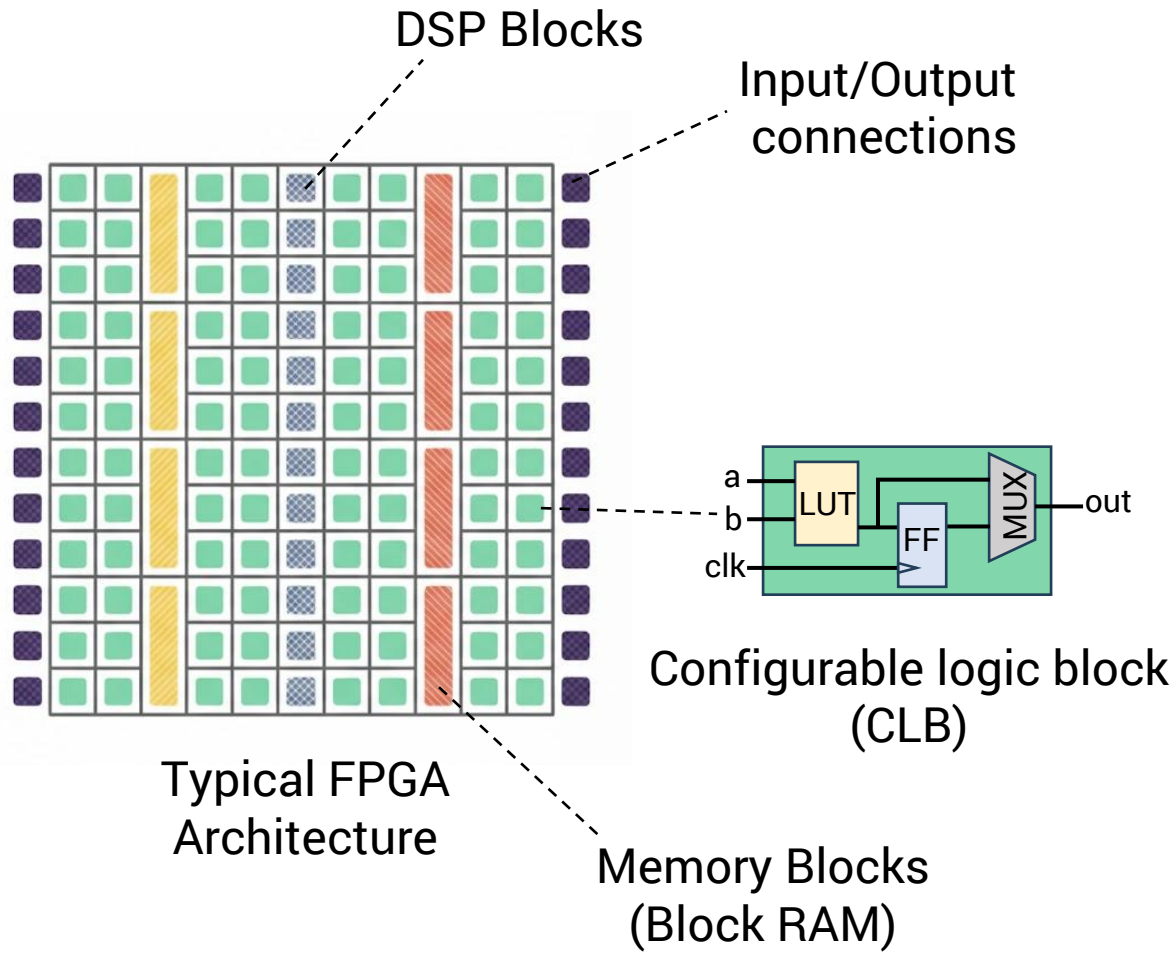
Limitations of Existing Work

- Existing work includes
  - Full hardware [SAC'23, HOST'24]
  - High-level synthesis [CBCrypto'23]
  - Hardware-Software codesign [WF-IoT'23]



FPGA Implementation Performance and Evaluation Metrics

- Time
- Area
- Area-Time product
  - Time * {max{LUT/4, FF/8} + 128 ∗ BRAM + 100 ∗ DSP}

