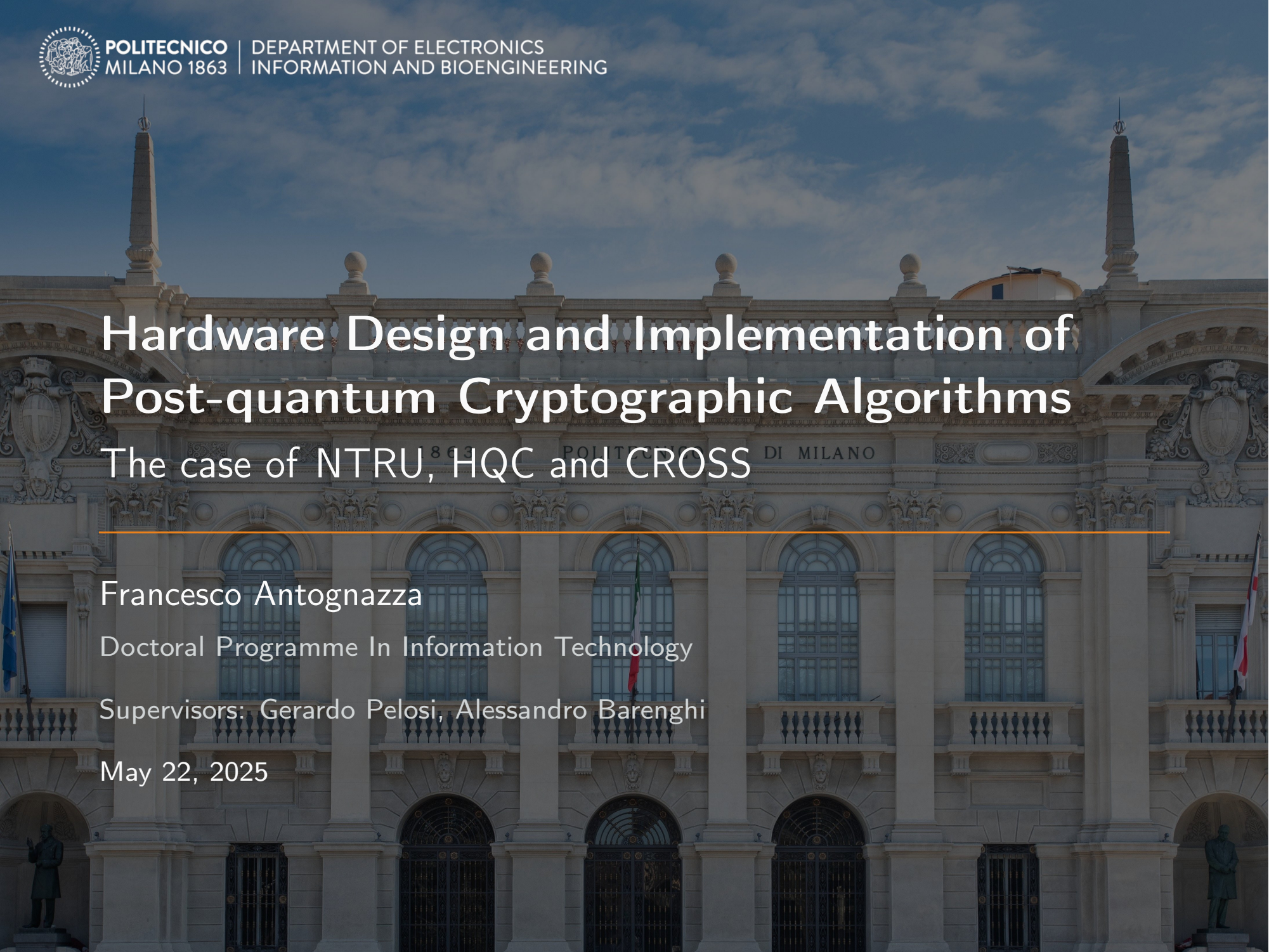


POLITECNICO MILANO 1863 | DEPARTMENT OF ELECTRONICS INFORMATION AND BIOENGINEERING

Hardware Design and Implementation of Post-quantum Cryptographic Algorithms

The case of NTRU, HQC and CROSS

Francesco Antognazza

Doctoral Programme In Information Technology

Supervisors: Gerardo Pelosi, Alessandro Barenghi

May 22, 2025

Overview

Introduction to Post-Quantum Cryptography

NTRU: N-th degree Truncated polynomial Ring Units

HQC: Hamming Quasi-Cyclic

CROSS: Codes and Restricted Objects Signature Scheme

Concluding remarks

Introduction to Post-Quantum Cryptography

Quantum computing

Today we assist to continuous advancements in the computational capabilities of quantum computers:

- more powerful QPUs: up to 1121 qubits (Condor, IBM)
- longer coherency time: circuits w/ 5000 gates (Heron, IBM)
- higher density: 100 nm pitch between qubits (spin qubit, Intel)

Quantum computing

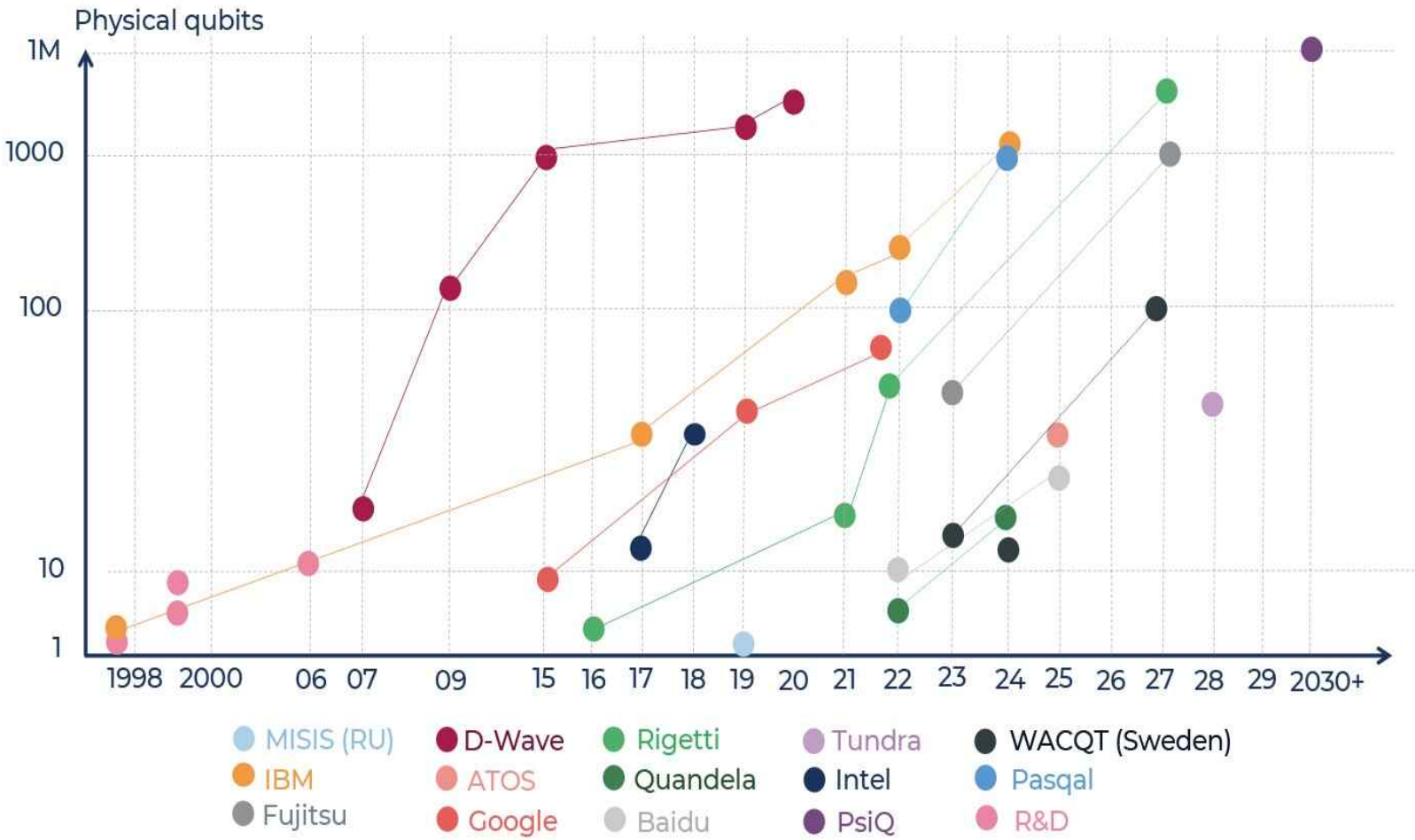
Today we assist to continuous advancements in the computational capabilities of quantum computers:

- more powerful QPUs: up to 1121 qubits (Condor, IBM)
- longer coherency time: circuits w/ 5000 gates (Heron, IBM)
- higher density: 100 nm pitch between qubits (spin qubit, Intel)

There are still several limitations afflicting the current technology, such as producing a scalable quantum memory [1], but many other roadblocks have been solved in the past few years [2].

Quantum computing

Source: Quantum Technologies 2023 report, Yole Intelligence, 2023



The quantum threat

Grover's algorithm [3] for quantum computers allows to finding zeros in a function quadratically faster compared to the best classical solution for regular computers.

Shor's algorithm [4] solves the order finding problem with a superpolynomial speedup w.r.t. non-quantum algorithms.

The quantum threat

Grover's algorithm [3] for quantum computers allows to finding zeros in a function quadratically faster compared to the best classical solution for regular computers.

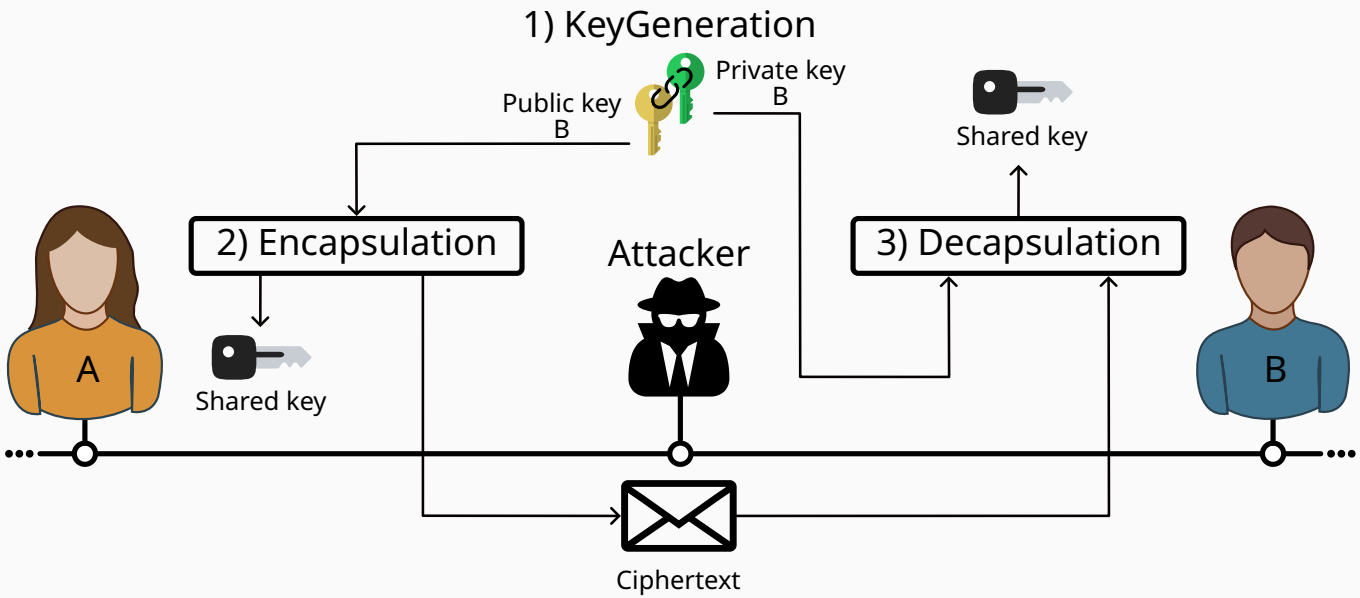
Shor's algorithm [4] solves the order finding problem with a superpolynomial speedup w.r.t. non-quantum algorithms.

The (big) trouble

All the hard problems at the base of the widely used public-key cryptosystems (e.g., ECC and RSA) are reducible to the order finding problem!

Post-Quantum Cryptography (PQC)

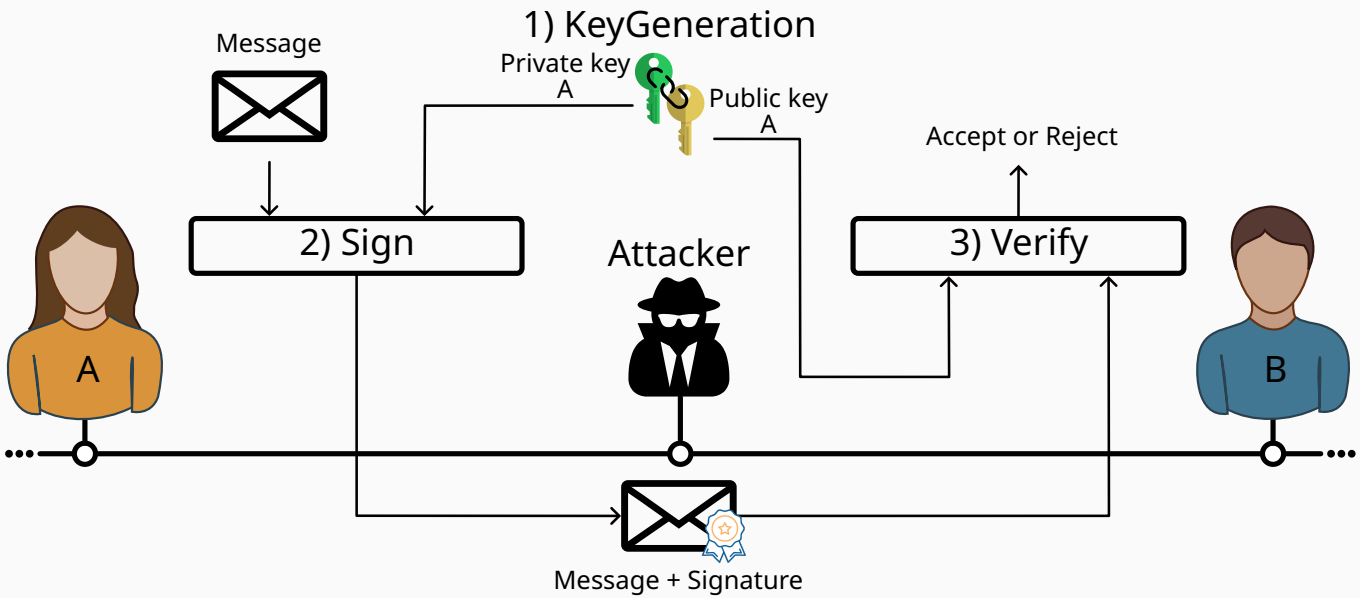
In 2016 the U.S. National Institute of Standards and Technology (NIST) started a standardization process of quantum-resistant asymmetric cryptoschemes implementing a Key Establishment Mechanism (KEM) or Digital Signature Algorithm (DSA).