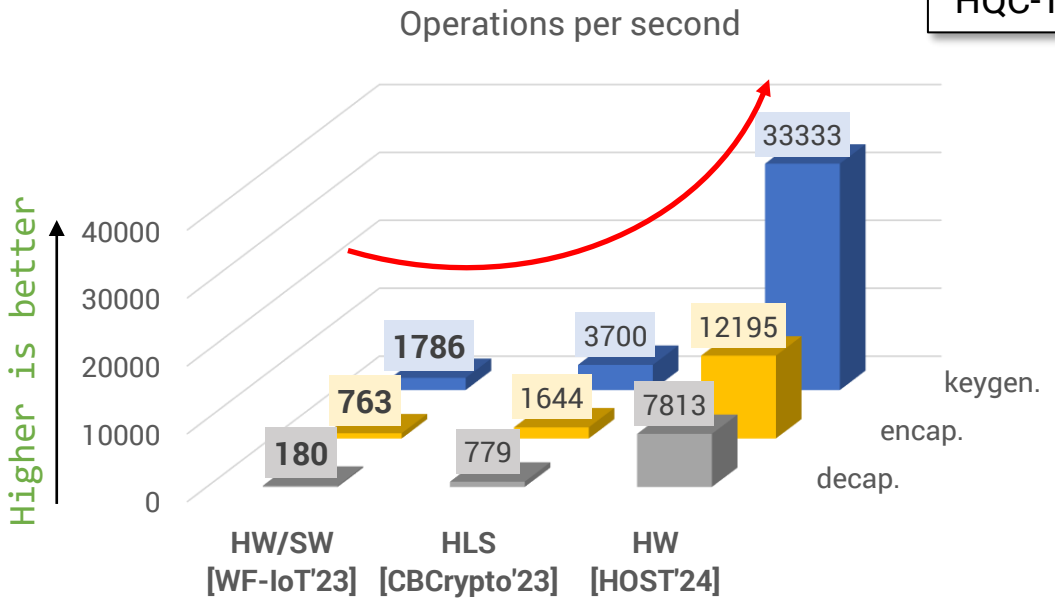


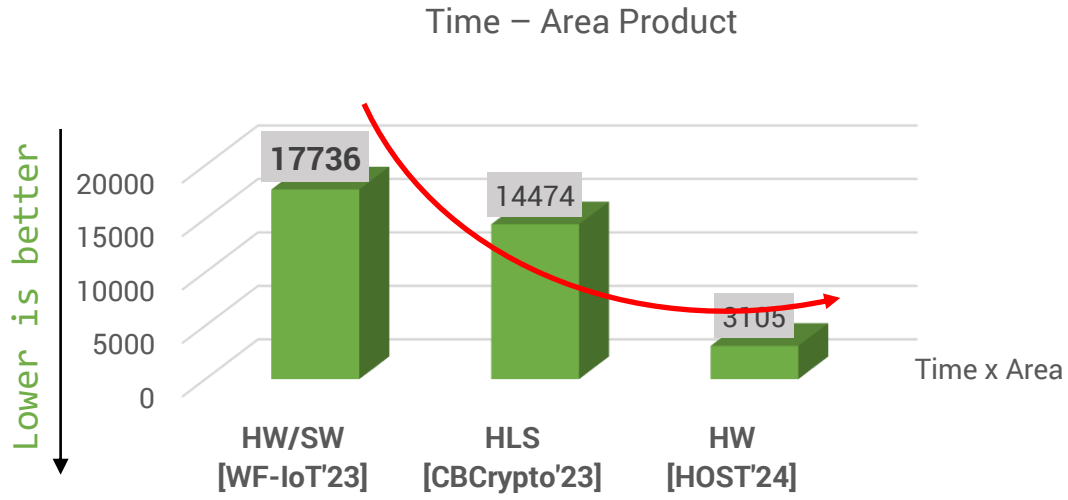
Limitations of Existing Work

- Existing work includes
  - Full hardware [SAC'23, HOST'24]
  - Hardware-Software codesign [CBCrypto'23]
  - High-level synthesis [WF-IoT'23]

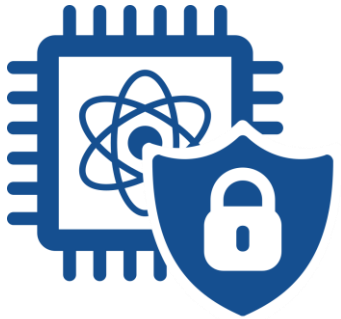
- **Previous implementations lack**
  - **Tweakable performance parameters**
  - **Support for new parameter sets**

HQC-1



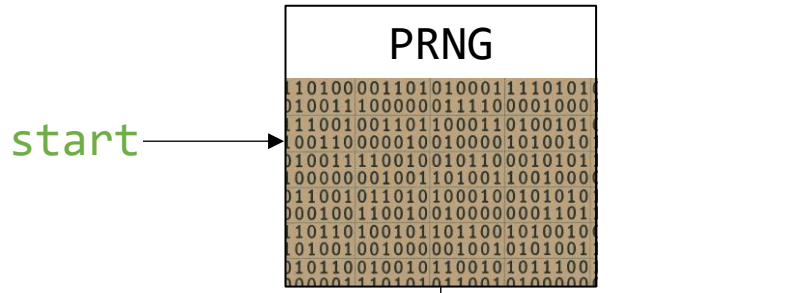


Performance Bottleneck
Fixed-Weight Vector



Fixed Weight Vector – Constant Weight Word Method

- Constant time method proposed by Sendrier [SEN21].



index[i] = i+ prng_bits %(N-i)

i+1

Weight == w?

Duplicate Check?

pos = i

pos = index[i]

done

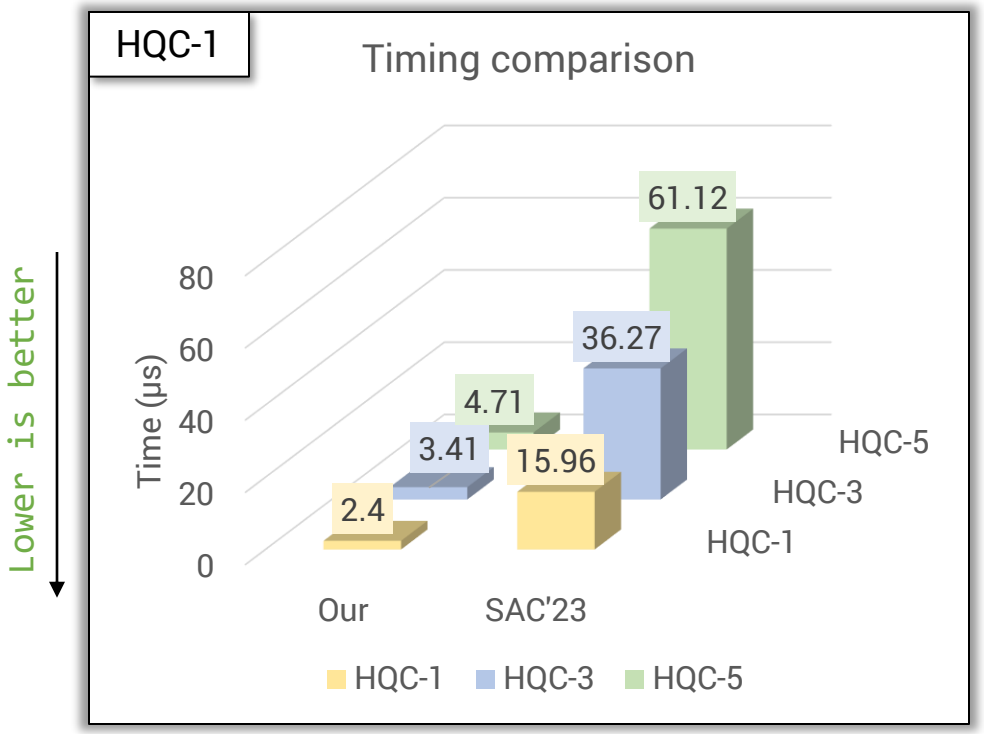
[Our]
- Technique: One-pass traversal technique – trades off space for speed
- Complexity: O(w)