Key Establishment Mechanism

Post-Quantum Cryptography (PQC)

In 2016 the U.S. National Institute of Standards and Technology (NIST) started a standardization process of quantum-resistant asymmetric cryptoschemes implementing a Key Establishment Mechanism (KEM) or Digital Signature Algorithm (DSA).



Digital Signature Algorithm

Post-Quantum Cryptography (PQC)

In 2016 the U.S. National Institute of Standards and Technology (NIST) started a standardization process of quantum-resistant asymmetric cryptoschemes implementing a Key Establishment Mechanism (KEM) or Digital Signature Algorithm (DSA).

BIG QUAKE	Gravity-SPHINCS	LUOV
BIKE	Guess Again	McNie
CFPKM	Gui	Mersenne-756839
Classic McEliece	HILA5	MQDSS
Compact LWE	HiMQ-3	NewHope
CRYSTALS-DILITHIUM	HK-17	NTRUEncrypt
CRYSTALS-KYBER	HQC	NTRU-HRSS-KEM
DAGS	KCL	NTRU Prime
Ding Key Exchange	KINDI	NTS-KEM
DME	LAC	Odd Manhattan
DRS	LAKE	Ouroboros-R
DualModeMS	LEDAkem	Picnic
Edon-K	LEDApkc	Post-quantum RSA Encryption
EMBLEM/R.EMBLEM	Lepton	Post-quantum RSA Signature
FALCON	LIMA	pqNTRUSign
FrodoKEM	Lizard	pqsigRM
GeMSS	LOCKER	QC-MDPC-KEM
Giophantus	LOTUS	qTESLA
RaCoSS	Round2	SPHINCS+
Rainbow	RQC	SRTPI
Ramstake	RVB	Three Bears
RankSign	SABER	Titanium
RLCE-KEM	SIKE	WalnutDSA

Some numbers:

- 82 submissions
- 69 valid schemes
- 26 after 1st round
- 7 after 2nd round

Post-Quantum Cryptography (PQC)

In 2016 the U.S. National Institute of Standards and Technology (NIST) started a standardization process of quantum-resistant asymmetric cryptoschemes implementing a Key Establishment Mechanism (KEM) or Digital Signature Algorithm (DSA).

Many other national and international standardization bodies are also working on Post-Quantum Cryptography (PQC):

- European Telecom. Standards Institute (ETSI) [5], [6]
- French Cybersecurity Agency (ANNSI) [7]
- German Federal Office for Information Security (BSI) [8], [9]
- Chinese Association for Cryptologic Research (CACR)
- Internet Engineering Task Force (IETF)
- International Organization for Standardization (ISO)

Post-Quantum Cryptography (PQC)

This is the current situation after the conclusion of the first standardization contest in March 2025:

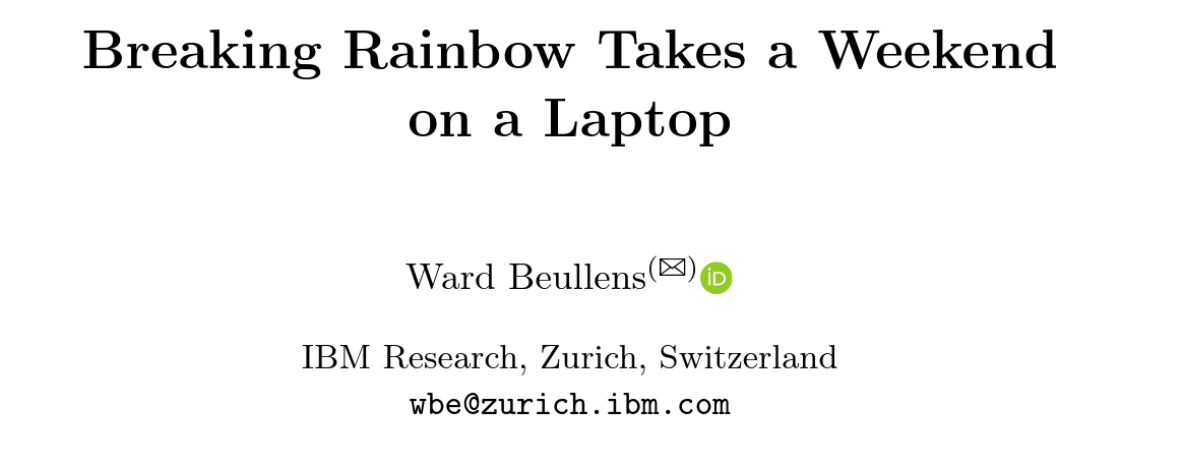
- **CRYSTALS-Kyber**: lattice-based ML-KEM, FIPS 203
- **CRYSTALS-Dilithium**: lattice-based ML-DSA, FIPS 204
- **SPHINCS$^+$**: hash-based SLH-DSA, FIPS 205
- **FALCON**: lattice-based FN-DSA, WIP
- **HQC**: code-based KEM, WIP

Post-Quantum Cryptography (PQC)

Now it's time to plan the transition to PQC [10], [11].
It is generally advised to employ a hybrid scheme combining a pre- and post-quantum cryptoscheme.

Post-Quantum Cryptography (PQC)

Now it's time to plan the transition to PQC [10], [11].
It is generally advised to employ a hybrid scheme combining a pre- and post-quantum cryptoscheme.



Breaking Rainbow Takes a Weekend on a Laptop

Ward Beullens(✉)

IBM Research, Zurich, Switzerland
wbe@zurich.ibm.com

Post-Quantum Cryptography (PQC)

Now it's time to plan the transition to PQC [10], [11].
It is generally advised to employ a hybrid scheme combining a pre- and post-quantum cryptoscheme.

Breaking Rainbow Takes a Weekend on a Laptop

IBM

An Efficient Key Recovery Attack on SIDH

Wouter Castryck[1,2(✉)] and Thomas Decru[1]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
wouter.castryck@esat.kuleuven.be
[2] Vakgroep Wiskunde: Algebra en Meetkunde, Universiteit Gent, Ghent, Belgium
thomas.decru@esat.kuleuven.be

Post-Quantum Cryptography (PQC)

A new standardization effort seeking for an alternate digital signature algorithm not based on structured lattice is still ongoing.

Code-Based	Lattice-Based	MPC-in-the-Head	Multivariate
CROSS	EagleSign	Biscuit	3WISE
Enhanced pqsigRM	EHTv4	**MIRA***	DME-Sign
FuLeeca	HAETAE	**MiRitH***	HPPC
LESS	**HAWK**	**MQOM**	**MAYO**
MEDS	HuFu	**PERK**	PROV
WAVE	Raccoon	**RYDE**	**QR-UOV**
	SQUIRRELS	**SDitH**	**SNOVA**
Other			TUOV
ALTEQ	Symmetric-Based	Isogeny-Based	**UOV**
eMLE-Sig 2.0	AIMer	**SQIsign**	VOX
KAZ-SIGN	Ascon-Sign		
PREON	**FAEST**		
Xifrat1-Sign.I	SPHINCS-alpha		

Challenges in designing and implementing PQC

Designing a new cryptographic algorithm is a challenging task:

- parameter selection, producing adequate security proofs
- trying algebraic attacks for key/message recovery
- perf. evaluation in protocols in emulated/real environments
- explore optimization corners: sub-algorithms, vectorization
  - in terms of latency, area, efficiency, power consumptions
- physical security evaluation: side-channel attacks, fault attacks

Challenges in designing and implementing PQC

Designing a new cryptographic algorithm is a challenging task:

- parameter selection, producing adequate security proofs
- trying algebraic attacks for key/message recovery
- perf. evaluation in protocols in emulated/real environments
- explore optimization corners: sub-algorithms, vectorization
  - in terms of latency, area, efficiency, power consumptions
- physical security evaluation: side-channel attacks, fault attacks

Ph.D. thesis contribution

We are focusing on the design of dedicated hardware accelerators for the full PQC algorithms NTRU, HQC, and CROSS.

NTRU: N-th degree Truncated polynomial Ring Units

Introduction to NTRU

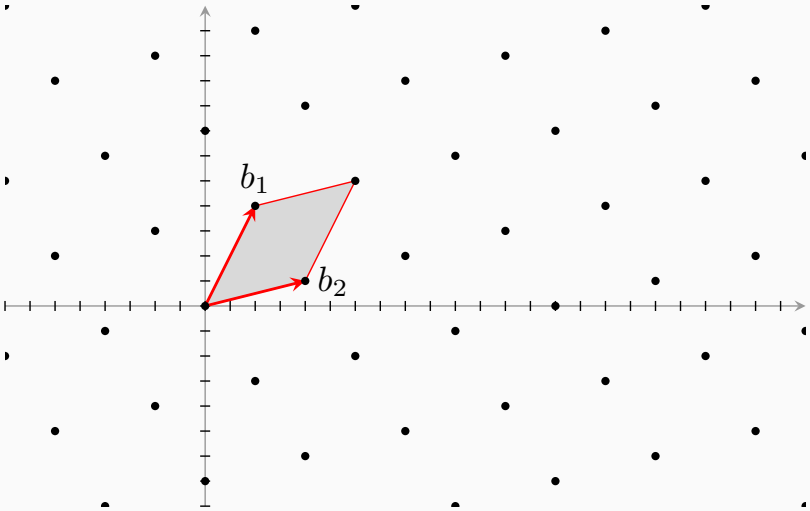
The N-th degree Truncated polynomial Ring Units (NTRU) is a lattice-based KEM relying on the hardness of the Shortest Vector Problem (SVP) – and the closely related Closest Vector Problem (CVP) – in a n-dimensional integer lattice, both NP-hard problems.

The original work is dated back in 1998 [12], and several iterations improved the performance and security estimation.
In the PQC contest, there were three independent submissions: **NTRU-HPS**, **NTRU-HRSS**, Streamlined NTRU Prime

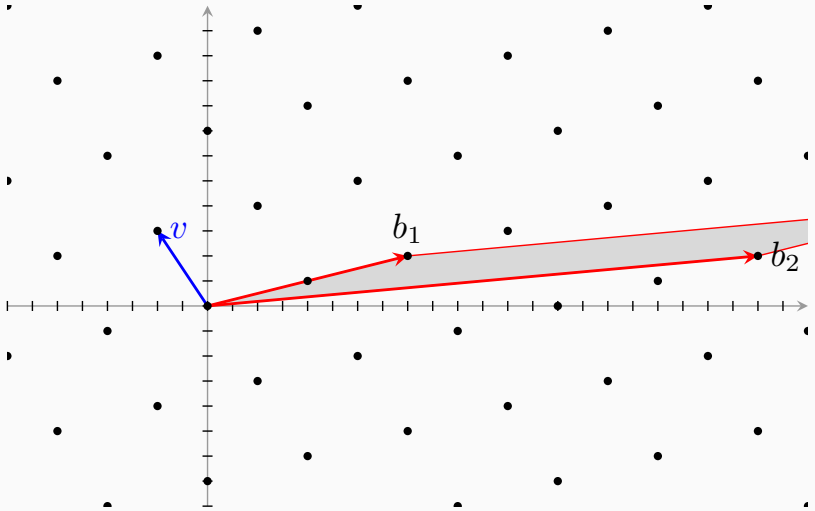
Compared to the standardized ML-KEM algorithm, NTRU can scale the security margin more easily, but has slightly large ciphertext sizes and a particularly slow key generation procedure.

Given $k \in \mathbb{N}$ linearly independent vectors $\mathrm{b}_i \in \mathbb{R}^n$, with $1 \leq i \leq k$ and $k \leq n$, an instance of a lattice $\mathcal{L}$ is the set of points in $\mathbb{R}^n$:

$$\mathcal{L} := \Lambda(\mathrm{b}_1, \mathrm{b}_2, \ldots, \mathrm{b}_k) = \left\{ \sum_{i=1}^{k} a_i \mathrm{b}_i \mid a_i \in \mathbb{Z} \right\} \subseteq \mathbb{R}^n$$



$\mathrm{b}_1 = [4, 2]$, $\mathrm{b}_2 = [2, 4]$

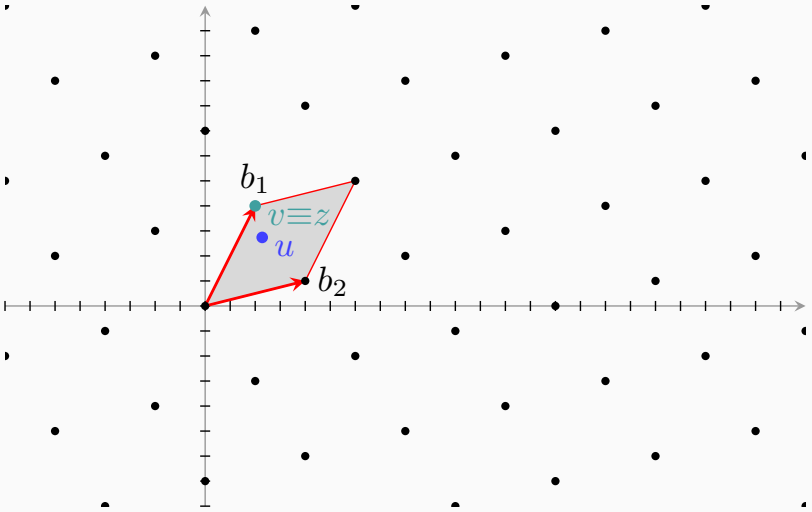


$\mathrm{b}_1 = [8, 2]$, $\mathrm{b}_2 = [22, 2]$

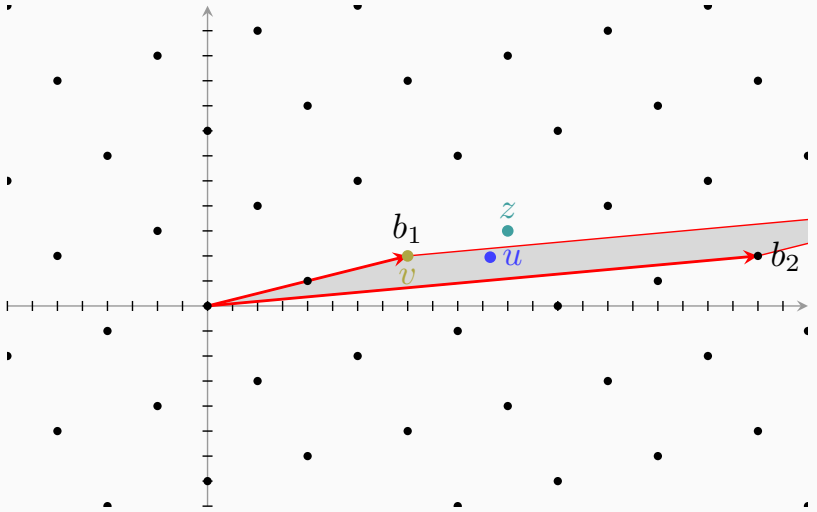
SVP in the integer lattice $\mathcal{L} \subset \mathbb{Z}^2$