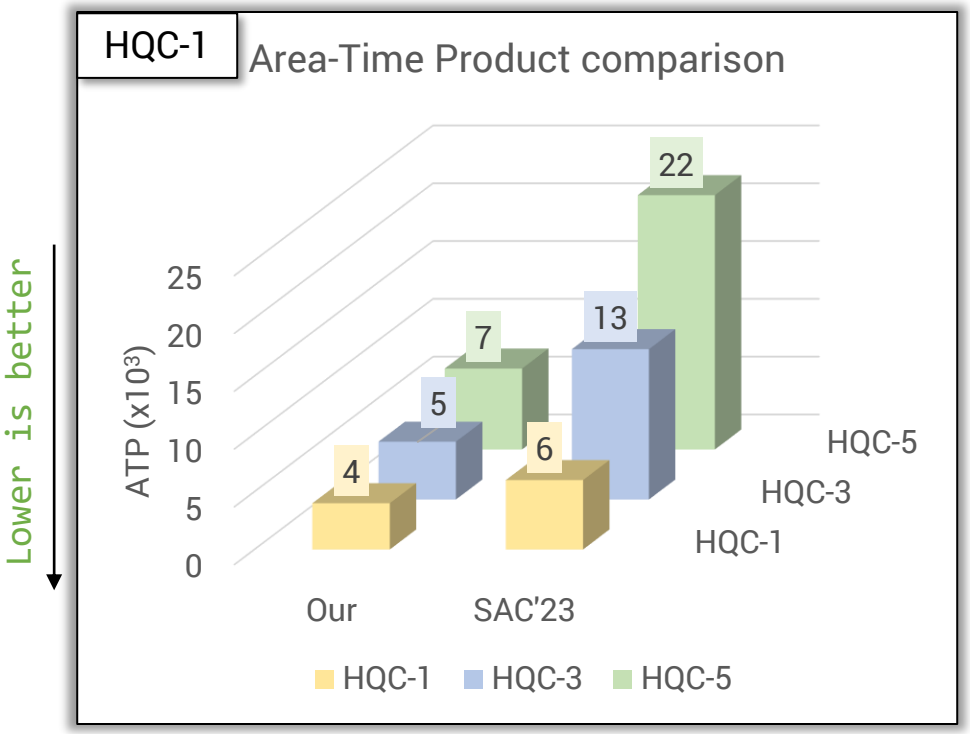
[SAC'23]
- Technique: Brute-force pairwise comparison.
- Complexity O(w^2)

set the position value to '1'

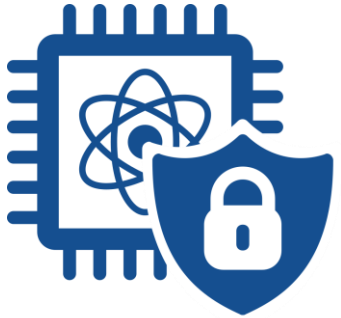
{0,1,0,0,0,1,0,0,…,0}

Fixed Weight Vector Generation - Evaluation





Performance Bottleneck
Polynomial Multiplication

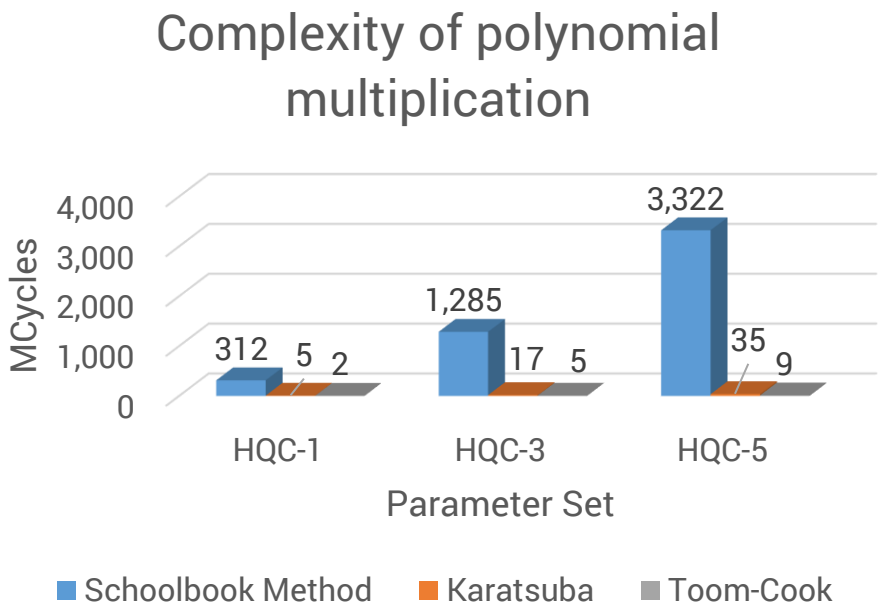


Polynomial Multiplication

A & B are polynomials in F_2

$$A(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n \qquad B(x)=b_0+b_1x+b_2x^2+\cdots+b_nx^n$$

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



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

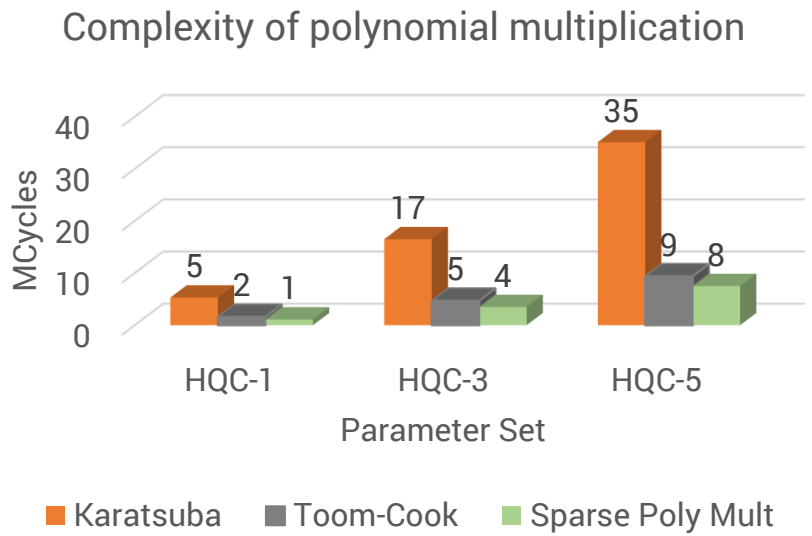
Polynomial Multiplication involving Sparse Polynomials

A & B are polynomials in F_2

$$A(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n \qquad B(x)=0+b_1x+0+\cdots+b_mx^m$$

$$c(x) = \sum_{i=0}^{w-1} A(x) << bindex[i]$$

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



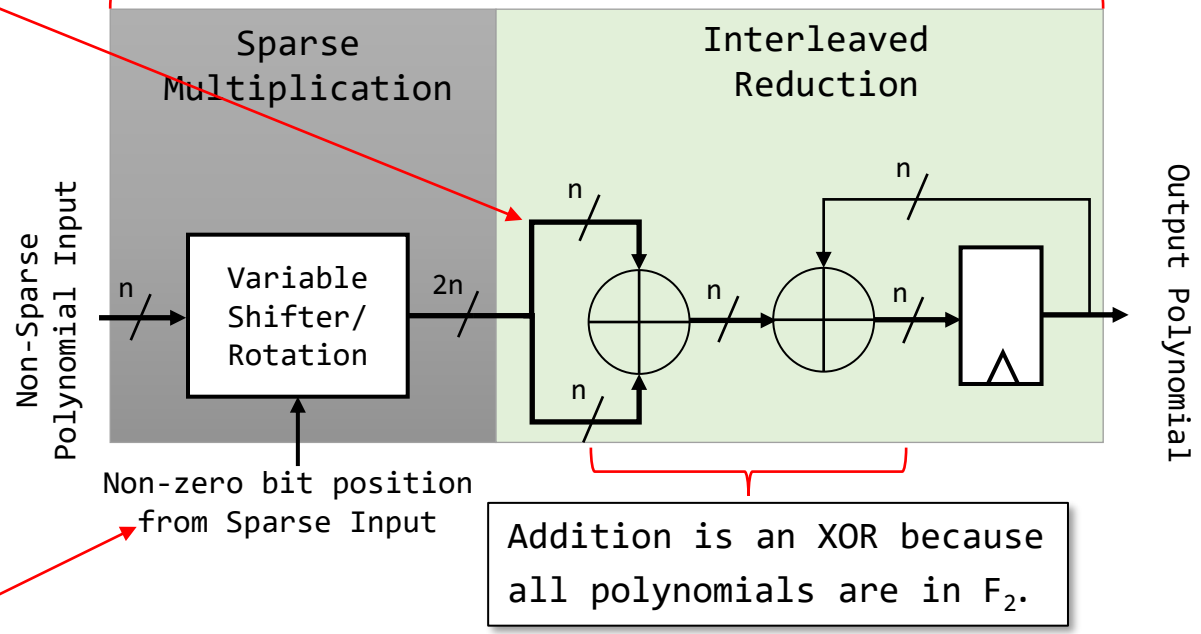
Karatsuba = $O(n^{1.585})$
Toom-Cook = $O(n^{1.465})$
Sparse poly mult = $O(n.w)$

Sparse F_2 Polynomial Multiplication with 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



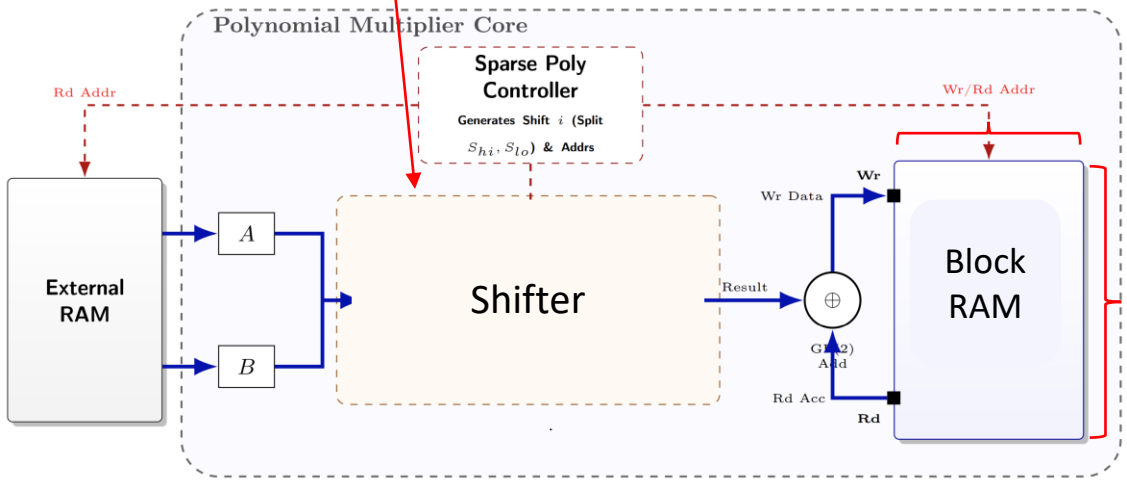
Addition is an XOR because all polynomials are in F_2.