Given $k \in \mathbb{N}$ linearly independent vectors $\mathrm{b}_i \in \mathbb{R}^n$, with $1 \leq i \leq k$ and $k \leq n$, an instance of a lattice $\mathcal{L}$ is the set of points in $\mathbb{R}^n$:

$$\mathcal{L} := \Lambda(\mathrm{b}_1, \mathrm{b}_2, \ldots, \mathrm{b}_k) = \left\{ \sum_{i=1}^{k} a_i \mathrm{b}_i \mid a_i \in \mathbb{Z} \right\} \subseteq \mathbb{R}^n$$



$\mathrm{b}_1 = [4, 2]$, $\mathrm{b}_2 = [2, 4]$



$\mathrm{b}_1 = [8, 2]$, $\mathrm{b}_2 = [22, 2]$

CVP in the integer lattice $\mathcal{L} \subset \mathbb{Z}^2$

NTRU makes use of arithmetic in the quotient polynomial ring

$$R = \mathbb{Z}[x] / \langle x^n - 1 \rangle \qquad \text{with } (x^n - 1) = \Phi_1 \Phi_n$$

$n \in \{509, 677, 701, 821\}$ prime numbers $\Rightarrow \Phi_1, \Phi_n$ are irreducible

NTRU makes use of arithmetic in the quotient polynomial ring

$$R = \mathbb{Z}[x] / \langle x^n - 1 \rangle \quad \text{with } (x^n - 1) = \Phi_1 \Phi_n$$

$n \in \{509, 677, 701, 821\}$ prime numbers $\Rightarrow \Phi_1, \Phi_n$ are irreducible

Internally the scheme actually works with three polynomial rings:

$R_q = \mathbb{Z}_q[x] / \langle \Phi_1 \Phi_n \rangle \quad S_q = \mathbb{Z}_q[x] / \langle \Phi_n \rangle \quad S_p = \mathbb{Z}_p[x] / \langle \Phi_n \rangle$

NTRU makes use of arithmetic in the quotient polynomial ring

$$\mathrm{R} = \mathbb{Z}[x] / \langle x^n - 1 \rangle \quad \text{with } (x^n - 1) = \Phi_1 \Phi_n$$

$n \in \{509, 677, 701, 821\}$ prime numbers $\Rightarrow \Phi_1, \Phi_n$ are irreducible

Internally the scheme actually works with three polynomial rings:

$\mathrm{R}_q = \mathbb{Z}_q[x] / \langle \Phi_1 \Phi_n \rangle \quad \mathrm{S}_q = \mathbb{Z}_q[x] / \langle \Phi_n \rangle \quad \mathrm{S}_p = \mathbb{Z}_p[x] / \langle \Phi_n \rangle$

- **large** polynomial: coefficients in $\mathbb{Z}_q$, $q \in \{2048, 4096, 8192\}$
- **small** polynomial: coefficients in $\mathbb{Z}_p = \mathbb{Z}_3$
  - **fixed-weight**: $d \in \{127, 255\}$ coefficients eq. to 1 and -1
  - **variable-weight**: any number of non-null coefficients

NTRU makes use of arithmetic in the quotient polynomial ring

$$R = \mathbb{Z}[x] / \langle x^n - 1 \rangle \quad \text{with } (x^n - 1) = \Phi_1 \Phi_n$$

$n \in \{509, 677, 701, 821\}$ prime numbers $\Rightarrow \Phi_1, \Phi_n$ are irreducible

Internally the scheme actually works with three polynomial rings:

$R_q = \mathbb{Z}_q[x] / \langle \Phi_1 \Phi_n \rangle \quad S_q = \mathbb{Z}_q[x] / \langle \Phi_n \rangle \quad S_p = \mathbb{Z}_p[x] / \langle \Phi_n \rangle$

- **large** polynomial: coefficients in $\mathbb{Z}_q$, $q \in \{2048, 4096, 8192\}$
- **small** polynomial: coefficients in $\mathbb{Z}_p = \mathbb{Z}_3$
  - **fixed-weight**: $d \in \{127, 255\}$ coefficients eq. to 1 and -1
  - **variable-weight**: any number of non-null coefficients

Further constraints to make a deterministic cryptographic scheme:

- $\gcd(p, q) = 1, p \ll q$
- $q > (6d + 1)\, p$

Background
Arithmetic in $\mathrm{R}_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$

$\mathrm{f} = f_{n-1}x^{n-1} + \ldots + f_1 x + f_0 \in \mathrm{R}$ seen as $[f_{n-1}, \cdots, f_1, f_0] \in \mathbb{Z}^n$

$\mathrm{f} = f_{n-1}x^{n-1} + \ldots + f_1 x + f_0 \in \mathrm{R}$ seen as $[f_{n-1}, \cdots, f_1, f_0] \in \mathbb{Z}^n$

Polynomial addition

Let $\mathrm{a}, \mathrm{b} \in \mathrm{R}_q$, their sum $\mathrm{c} = \mathrm{a} + \mathrm{b}$ has coefficients

$$c_k \equiv_q a_k + b_k, \quad \forall k \in \{0, \ldots, n-1\}$$

$\mathsf{f} = f_{n-1}x^{n-1} + \ldots + f_1 x + f_0 \in \mathsf{R}$ seen as $[f_{n-1}, \cdots, f_1, f_0] \in \mathbb{Z}^n$

Polynomial addition

Let $\mathsf{a}, \mathsf{b} \in \mathsf{R}_q$, their sum $\mathsf{c} = \mathsf{a} + \mathsf{b}$ has coefficients

$$c_k \equiv_q a_k + b_k, \quad \forall k \in \{0, \ldots, n-1\}$$

Polynomial product (circular convolution)

Let $\mathsf{a}, \mathsf{b} \in \mathsf{R}_q$, their product $\mathsf{c} = \mathsf{a} \cdot \mathsf{b}$ has coefficients

$$c_k \equiv_q \sum_{i+j \equiv k \bmod n} a_i \cdot b_j, \quad \forall k \in \{0, \ldots, n-1\}$$

NTRU KEM scheme

NTRU.KEM-KeyGeneration

Require: None

Ensure: $\mathsf{pk} = h^{\mathtt{pkd}}$
$\mathsf{sk} = (f^{\mathtt{pkd}}, f_p^{\mathtt{pkd}}, h_q^{\mathtt{pkd}}, s \in \{0,1\}^{256})$

1: $\gamma \xleftarrow{\$} \{0,1\}^{320}$
2: $(s, \mathtt{str\_f}, \mathtt{str\_g}) \leftarrow \mathsf{CSPRNG}(\gamma, \{0,1\}^{8\cdot(n-1)} \times \{0,1\}^{\beta})$
3: $f \leftarrow \mathsf{CSPRNG}(\mathtt{str\_f}, \mathcal{L}_f)$, $g \leftarrow \mathsf{CSPRNG}(\mathtt{str\_g}, \mathcal{L}_g)$ ▷ *Sample two random small polys*
4: $f_p \leftarrow f^{-1} \bmod (p, \Phi_n)$ ▷ *Inverse ring element*
5: $G \leftarrow p \cdot g$
6: $v \leftarrow (G \cdot f) \bmod (q, \Phi_n)$ ▷ *Small poly by a large poly multiplication*
7: $v_q \leftarrow v^{-1} \bmod (q, \Phi_n)$ ▷ *Inverse ring element*
8: $h \leftarrow (v_q \cdot G \cdot G) \bmod (q, \Phi_1\Phi_n)$ ▷ *Two small poly by a large poly multiplications*
9: $h_q \leftarrow (v_q \cdot f \cdot f) \bmod (q, \Phi_1\Phi_n)$ ▷ *Two small poly by a large poly multiplications*
10: $h^{\mathtt{pkd}} \leftarrow \mathsf{Pack}_q(h)$; $f^{\mathtt{pkd}} \leftarrow \mathsf{Pack}_p(f)$; $f_p^{\mathtt{pkd}} \leftarrow \mathsf{Pack}_p(f_p)$; $h_q^{\mathtt{pkd}} \leftarrow \mathsf{Pack}_q(h_q)$
11: **return** $\mathsf{pk} = h^{\mathtt{pkd}}$, $\mathsf{sk} = (f^{\mathtt{pkd}}, f_p^{\mathtt{pkd}}, h_q^{\mathtt{pkd}}, s)$

Lattice	Polynomials	HPS	HRSS
L_f	f	variable-weight	variable-weigth$^+$
L_g	g	fixed-weight (d)	variable-weigth$^+$

NTRU KEM scheme

NTRU.KEM-Encapsulation

Require: $\mathsf{pk} = h^{\mathrm{pkd}}$
Ensure: $\mathsf{ctx} = c^{\mathrm{pkd}}$
$\mathsf{K} \in \{0,1\}^{256}$

1: $h \leftarrow \mathsf{Unpack}_q(h^{\mathrm{pkd}})$
2: $\gamma \xleftarrow{\$} \{0,1\}^{320}$
3: $(\mathtt{str\_r}, \mathtt{str\_m}) \leftarrow \mathsf{CSPRNG}(\gamma, \{0,1\}^{8\cdot(n-1)} \times \{0,1\}^{\beta})$
4: $r \leftarrow \mathsf{CSPRNG}(\mathtt{str\_r}, \mathcal{L}_r)$; $m \leftarrow \mathsf{CSPRNG}(\mathtt{str\_m}, \mathcal{L}_m)$ ▷ *Sample two random small polys*
5: $m' \leftarrow \mathsf{Lift}(m)$ ▷ *Map* $\mathsf{Lift} : \mathsf{S}_p \mapsto \mathsf{R}_q$ *s.t.* $\mathsf{Lift}(m) \bmod (p, \Phi_n) = m$
6: $c \leftarrow (r \cdot h + m') \bmod (q, \Phi_1\Phi_n)$ ▷ *Small poly by a large poly multiplication*
7: $c^{\mathrm{pkd}} \leftarrow \mathsf{Pack}_q(c)$; $r^{\mathrm{pkd}} \leftarrow \mathsf{Pack}_p(r)$; $m^{\mathrm{pkd}} \leftarrow \mathsf{Pack}_p(m)$
8: $\mathsf{K} \leftarrow \mathsf{Hash}(r^{\mathrm{pkd}} \| m^{\mathrm{pkd}})$
9: **return** $c^{\mathrm{pkd}}, \mathsf{K}$

Lattice	Polynomials	HPS	HRSS
L_r	r	variable-weight	variable-weight
L_m	m	fixed-weight (d)	variable-weight
$\mathsf{Lift}(\cdot)$		sign extension	$\Phi_1 \cdot ((\mathrm{a}/\Phi_1) \bmod (p, \Phi_n))$

NTRU KEM scheme

NTRU.KEM-Decapsulation

Require: $\mathsf{sk} = (f^{\mathtt{pkd}}, f_p^{\mathtt{pkd}}, h_q^{\mathtt{pkd}}, s)$
$\mathsf{ctx} = c^{\mathtt{pkd}}$
Ensure: $\mathsf{K} \in \{0,1\}^{256}$

1: $f \leftarrow \mathsf{Unpack}_q(f^{\mathtt{pkd}})$; $f_p \leftarrow \mathsf{Unpack}_q(f_p^{\mathtt{pkd}})$; $h_q \leftarrow \mathsf{Unpack}_q(h_q^{\mathtt{pkd}})$; $c \leftarrow \mathsf{Unpack}_q(c^{\mathtt{pkd}})$
2: ▷ $a = (r \cdot h + m) \cdot f = (r \cdot p \cdot f_p \cdot g + m) \cdot f = r \cdot p \cdot g + f \cdot m$ ◁
3: $a \leftarrow (c \cdot f) \bmod (q, \Phi_1\Phi_n)$ ▷ *Small poly by a large poly multiplication*
4: $m \leftarrow (a \cdot f_p) \bmod (p, \Phi_n)$ ▷ *Small poly by a large poly multiplication*
5: $m' \leftarrow \mathsf{Lift}(m)$ ▷ *Map* $\mathsf{Lift} : \mathsf{S}_p \mapsto \mathsf{R}_q$ *s.t.* $\mathsf{Lift}(m) \bmod (p, \Phi_n) = m$
6: $r \leftarrow ((c - m') \cdot h_q) \bmod (q, \Phi_n)$ ▷ *Large poly by a large poly multiplication*
7: $r^{\mathtt{pkd}} \leftarrow \mathsf{Pack}_p(r)$; $m^{\mathtt{pkd}} \leftarrow \mathsf{Pack}_p(m)$
8: $\mathsf{K}_1 \leftarrow \mathsf{Hash}(r^{\mathtt{pkd}} \| m^{\mathtt{pkd}})$; $\mathsf{K}_2 \leftarrow \mathsf{Hash}(s \| c^{\mathtt{pkd}})$
9: **if** $(r, m) \notin \mathcal{L}_r \times \mathcal{L}_m \vee c \not\equiv 0 \bmod (q, \Phi_1)$ **then** $\mathsf{K} = \mathsf{K}_2$ **else** $\mathsf{K} = \mathsf{K}_1$
10: **return** K

line 3: since $a \in \mathsf{R}_q$, no actual mod q is performed

line 4: performing mod p removes the first term containing the randomness r, obtaining $f \cdot f_p \cdot m$

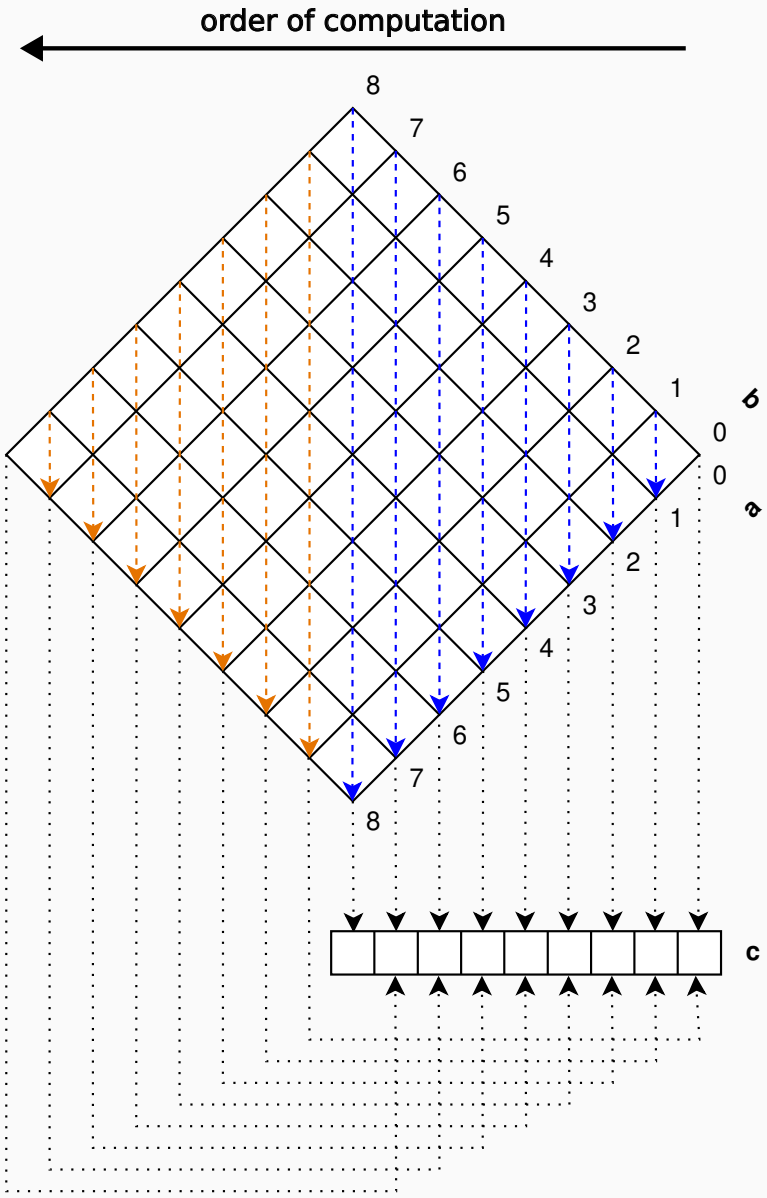
Multiplication in $R_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$ – Comba algorithm

Comba multiplication

Require: $a \in R_q$, $b \in R_q$
Ensure: $c \in R_q \mid c = a \cdot b$

1: $c \leftarrow 0$
2: **for** $j \leftarrow 0$ **to** $n-1$ **do**
3: **for** $i \leftarrow 0$ **to** j **do**
4: $t \leftarrow t + a_{j-i} \cdot b_i$
5: $c_j \leftarrow c_j + t$ ▷ *To memory*
6: **for** $j \leftarrow 0$ **to** $n-2$ **do**
7: **for** $i \leftarrow 0$ **to** j **do**
8: $t \leftarrow t + a_{n-1-j+i} \cdot b_{n-1-i}$
9: $c_{n-1-j} \leftarrow c_{n-1-j} + t$ ▷ *To memory*
10: **return** c

Employs a single scalar multiplier and an adder.



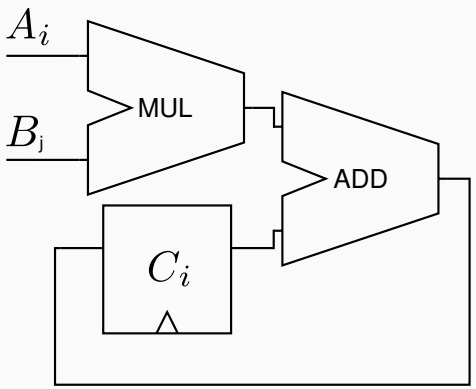
Multiplication in $R_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$ – x-net algorithm

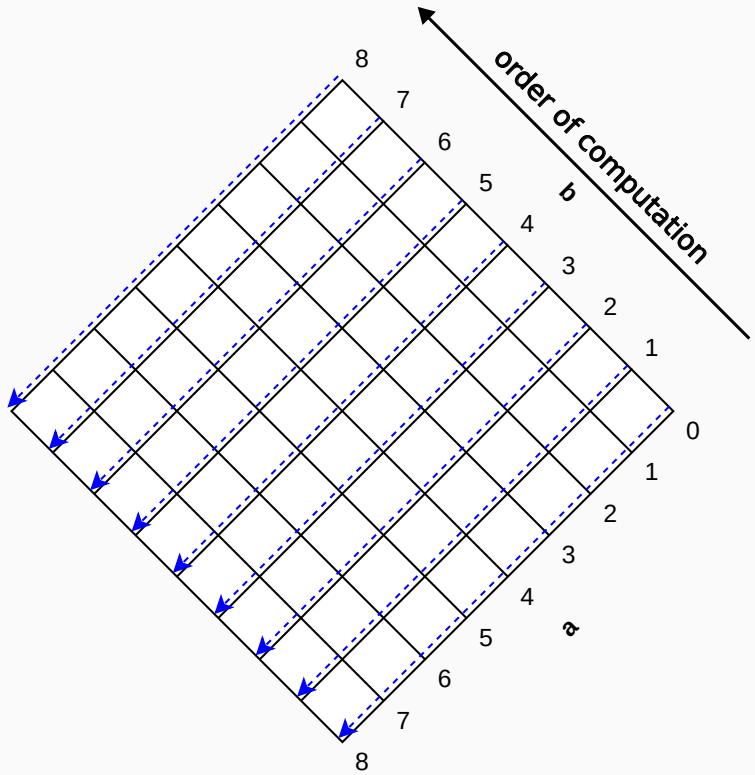
a and c in stored in flip-flops. One coefficient of b processed per CC by n multiply-and-accumulate units (MACs).

x-net multiplication

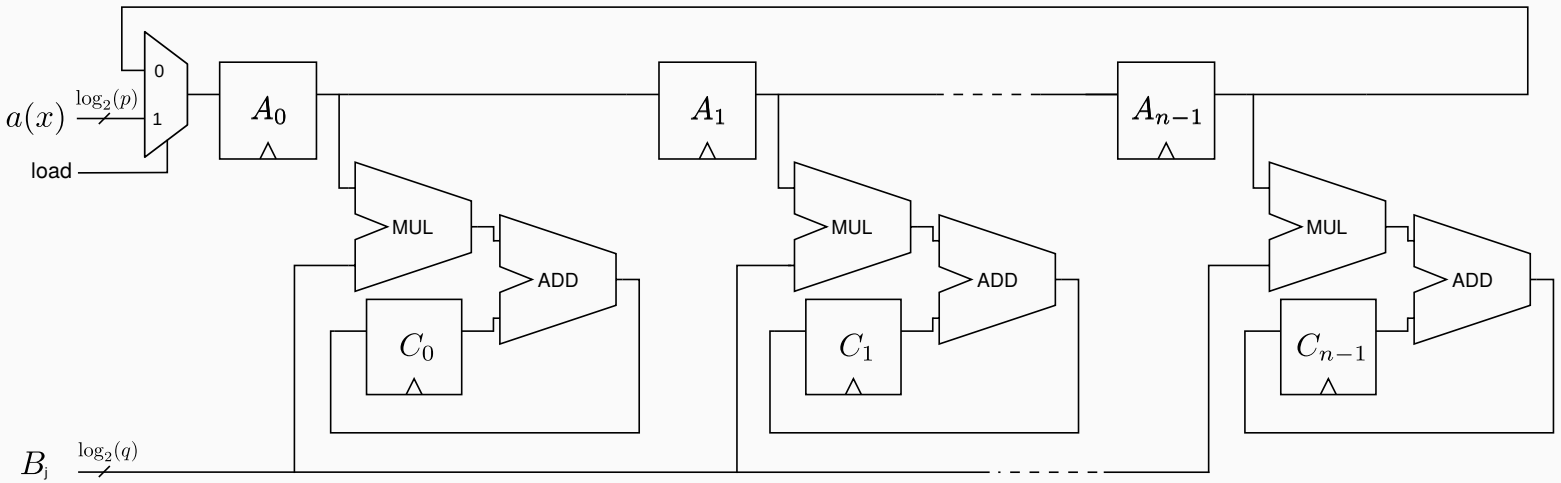
Require: $a \in R_q$, $b \in R_q$
Ensure: $c \in R_q \mid c = a \cdot b$

1: $c \leftarrow 0$
2: **for** $j \leftarrow 0$ **to** $n - 1$ **do**
3: $\quad c \leftarrow c + b_j \cdot a$ ▷ *n parallel MACs*
4: $\quad a \leftarrow a \cdot [0, 1, 0, \ldots, 0] \bmod x^n - 1$ ▷ *Rotation*
5: **return** c





Multiplication in $R_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$ – x-net algorithm



Poly multiplication with n coefficients performed in n CC.
If $a \in S_p$:

- scalar multiplications replaced with a 3-input multiplexer
- three scalar mul. results computed only once and distributed

Lift $S_p \mapsto R_q$

Lift operation in NTRU-HRSS [13] – $\Phi_1 \cdot ((a/\Phi_1) \bmod (p, \Phi_n))$ without multiplications

Require: $a \in \mathcal{S}_p$
Ensure: $b \in \mathcal{R}_q \mid (b) \bmod (p, \Phi_n) = a$

for $i \leftarrow 0$ **to** $n-2$ **do**
  $c_i \leftarrow (1-i) \mod p$ ▷ $c \leftarrow (1/\Phi_1) \bmod (p, \Phi_n)$ *for NTRU parameters*
for $i \leftarrow 0$ **to** $p-1$ **do**
  $d_i \leftarrow \langle x^i \bar{c}, a \rangle$ ▷ *inner-product of a with the rotated reversal map of c*
for $i \leftarrow p$ **to** $n-1$ **do**
  $d_i \leftarrow d_{i-p} - \sum_{j=0}^{p-1} a_{i-j}$
$d_0 \leftarrow d_0 - d_{n-1} \mod p$
$b_0 \leftarrow -d_0$
▷ *multiplication by Φ_1 replaced by additions* ◁
for $i \leftarrow 1$ **to** $n-1$ **do**
  $d_i \leftarrow d_i - d_{n-1} \mod p$
  $b_i \leftarrow d_{i-1} - d_i \mod q$
return b

2 poly multiplications as $8n$ scalar additions/subtractions

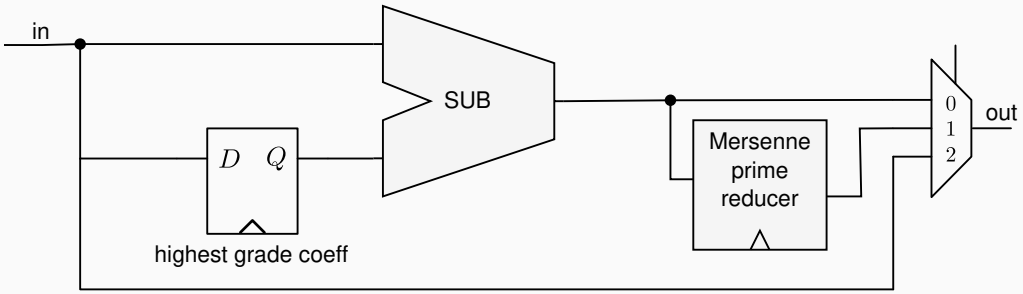
Embedding $\mathsf{E}_1 : \mathsf{R}_q \mapsto \mathsf{S}_q$ and $\mathsf{E}_2 : \mathsf{R}_q \mapsto \mathsf{S}_p$

Moving from ring R to S is efficiently performed subtracting the coefficient with highest grade x^{n-1} to all the others.

Embedding $\mathsf{E}_1 : \mathsf{R}_q \mapsto \mathsf{S}_q$ and $\mathsf{E}_2 : \mathsf{R}_q \mapsto \mathsf{S}_p$

Moving from ring R to S is efficiently performed subtracting the coefficient with highest grade x^{n-1} to all the others.

If $\mathsf{S} = \mathsf{S}_p$ the coefficient-wise reductions modulo p are computed with a pipelined Mersenne prime reduction algorithm (only adds).



Multiple coefficients can be processed in parallel.

Sample random polynomial in $\mathcal{S}_p$

Variable-weight small polynomials

Two strategy for sampling each small ternary coefficients:

- reduce an 8-bit number mod p via Mersenne prime algorithm: constant execution time, approximated uniform distribution
- rejection of the single invalid encoding of a random 2-bit string: fewer bits from PRNG, perfect uniform distribution

In both cases, the parallel computation of more than one coefficient is limited only by the pressure onto the PRNG.

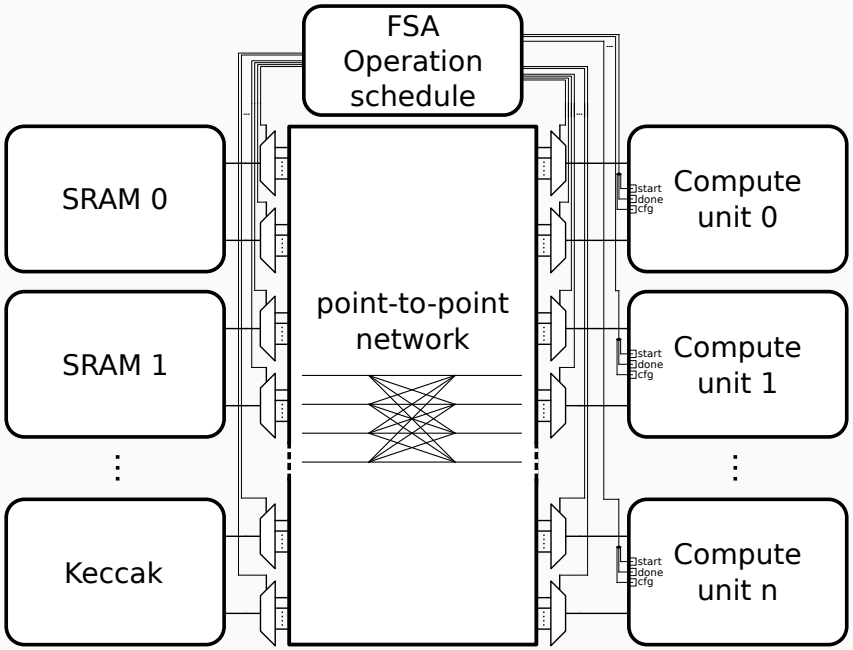
Sample random polynomial in $\mathcal{S}_p$

Fixed-weight small polynomials

Generate a polynomial with the first d coefficients set as 1, and the following d coefficients set as -1, then scramble it

When caches are not in use, the Fisher-Yates shuffle algorithm is safe to use as memory has a constant time access.