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

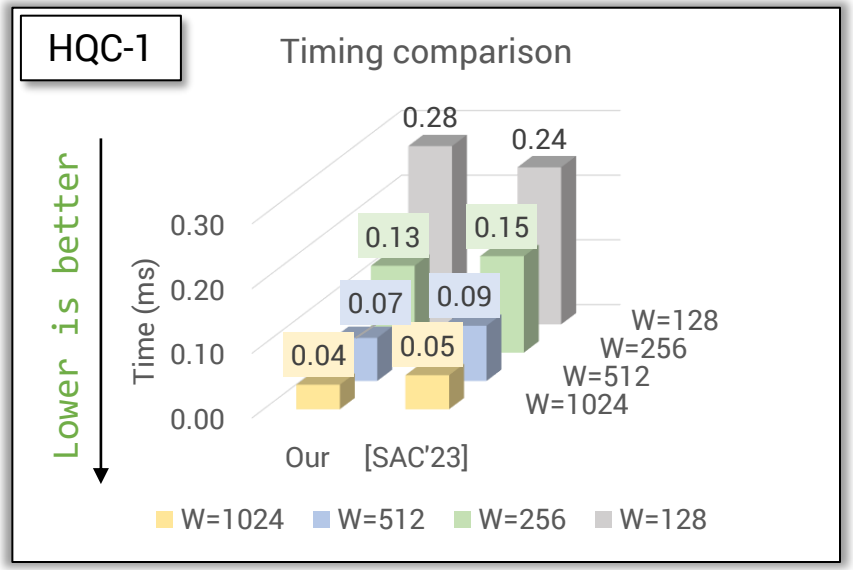
Conceptual Block Diagram

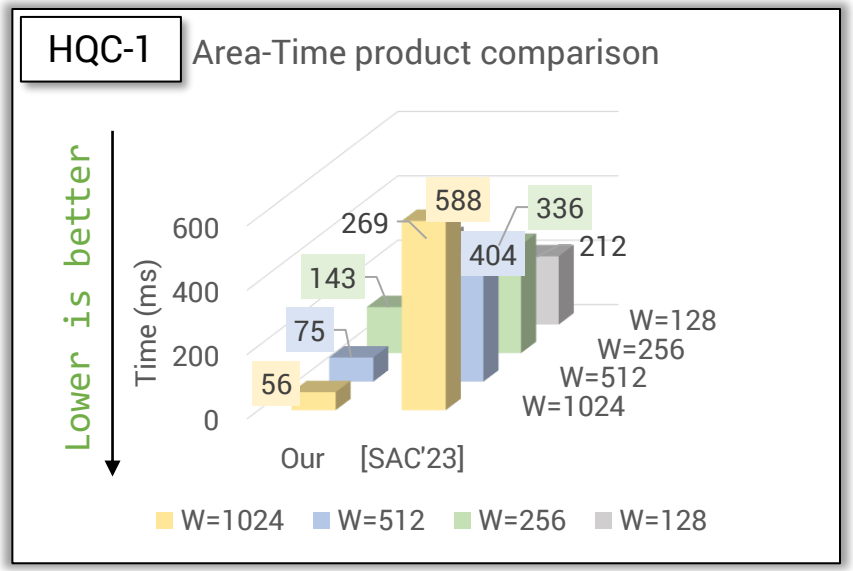
Sparse F_2 Polynomial Multiplication Hardware Design

W: Performance parameter [SAC'23]

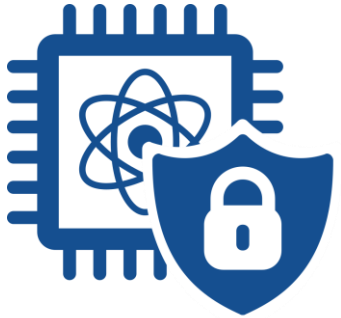


Hardware design of sparse polynomial multiplication module





Key Generation, Encapsulation, and Decapsulation

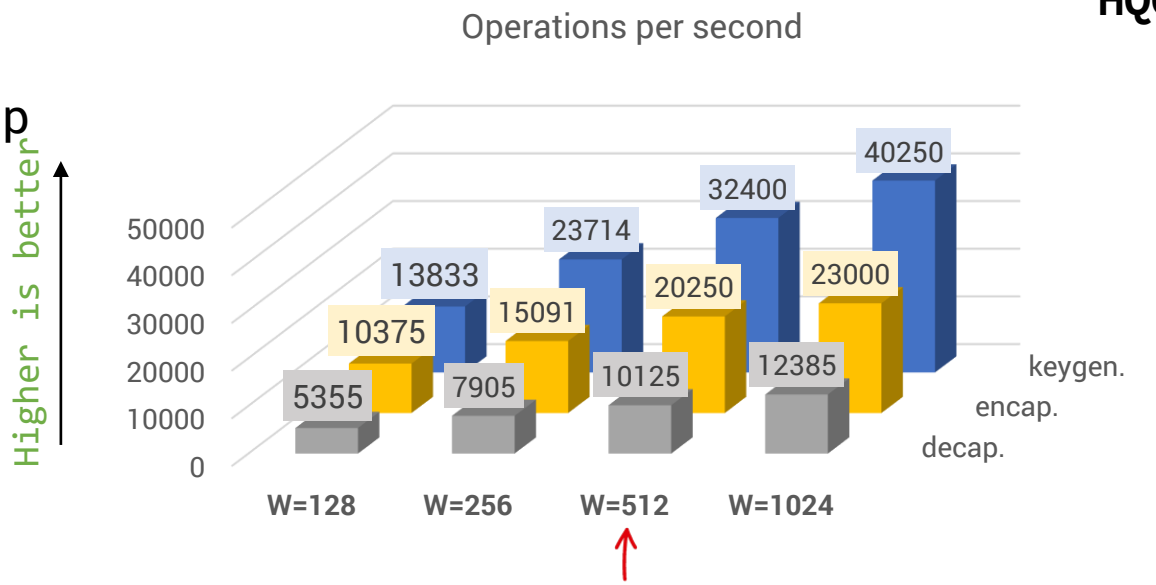


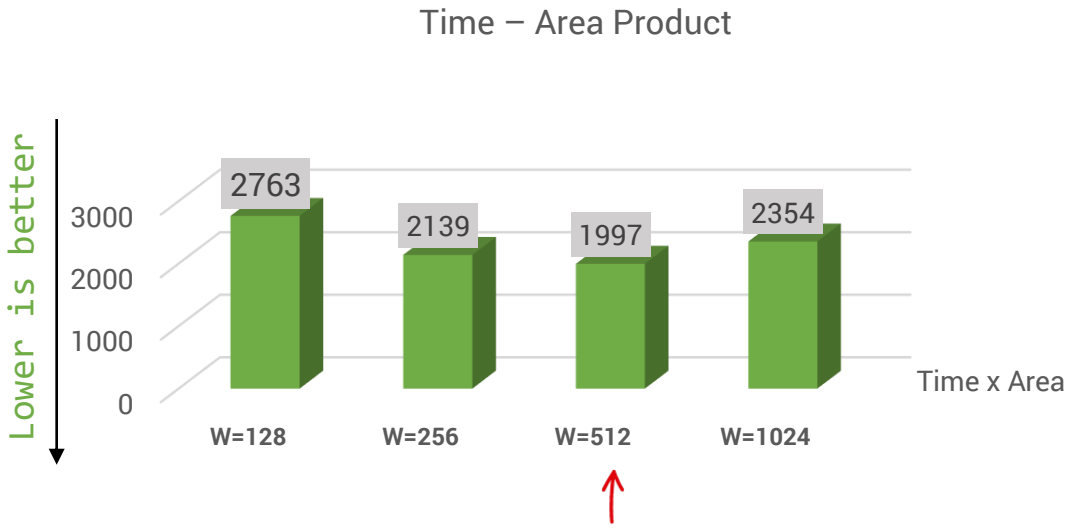
Joint Design – Sweep

FPGA: AMD Artix 7

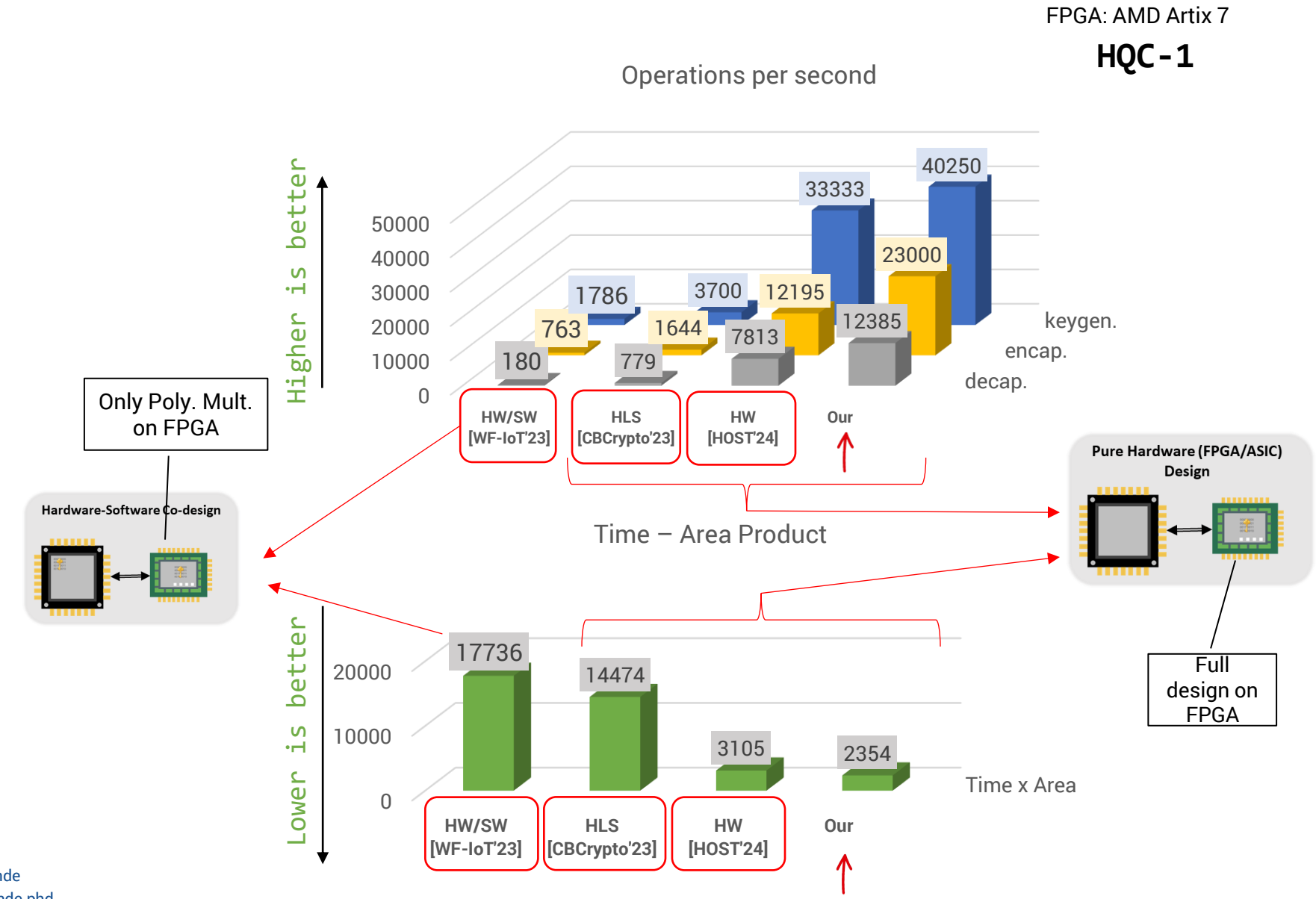
HQC-1

- 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
  - Time * max{LUT/4 + 128 ∗ BRAM + 100 ∗ DSP, FF/8}