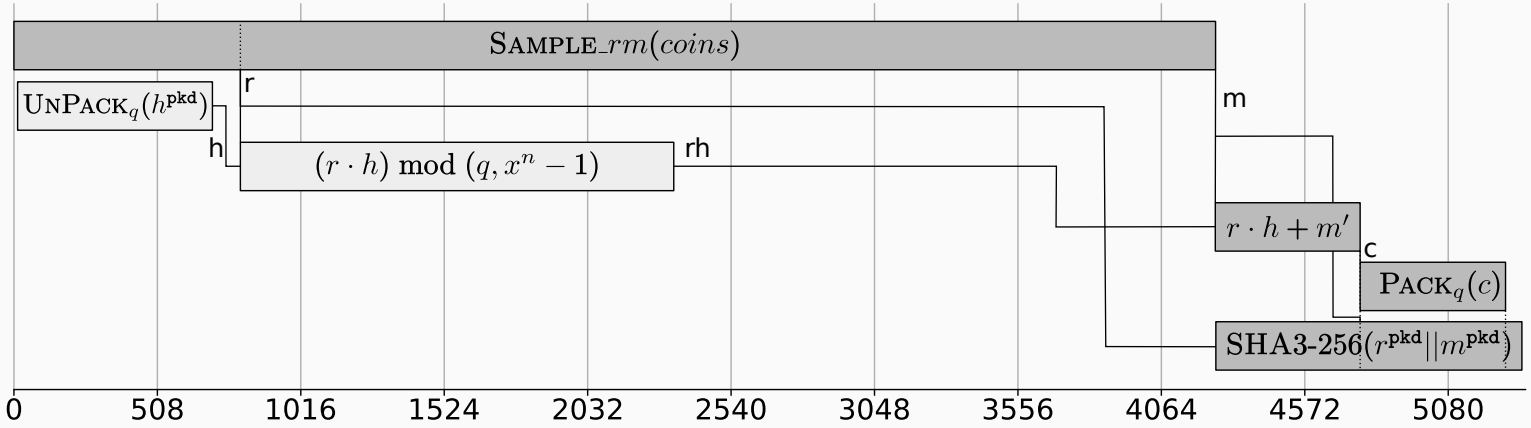
Operations schedule
Top-Level Design configuration



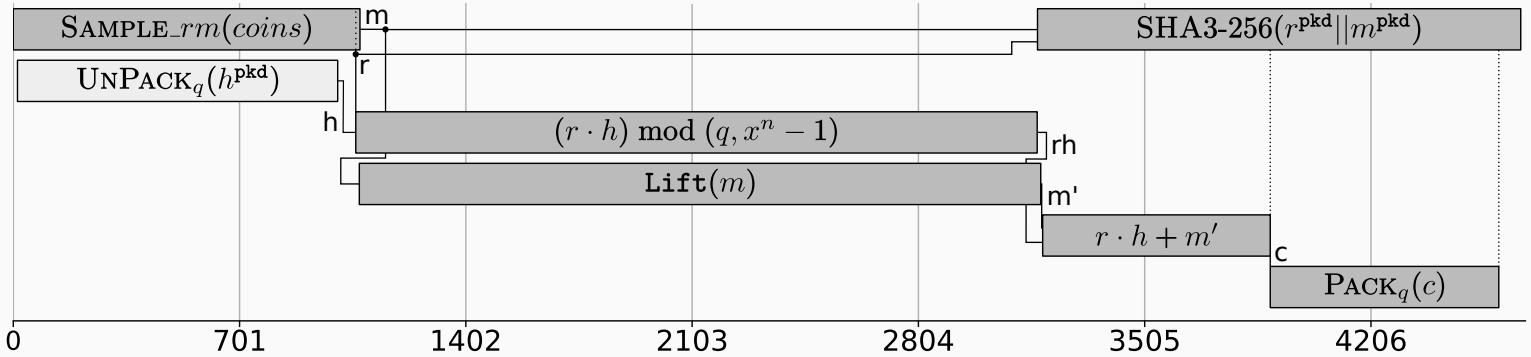
Top-Level Design (TLD) configuration implementing a KEM primitive.

We employed four Simple Dual-Port memories, which are connected in every moment to a specific compute unit (e.g., polynomial multiplier) by global Finite State Automata (FSA) following a fixed schedule of operations.

Operations schedule
Schedule for NTRU.KEM-Encapsulation



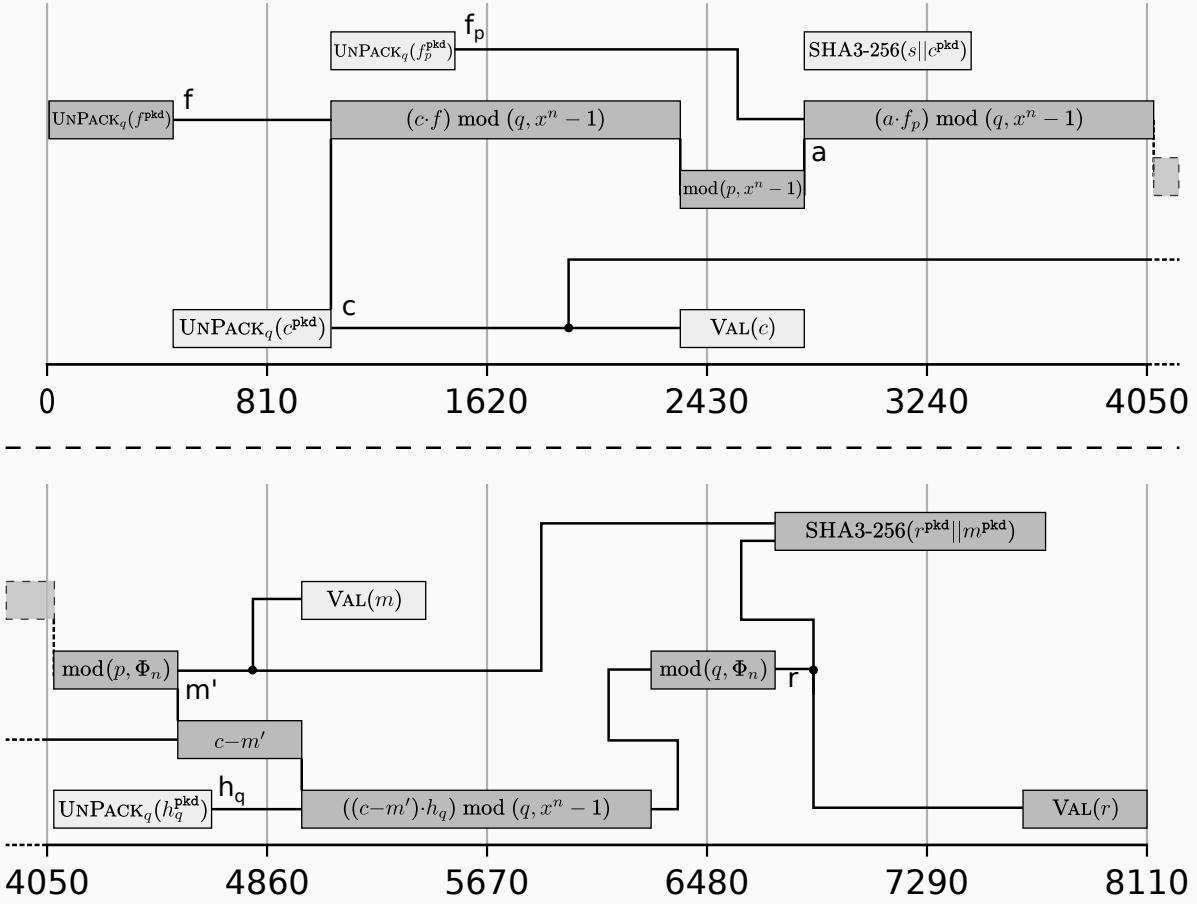
NTRU-HPS ($q = 2048, n = 509$)



NTRU-HRSS ($q = 8192, n = 701$)

Schedule of the NTRU.KEM-Encapsulation with a x-net multiplier. The x axis represents the latency (number of clock cycles).

Operations schedule
Schedule for NTRU.KEM-Decapsulation



NTRU-HPS ($q = 2048, n = 509$)

Schedule of the NTRU.KEM-Decapsulation (HPS variant) with a x-net multiplier. The HRSS variant includes also the Lift operation.
The x axis represents the latency (number of clock cycles).

Experimental results
Design Space Exploration on FPGA

We synthesized the KEM-Encapsulation and KEM-Decapsulation in separate top-level modules on a AMD ZYNQ UltraScale+ FPGA.

Experimental results
Design Space Exploration on FPGA

We synthesized the KEM-Encapsulation and KEM-Decapsulation in separate top-level modules on a AMD ZYNQ UltraScale+ FPGA.

The fully decoupled arithmetic components allowed us to conduct a Design Space Exploration (DSE) considering:

- all the four parameter sets defined by **HPS** and **HRSS** variants
- either **x-net** and **Comba** polynomial multiplier algorithms
- variable-weight sampler based either on **rejection** or **modulo** algorithms
- varying the degree of computing parallelism per arithmetic component

Experimental results
Design Space Exploration on FPGA

We synthesized the KEM-Encapsulation and KEM-Decapsulation in separate top-level modules on a AMD ZYNQ UltraScale+ FPGA.

The fully decoupled arithmetic components allowed us to conduct a Design Space Exploration (DSE) considering:

- all the four parameter sets defined by **HPS** and **HRSS** variants
- either **x-net** and **Comba** polynomial multiplier algorithms
- variable-weight sampler based either on **rejection** or **modulo** algorithms
- varying the degree of computing parallelism per arithmetic component

The solutions with lowest latency and Area-Time product (highest efficiency) were compared with the State of the Art.

Experimental results
Comparison with the state-of-the-art

NTRU.KEM-Encapsulation						
Sec. lvl.	NTRU Variant	Work	Freq.	Area eSlice	Latency µs	AT prod.
1	hps2048509	This work	400	6647	10.9	72.4
3	hps2048677	This work	400	7968	14.8	117
		[14]	250	7642	14.8	113
	hrss701	This work	400	9007	7.2	64.8
		[14]	300	8404	7.4	62.1
	sntrup761	[15]	289	9760	17.3	168
5	hps4096821	This work	400	9318	17.9	166
		[14]	250	9591	18.3	175

NTRU.KEM-Decapsulation						
Sec. lvl.	NTRU Variant	Work	Freq.	Area eSlice	Latency µs	AT prod.
1	hps2048509	This work	350	74640	13.3	992
3	hps2048677	This work	350	98251	17.6	1729
		[14]	300	14067	25.1	353
	hrss701	This work	350	102504	21.7	2224
		[14]	300	16008	29.4	470
	sntrup761	[15]	285	10028	38.6	387
5	hps4096821	This work	350	118965	21.4	2545
		[14]	300	16243	34.0	552

Experimental results
ASIC synthesis

Synthesis using a 40 nm tech library and slow process corner 1.15V@40°C

NTRU.KEM-Encapsulation

Mul. type	NTRU Variant	Area (10^3 µm^2) add	sample	Keccak	q pack	k gen	$\mathcal{R}_p \times \mathcal{R}_q$	q unp.	lift	Total	Latency Freq.	µs
x-net	hps2048509	0.66	2.76	39.12	1.12	2.64	90.40	1.41	-	140.19	700	6.2
	hps2048677	0.68	2.97	39.16	1.11	2.67	120.44	1.41	-	170.44	700	8.4
	hps4096821	0.73	2.95	39.28	0.82	2.68	161.21	1.07	-	211.05	700	10.2
	hrss701	0.83	1.36	40.24	1.26	2.67	148.91	1.54	1.87	201.15	700	4.1
Comba	hps2048509	0.39	2.92	41.56	1.14	1.63	1.22	1.39	-	51.52	750	349.2
	hps2048677	0.42	3.01	40.06	1.13	1.65	1.30	1.40	-	50.30	750	616.1
	hps4096821	0.44	3.00	40.29	0.82	1.67	1.31	1.07	-	49.91	750	904.7
	hrss701	0.47	2.40	40.96	1.35	1.68	1.36	1.52	1.22	52.43	750	660.4

NTRU.KEM-Decapsulation

Mul. type	NTRU Variant	Area (10^3 µm^2) add	Keccak	k_1 gen.	k_2 gen.	$\mathcal{R}_q \times \mathcal{R}_q$ mult.	unpack p	unpack q	validat.	lift	Total	Latency Freq.	µs
x-net	hps2048509	0.78	40.60	0.68	2.67	268.95	1.02	1.54	0.21	-	320.06	650	7.1
	hps2048677	0.81	40.10	0.72	2.70	359.13	1.06	1.52	0.26	-	410.03	650	9.4
	hps4096821	0.81	38.96	0.71	2.66	495.69	1.07	1.17	0.28	-	517.80	650	11.5
	hrss701	0.91	40.71	0.72	2.71	507.23	1.05	1.66	0.25	1.72	561.02	650	11.7
Comba	hps2048509	0.42	41.46	0.67	1.64	1.59	1.08	1.51	0.31	-	51.45	750	1047.1
	hps2048677	0.45	40.70	0.71	1.68	2.38	1.05	1.48	0.29	-	50.74	750	1847.6
	hps4096821	0.46	40.14	0.72	1.68	2.49	1.07	1.17	0.30	-	50.08	750	2713.5
	hrss701	0.50	40.33	0.71	1.69	2.66	1.09	1.63	0.31	1.14	52.26	750	1983.4

HQC: Hamming Quasi-Cyclic

Introduction to HQC

Hamming Quasi-Cyclic (HQC) is a KEM scheme which is relying on the hardness of the syndrome decoding problem on a double circulant quasi-cyclic random code.

HQC doesn't need an efficient decoder for the quasi-cyclic code, hence there's no need to obfuscate a known good code.
This is not the case for BIKE and Classic McEliece, the other two code-based PQC schemes in the final round of the PQC contest.

NIST announced that will standardize HQC as an general-purpose KEM algorithm as an alternative to ML-KEM (CRYSTALS-Kyber).
In comparison to ML-KEM, HQC produces larger ciphertexts and is slightly slower, but is backed by a different hard problem.

The Syndrome Decoding Problem – search variant

- $\mathsf{H} \in \mathbb{F}_q^{(n-k)\times n}$ of a q-ary $[n, k, d]$ public random linear code
- $\mathsf{e} \in \mathbb{F}_q^n$ the secret error vector having Hamming weight $< w$
- $\mathsf{s} = \mathsf{e}\mathsf{H}^\top \in \mathbb{F}_q^{n-k}$ the syndrome associated to the error e

Given $\mathsf{pk} = (\mathsf{H}, \mathsf{s})$, find a vector $\mathsf{e} \in \mathbb{F}_q^n$ such that $\mathsf{s} = \mathsf{e}\mathsf{H}^\top$.
If the weight w of the error e is low enough, there is a single solution which is hard to find.

The double circulant quasi-cyclic variant employs a parity-check matrix composed by two matrices defined just by their first row. The other rows are a cyclic rotation of the previous ones.

The community believes that this variant retains the hardness. Currently the best attacks are the same for the generic codes.

Background

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number

$a = a_0 + a_1 x + \ldots + a_{p-1} x^{p-1} \in$ R stored as vector a$= [a_0, a_1, \ldots, a_{p-1}]$

Background

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number
- $\omega(a)$: Hamming weight of $a \in$ R (number of non-zero coeffs)

$a = a_0 + a_1 x + \ldots + a_{p-1} x^{p-1} \in$ R stored as vector a$= [a_0, a_1, \ldots, a_{p-1}]$

Background

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number
- $\omega(a)$: Hamming weight of $a \in \mathrm{R}$ (number of non-zero coeffs)
- R_w: set of all polynomials in R with Hamming weight w

$a = a_0 + a_1 x + \ldots + a_{p-1} x^{p-1} \in \mathrm{R}$ stored as vector $\mathrm{a} = [a_0, a_1, \ldots, a_{p-1}]$

Moderate Density Parity Check code $\implies w \approx \sqrt{p}$
$a \in \mathrm{R}_w$ is represented as a vector of indexes of non-zero coeffs.