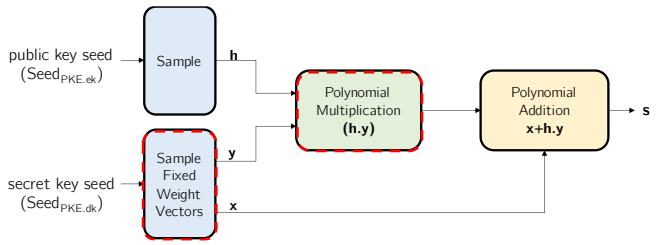




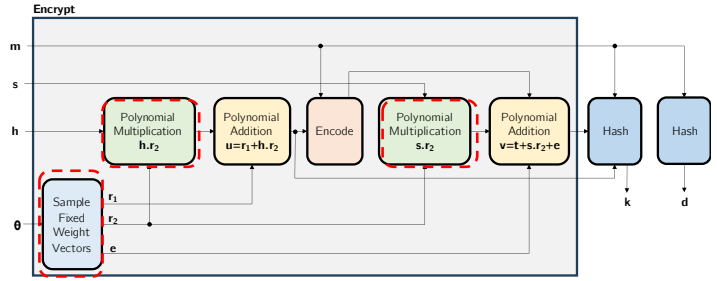
Joint Design – Comparison with other HQC Hardware Designs



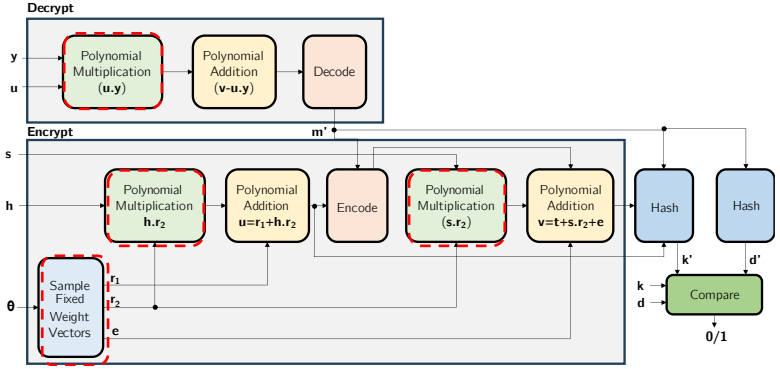
Summary



Key Generation

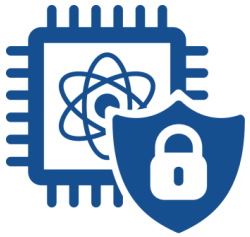


Encapsulation



Decapsulation

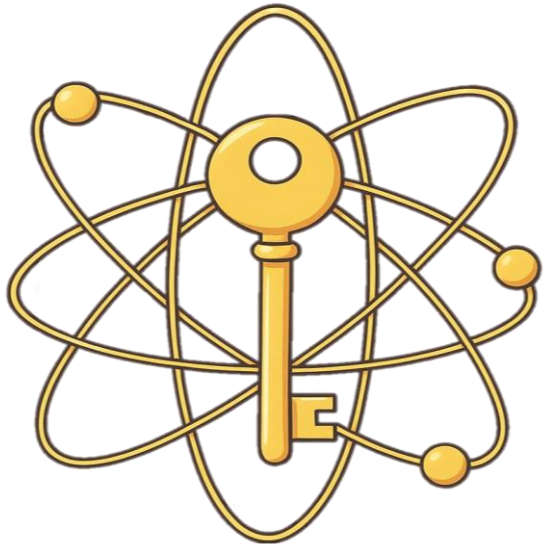
- Design and evaluation of performance critical aspects of HQC implementation
- When compared to state-of-the-art we demonstrated:
  - ~ 20% improvement in Key Generation time
  - ~ 80% improvement in Encapsulation time
  - ~ 60% improvement in Decapsulation time
  - ~ 33% improvement in the Time–Area product



Thank you!

https://sanjaydeshpande.phd/

Backup Slides

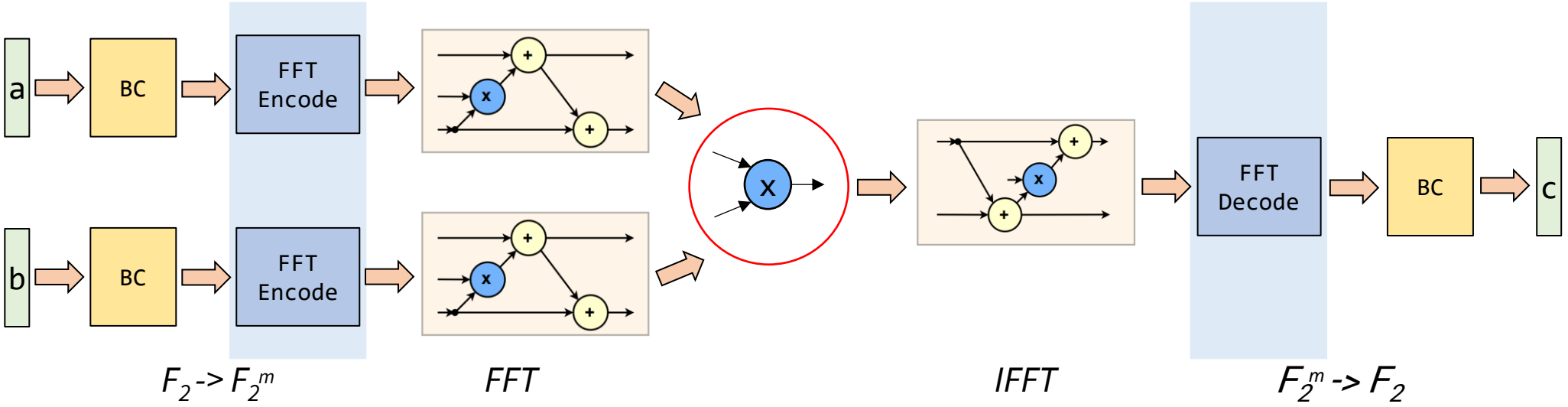


What about NTT/FFT type multiplication for HQC?

- 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).**
  - Implemented in hardware by Ras et. al [RLC+25]

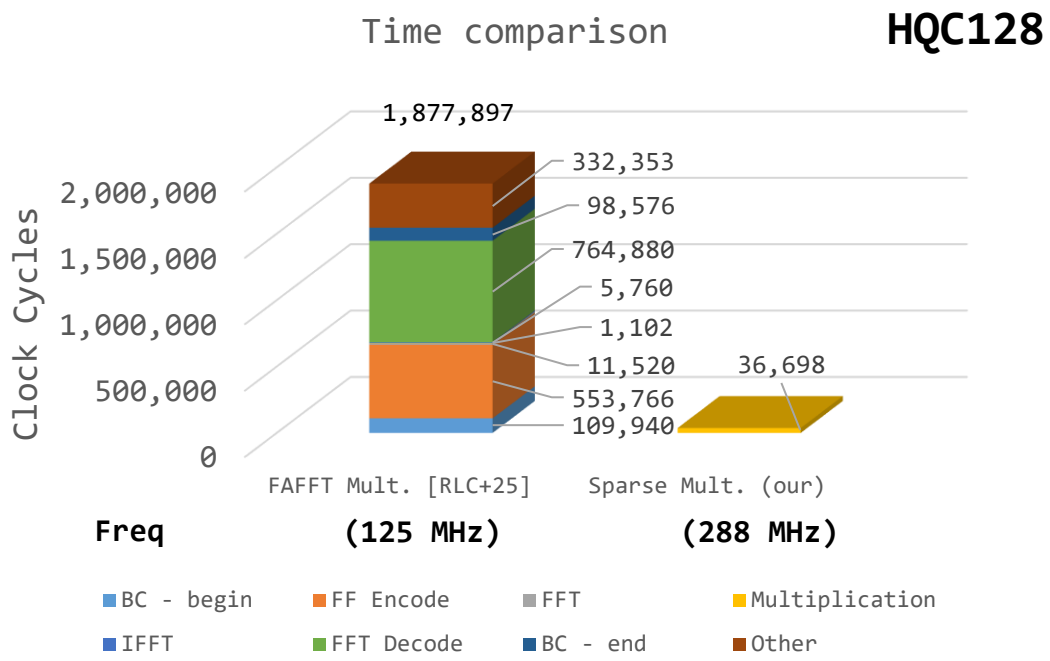
Frobenius Additive FFT (FAFFT)

- 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

- Taking advantage of sparsity doesn't lead to non-constant time behavior on hardware.
  - Weight of the sparse polynomial is fixed.



Comparison HQC vs CRYSTALS-Kyber

Scheme	Sizes (Bytes)		
	ek	dk	ct
HQC128	2,241	4,602	7,333
Kyber512	800	1,632	768

Key Sizes

Scheme	Device	Keygen	Encap	Decap	Freq. (MHz)
		Kcycles			
HQC-1 [GAA+25]	Intel Core i7-11850H	76	150	353	2,500
Kyber-512 [ABD+21]	Intel Core i7-4770K	34	45	35	3,492

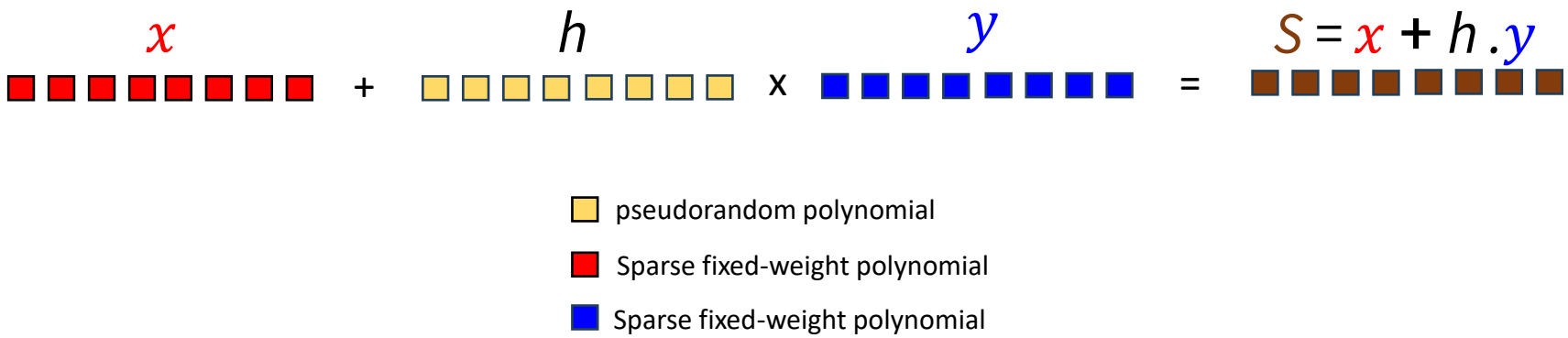
CPU Cycle Comparison

Scheme	Device	Keygen	Encap	Decap	Freq. (MHz)	Area			
		Kcycles				KLUT	KFF	DSP	BRAM
HQC-1 [our]	Artix-7	5	8	16	162	26	27	8	32.5
Kyber-512 [BMG+21]	Artix-7	2	3	5	220	10	9	4	4.5

FPGA Implementation Comparison

Syndrome Decoding Problem

- h, x, y are 'n' degree polynomials
- Setup:
  - h – sampled from a pseudo-random number generator (PRNG)
  - x, y – non-zero location of the sparse polynomial sampled from a PRNG



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