Background

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number
- $\omega(a)$: Hamming weight of $a \in \mathrm{R}$ (number of non-zero coeffs)
- R_w: set of all polynomials in R with Hamming weight w

polynomial addition ($+$)

Coefficient-wise addition: XOR ($\oplus$) boolean operator

$$\begin{array}{rccccccc} a = & a_0 & + & a_1 x & + \ldots + & a_{p-1} x^{p-1} \\ b = & b_0 & + & b_1 x & + \ldots + & b_{p-1} x^{p-1} \\ \hline a + b = (a_0 \oplus b_0) & + & (a_1 \oplus b_1)x & + \ldots + & (a_{p-1} \oplus b_{p-1})x^{p-1} \end{array}$$

$$[a_0 \oplus b_0, a_1 \oplus b_1, \ldots, a_{p-1} \oplus b_{p-1}]$$

Background

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number
- $\omega(a)$: Hamming weight of $a \in \mathrm{R}$ (number of non-zero coeffs)
- R_w: set of all polynomials in R with Hamming weight w

polynomial subtraction $(-)$

Coefficient-wise subtraction: XOR $(\oplus)$ boolean operator

$$\begin{array}{rccccccc} a = & a_0 & + & a_1 x & + \ldots + & a_{p-1} x^{p-1} \\ b = & b_0 & + & b_1 x & + \ldots + & b_{p-1} x^{p-1} \\ \hline a - b = (a_0 \oplus b_0) & + & (a_1 \oplus b_1)x & + \ldots + & (a_{p-1} \oplus b_{p-1})x^{p-1} \end{array}$$

$$[a_0 \oplus b_0, a_1 \oplus b_1, \ldots, a_{p-1} \oplus b_{p-1}]$$

Background

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number
- $\omega(a)$: Hamming weight of $a \in \mathrm{R}$ (number of non-zero coeffs)
- R_w: set of all polynomials in R with Hamming weight w

polynomial multiplication $(\cdot)$

Cyclic convolution: $c_i = \bigoplus_{j+k \equiv i \bmod p} a_j \otimes b_k, \;\; i, j, k \in \{0, \ldots, p-1\}$

$$
\begin{array}{rccccccc}
a = & a_0 & + & a_1 x & + \ldots + & a_{p-1} x^{p-1} \\
b = & b_0 & + & b_1 x & + \ldots + & b_{p-1} x^{p-1} \\
\hline
\text{acc.} = & (a_0 \oplus b_0) & + & (a_0 \oplus b_1) x & + \ldots + & (a_0 \oplus b_{p-1}) x^{p-1}
\end{array}
$$

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number
- $\omega(a)$: Hamming weight of $a \in \mathrm{R}$ (number of non-zero coeffs)
- R_w: set of all polynomials in R with Hamming weight w

polynomial multiplication $(\cdot)$

Cyclic convolution: $c_i = \bigoplus_{j+k \equiv i \bmod p} a_j \otimes b_k, \;\; i, j, k \in \{0, \ldots, p-1\}$

$$
\begin{array}{rccccccc}
a = & a_0 & + & a_1 x & + \ldots + & a_{p-1} x^{p-1} \\
b = & b_0 & + & b_1 x & + \ldots + & b_{p-1} x^{p-1} \\
\hline
\text{acc.} = & (a_0 \oplus b_0) & + & (a_0 \oplus b_1) x & + \ldots + & (a_0 \oplus b_{p-1}) x^{p-1} \\
 & (a_1 \oplus b_{p-1}) & + & (a_1 \oplus b_0) x & + \ldots + & (a_1 \oplus b_{p-2}) x^{p-1}
\end{array}
$$

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number
- $\omega(a)$: Hamming weight of $a \in \mathrm{R}$ (number of non-zero coeffs)
- R_w: set of all polynomials in R with Hamming weight w

polynomial multiplication $(\cdot)$

Cyclic convolution: $c_i = \bigoplus_{j+k \equiv i \bmod p} a_j \otimes b_k, \;\; i, j, k \in \{0, \ldots, p-1\}$

$$
\begin{array}{rccccccc}
a = & a_0 & + & a_1 x & + \ldots + & a_{p-1} x^{p-1} \\
b = & b_0 & + & b_1 x & + \ldots + & b_{p-1} x^{p-1} \\
\hline
\text{acc.} = & (a_0 \oplus b_0) & + & (a_0 \oplus b_1) x & + \ldots + & (a_0 \oplus b_{p-1}) x^{p-1} \\
 & (a_1 \oplus b_{p-1}) & + & (a_1 \oplus b_0) x & + \ldots + & (a_1 \oplus b_{p-2}) x^{p-1} \\
 & \vdots & & \vdots & \vdots & \vdots \\
 & (a_{p-1} \oplus b_1) & + & (a_{p-1} \oplus b_2) x & + \ldots + & (a_{p-1} \oplus b_0) x^{p-1}
\end{array}
$$

Background

Algebraic structure: binary polynomial ring

- R: polynomial ring $\mathbb{F}_2[x]/\langle x^p - 1\rangle$, where p is a prime number
- $\omega(a)$: Hamming weight of $a \in \mathrm{R}$ (number of non-zero coeffs)
- R_w: set of all polynomials in R with Hamming weight w

polynomial multiplication $(\cdot)$

Cyclic convolution: $c_i = \bigoplus_{j+k \equiv i \bmod p} a_j \otimes b_k,\ \ i,j,k \in \{0,\ldots,p-1\}$

$$
\begin{array}{rccccc}
a = & a_0 & + & a_1 x & + \ldots + & a_{p-1} x^{p-1} \\
b = & b_0 & + & b_1 x & + \ldots + & b_{p-1} x^{p-1} \\
\hline
\text{acc.} = & (a_0 \oplus b_0) & + & (a_0 \oplus b_1) x & + \ldots + & (a_0 \oplus b_{p-1}) x^{p-1} \\
 & (a_1 \oplus b_{p-1}) & + & (a_1 \oplus b_0) x & + \ldots + & (a_1 \oplus b_{p-2}) x^{p-1} \\
 & \vdots & & \vdots & \vdots & \vdots \\
 & (a_{p-1} \oplus b_1) & + & (a_{p-1} \oplus b_2) x & + \ldots + & (a_{p-1} \oplus b_0) x^{p-1}
\end{array}
$$

if $a \in \mathrm{R}_w, b \in \mathrm{R}$, asymptotic complexity $\Theta(pw) = \Theta(p\sqrt{p}) = \Theta(p^{1.5})$

Background

Error correction code

- quasi-cyclic random $[2p, p, d]$ code with a public parity-check matrix $\mathsf{H} = [\mathsf{I}_p \mid \mathrm{rot}(h)]$

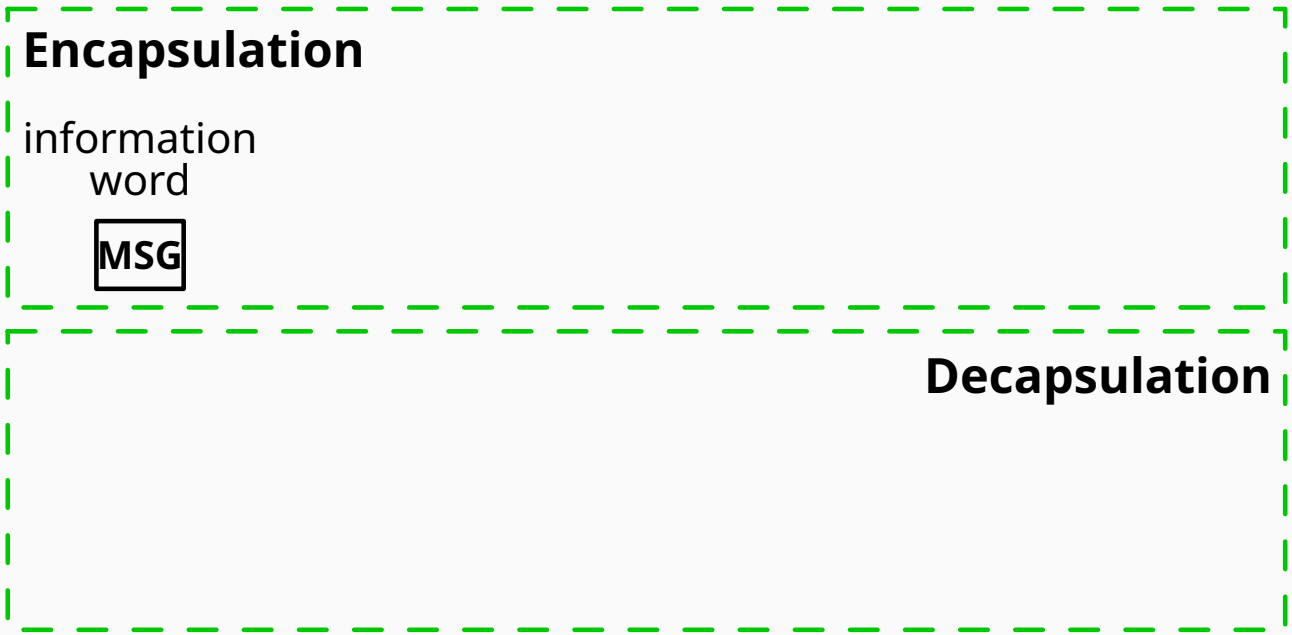
$$\mathsf{H} = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & \cdots & 0 & h_0 & h_{p-1} & h_{p-2} & \cdots & h_1 \\ 0 & 1 & 0 & \cdots & 0 & h_1 & h_0 & h_{p-1} & \cdots & h_2 \\ 0 & 0 & 1 & \cdots & 0 & h_2 & h_1 & h_0 & \cdots & h_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & h_{p-1} & h_{p-2} & h_{p-3} & \cdots & h_0 \end{array}\right]$$

h is a random vector generated from the public key seed

Background

Error correction code(s)

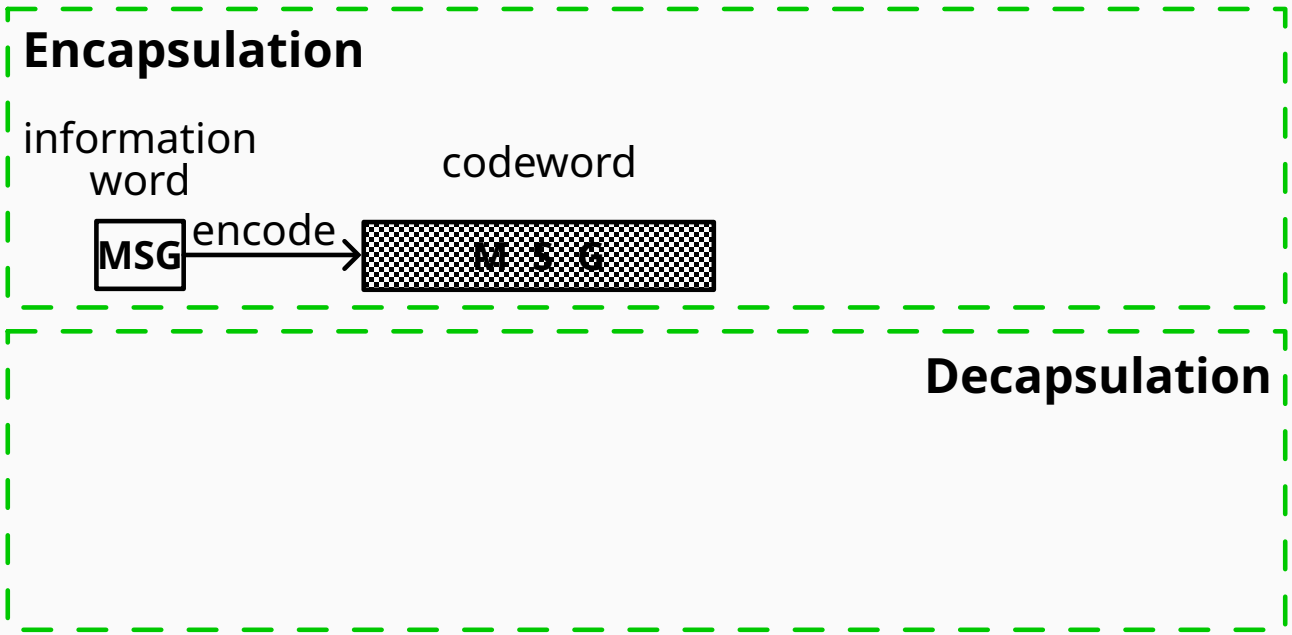
- quasi-cyclic random $[2p, p, d]$ code with a public parity-check matrix $\mathrm{H} = [\mathrm{I}_p \mid \mathrm{rot}(h)]$
- public $[n_e n_i \approx p, k_e k_i, d_e d_i]$ fixed code generated by a shortened Reed-Solomon (RS) $[n_e, k_e, d_e]$ external code with a duplicated Reed-Muller (RM) $[n_i, k_i, d_i]$ internal code.



Background

Error correction code(s)

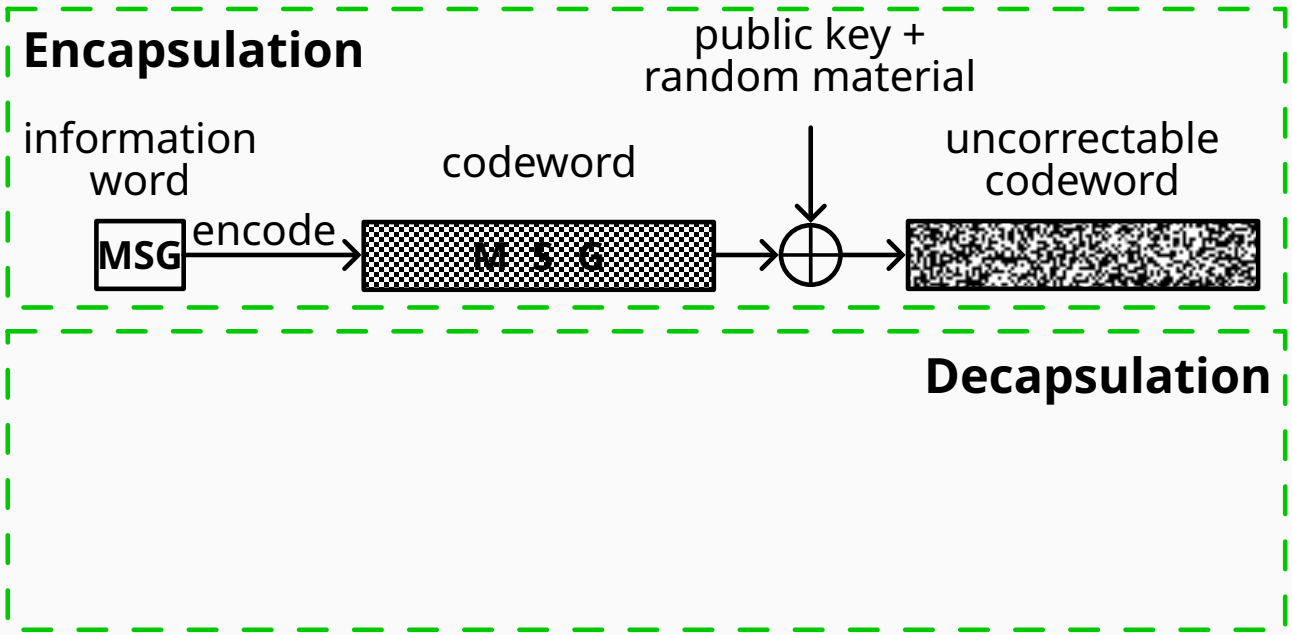
- quasi-cyclic random $[2p, p, d]$ code with a public parity-check matrix $\mathrm{H} = [\mathrm{I}_p \mid \mathrm{rot}(h)]$
- public $[n_e n_i \approx p, k_e k_i, d_e d_i]$ fixed code generated by a shortened Reed-Solomon (RS) $[n_e, k_e, d_e]$ external code with a duplicated Reed-Muller (RM) $[n_i, k_i, d_i]$ internal code.



Background

Error correction code(s)

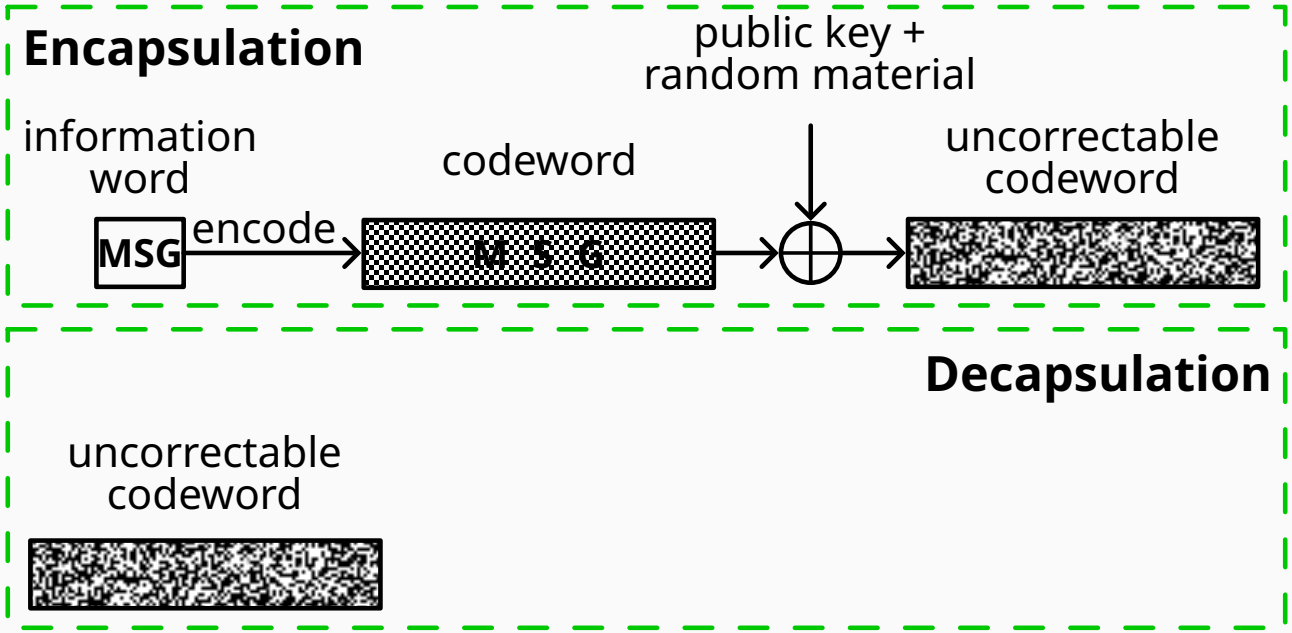
- quasi-cyclic random $[2p, p, d]$ code with a public parity-check matrix $\mathrm{H} = [\mathrm{I}_p \mid \mathrm{rot}(h)]$
- public $[n_e n_i \approx p, k_e k_i, d_e d_i]$ fixed code generated by a shortened Reed-Solomon (RS) $[n_e, k_e, d_e]$ external code with a duplicated Reed-Muller (RM) $[n_i, k_i, d_i]$ internal code.



Background

Error correction code(s)

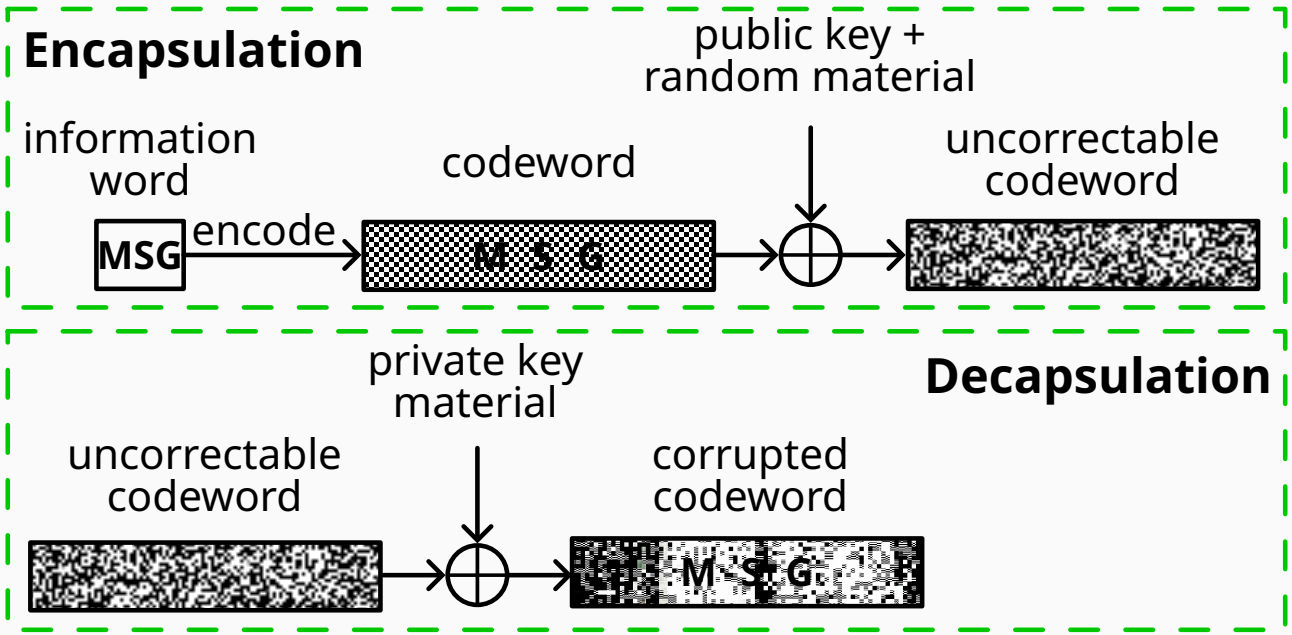
- quasi-cyclic random $[2p, p, d]$ code with a public parity-check matrix $\mathrm{H} = [\mathrm{I}_p \mid \mathrm{rot}(h)]$
- public $[n_e n_i \approx p, k_e k_i, d_e d_i]$ fixed code generated by a shortened Reed-Solomon (RS) $[n_e, k_e, d_e]$ external code with a duplicated Reed-Muller (RM) $[n_i, k_i, d_i]$ internal code.



Background

Error correction code(s)

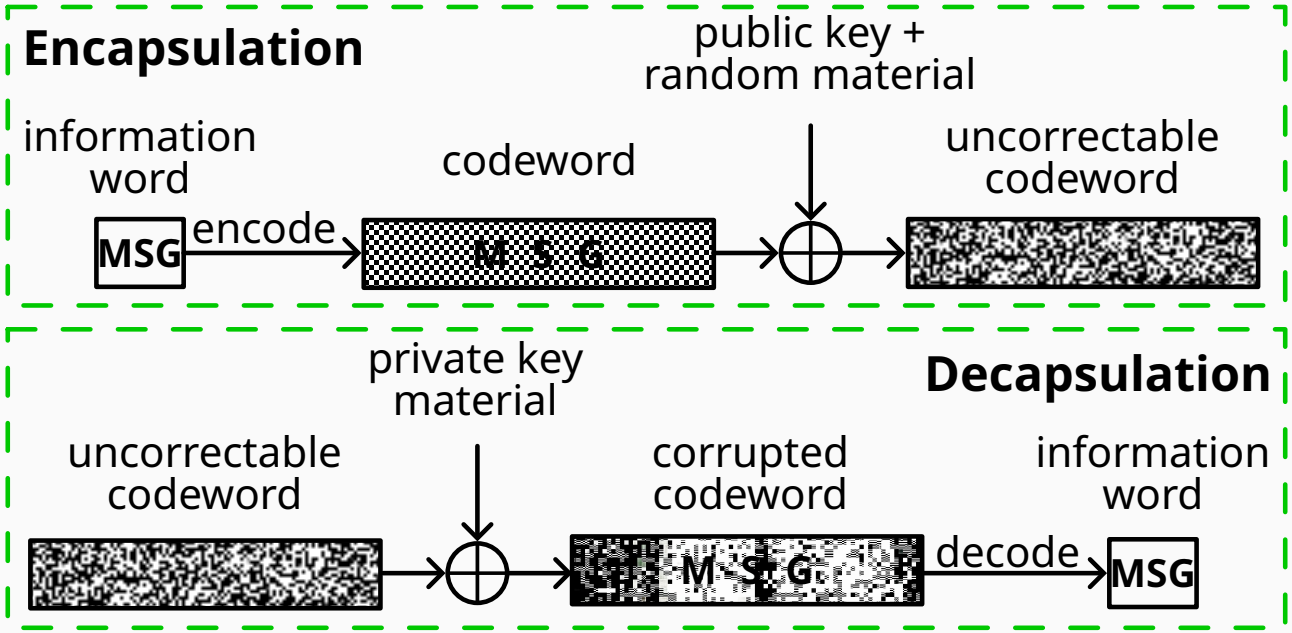
- quasi-cyclic random $[2p, p, d]$ code with a public parity-check matrix $\mathrm{H} = [\mathrm{I}_p \mid \mathrm{rot}(h)]$
- public $[n_e n_i \approx p, k_e k_i, d_e d_i]$ fixed code generated by a shortened Reed-Solomon (RS) $[n_e, k_e, d_e]$ external code with a duplicated Reed-Muller (RM) $[n_i, k_i, d_i]$ internal code.



Background

Error correction code(s)

- quasi-cyclic random $[2p, p, d]$ code with a public parity-check matrix $\mathrm{H} = [\mathrm{I}_p \mid \mathrm{rot}(h)]$
- public $[n_e n_i \approx p, k_e k_i, d_e d_i]$ fixed code generated by a shortened Reed-Solomon (RS) $[n_e, k_e, d_e]$ external code with a duplicated Reed-Muller (RM) $[n_i, k_i, d_i]$ internal code.



HQC KEM scheme

HQC.KEM-KeyGeneration

Require: None

Ensure: $\mathsf{pk} = (\varphi \in \{0,1\}^{320}, s \in \mathsf{R})$
$\mathsf{sk} = (\gamma \in \{0,1\}^{320}, \sigma \in \{0,1\}^{k}, \varphi \in \{0,1\}^{320}, s \in \mathsf{R})$

1: $\sigma \xleftarrow{\$} \{0,1\}^{k}$
2: $(\gamma, \varphi) \xleftarrow{\$} \{0,1\}^{320} \times \{0,1\}^{320}$
3: $h \leftarrow \mathsf{CSPRNG}(\varphi, \mathsf{R})$ ▷ *Sample a random dense poly*
4: $(x, y) \leftarrow \mathsf{CSPRNG}(\gamma, \mathsf{R}_w \times \mathsf{R}_w)$ ▷ *Sample two random sparse polys*
5: $s \leftarrow x + h \cdot y$ ▷ *Sparse poly by a dense poly multiplication*
6: **return** $\mathsf{pk} = (\varphi, s)$, $\mathsf{sk} = (\gamma, \sigma, \varphi, s)$

HQC KEM scheme

HQC.KEM-Encapsulation

Require: $\mathsf{pk}=(\varphi \in \{0,1\}^{320}, s \in \mathsf{R})$

Ensure: $\mathsf{ctx}=(u \in \mathsf{R}, \mathsf{v} \in \mathbb{F}_2^{n_e n_i}, \mathrm{salt} \in \{0,1\}^{128}), K \in \{0,1\}^{512}$

1: $\mathsf{m} \xleftarrow{\$} \{0,1\}^k$, $\mathrm{salt} \xleftarrow{\$} \{0,1\}^{128}$

2: $\theta \leftarrow \mathsf{Hash}_G(\mathsf{m} \| \varphi \| s \| \mathrm{salt})$

3: $(e, r_a, r_b) \leftarrow \mathsf{CSPRNG}(\theta, \mathsf{R}_{w_e} \times \mathsf{R}_{w_r} \times \mathsf{R}_{w_r})$ ▷ *Sample three random sparse polys*

4: $h \leftarrow \mathsf{CSPRNG}(\varphi, \mathsf{R})$ ▷ *Sample a random dense poly*

5: $u \leftarrow r_a + h \cdot r_b$ ▷ *Sparse poly by a dense poly multiplication*

6: $\mathsf{v} \leftarrow \mathsf{Encode}_G(\mathsf{m}) + \mathsf{Tr}\left(s \cdot r_b + e\right)$ ▷ *RMRS encoding, sparse poly by a dense poly mul*

7: $K \leftarrow \mathsf{Hash}_K(\mathsf{m} \| u \| \mathsf{v})$

8: **return** $\mathsf{ctx} = (u, \mathsf{v}, \mathrm{salt}), K$

HQC KEM scheme

HQC.KEM-Decapsulation

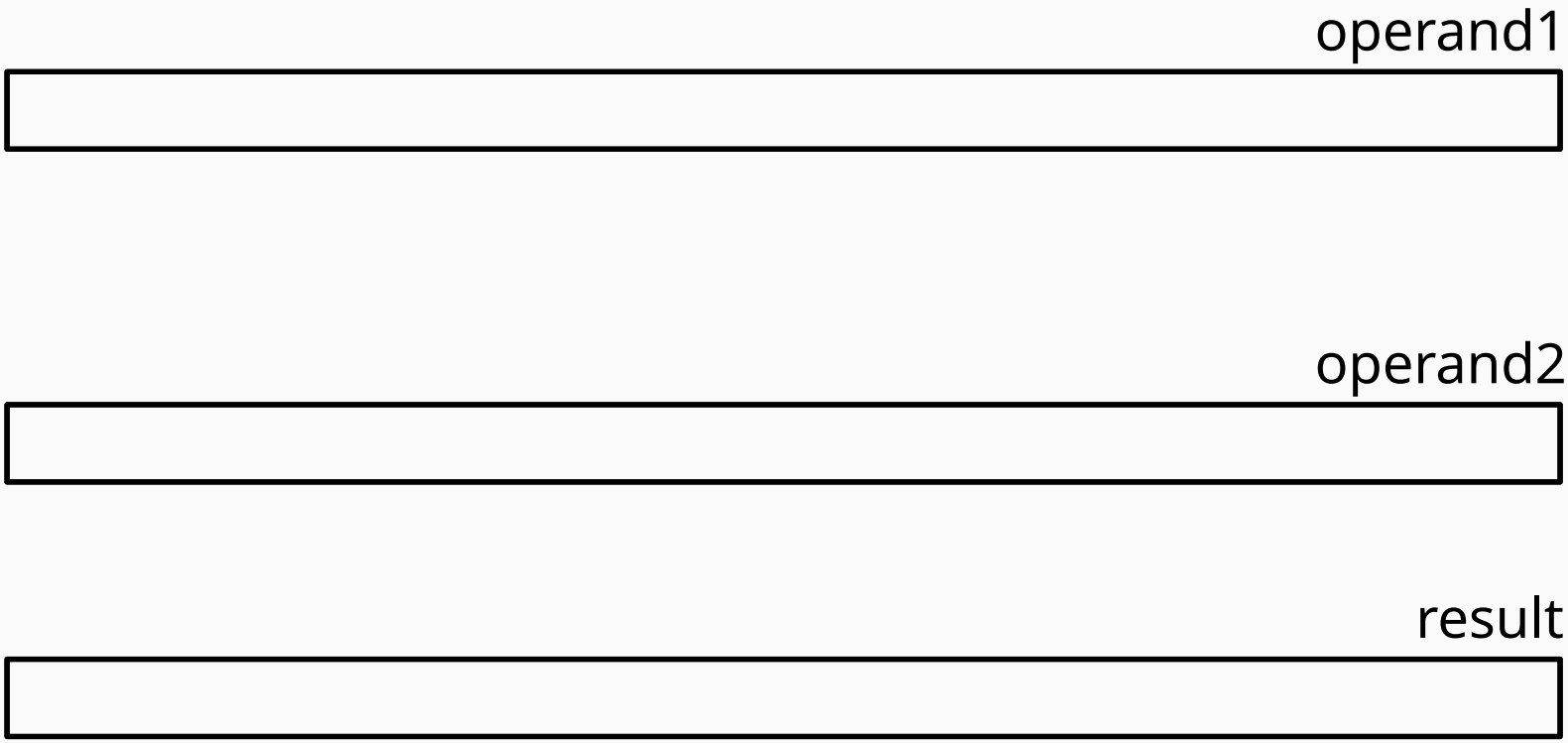
Require: $\mathsf{ctx} = (u \in \mathsf{R}, \mathsf{v} \in \mathbb{F}_2^{n_e n_i}, \mathrm{salt} \in \{0,1\}^{128})$
$\mathsf{sk} = (\gamma \in \{0,1\}^{320}, \sigma \in \{0,1\}^{k}, \varphi \in \{0,1\}^{320}, s \in \mathsf{R})$

Ensure: $K \in \{0,1\}^{512}$

1: $(x, y) \leftarrow \mathsf{CSPRNG}(\gamma, \mathsf{R}_w \times \mathsf{R}_w)$ ▷ *Sample two random sparse polys*
2: $\mathsf{m}' \leftarrow \mathsf{Decode}_G(\mathsf{Tr}([\mathsf{v} \| 0^{p-n_e n_i}] - u \cdot y))$ ▷ *RMRS decoding, sparse poly by a dense poly mul*
3: $\theta' \leftarrow \mathsf{Hash}_G(\mathsf{m}' \| \varphi \| s \| \mathrm{salt})$
4: $(e, r_a, r_b) \leftarrow \mathsf{CSPRNG}(\theta, \mathsf{R}_{w_e} \times \mathsf{R}_{w_r} \times \mathsf{R}_{w_r})$ ▷ *Sample three random sparse polys*
5: $h \leftarrow \mathsf{CSPRNG}(\varphi, \mathsf{R})$ ▷ *Sample a random dense poly*
6: $u \leftarrow r_a + h \cdot r_b$ ▷ *Sparse poly by a dense poly multiplication*
7: $\mathsf{v} \leftarrow \mathsf{Encode}_G(\mathsf{m}) + \mathsf{Tr}(s \cdot r_b + e)$ ▷ *RMRS encoding, sparse poly by a dense poly mul*
8: **if** $(\mathsf{ctx} \neq \mathsf{ctx}')$ $K' \leftarrow \mathsf{Hash}_K(\sigma \| u \| \mathsf{v})$ **else** $K' \leftarrow \mathsf{Hash}_K(\mathsf{m}' \| u \| \mathsf{v})$
9: **return** K'

Polynomial adder
Addition/subtraction $R \times R \mapsto R$

In case both operands are in R:



Polynomial adder
Addition/subtraction $\mathrm{R} \times \mathrm{R} \mapsto \mathrm{R}$

In case both operands are in R:

- access data in blocks of $B = 128$ bits

operand1

0	1	2	3	4	5	6	7	8	9	10	11

operand2

0	1	2	3	4	5	6	7	8	9	10	11

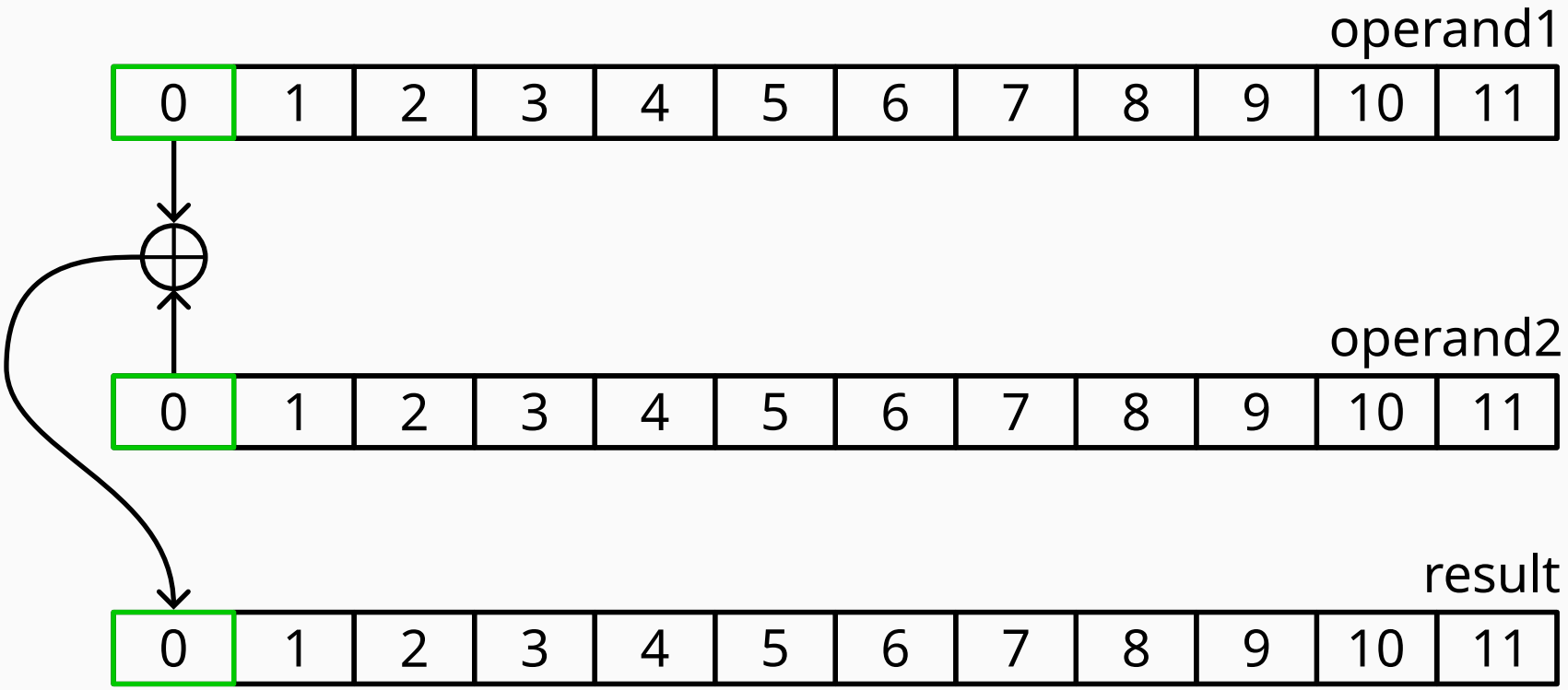
result

0	1	2	3	4	5	6	7	8	9	10	11

Polynomial adder
Addition/subtraction $\mathrm{R} \times \mathrm{R} \mapsto \mathrm{R}$

In case both operands are in R:

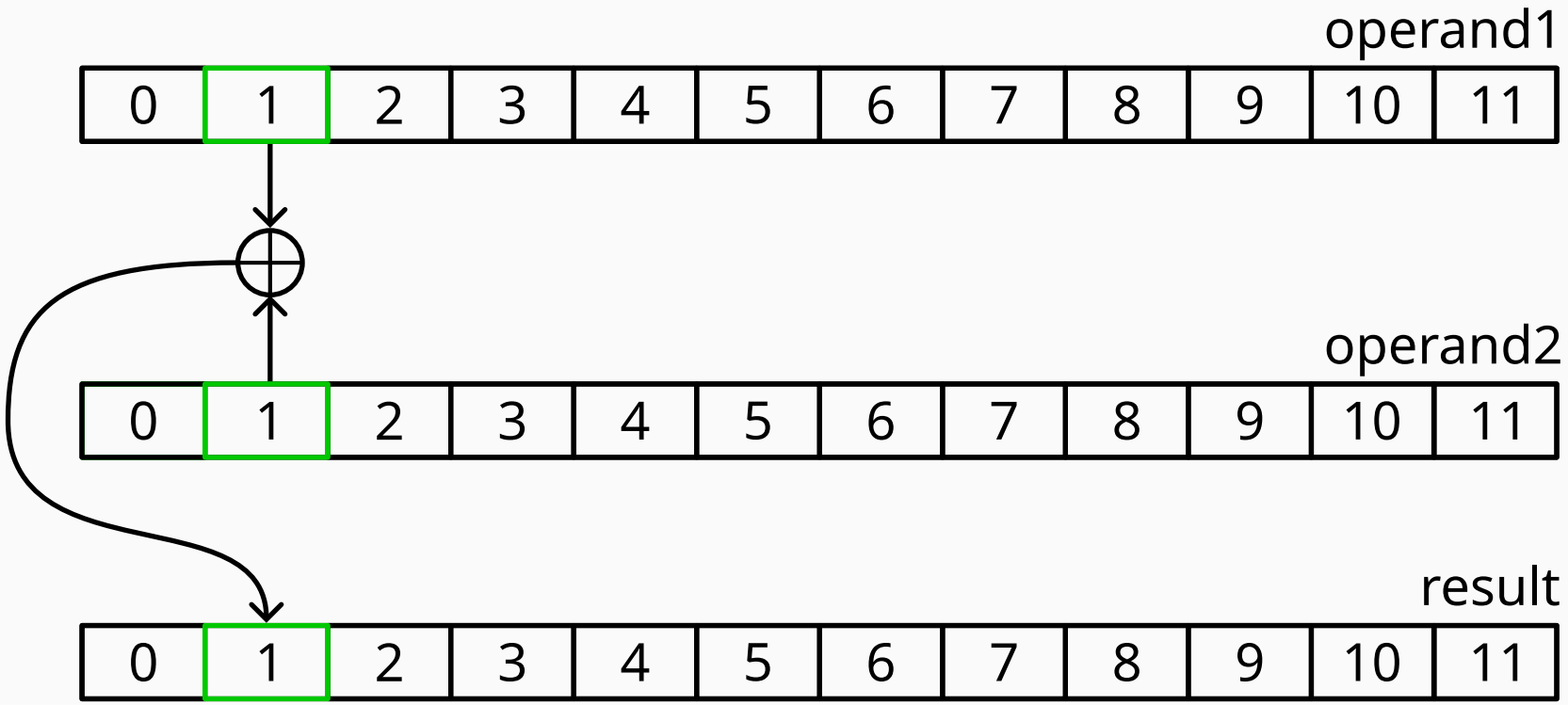
- access data in blocks of $B = 128$ bits
- perform the XOR operation block-wise



Polynomial adder
Addition/subtraction $\mathrm{R} \times \mathrm{R} \mapsto \mathrm{R}$

In case both operands are in R:

- access data in blocks of $B = 128$ bits
- perform the XOR operation block-wise



Polynomial adder
Addition $R_w \times R \mapsto R$

In case operand1 in R_w and operand2 is in R:
For each index i in the vector of operand1:

operand1

544	284	302	1402	239	819	265	1053

operand2/result

0	1	2	3	4	5	6	7	8	9	10	11

Polynomial adder
Addition $R_w \times R \mapsto R$

In case operand1 in R_w and operand2 is in R:
For each index i in the vector of operand1:

- determine the operand2 block index as $\lfloor i/B \rfloor$

operand1

544	284	302	1402	239	819	265	1053

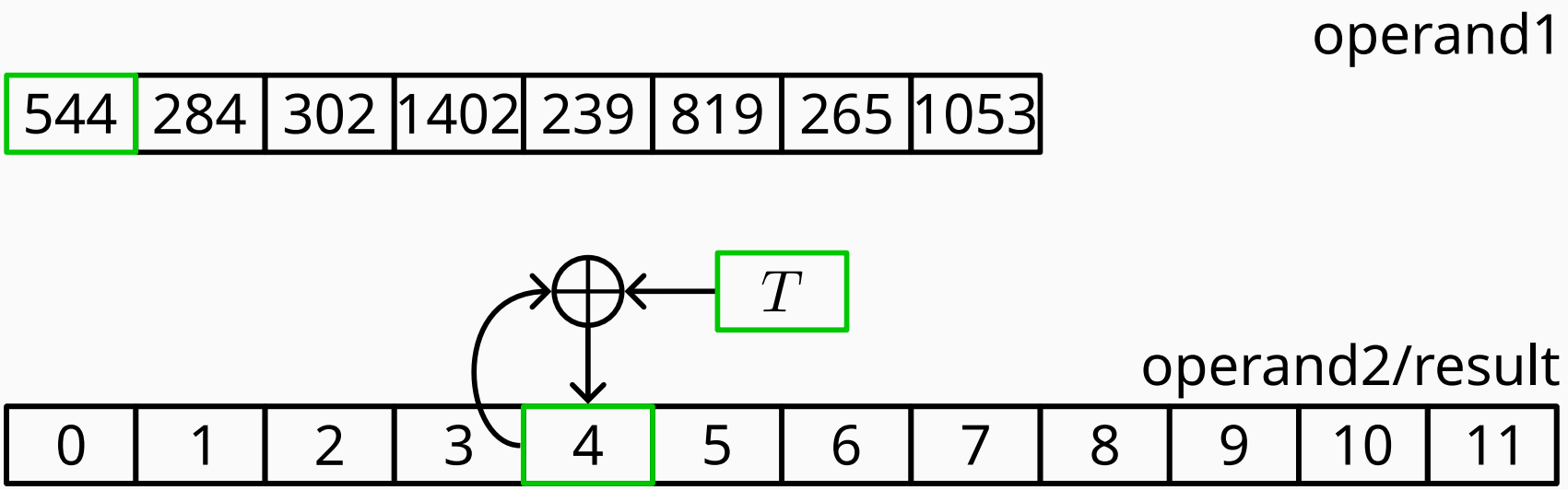
operand2/result

0	1	2	3	4	5	6	7	8	9	10	11

Polynomial adder
Addition $R_w \times R \mapsto R$

In case operand1 in R_w and operand2 is in R:
For each index i in the vector of operand1:

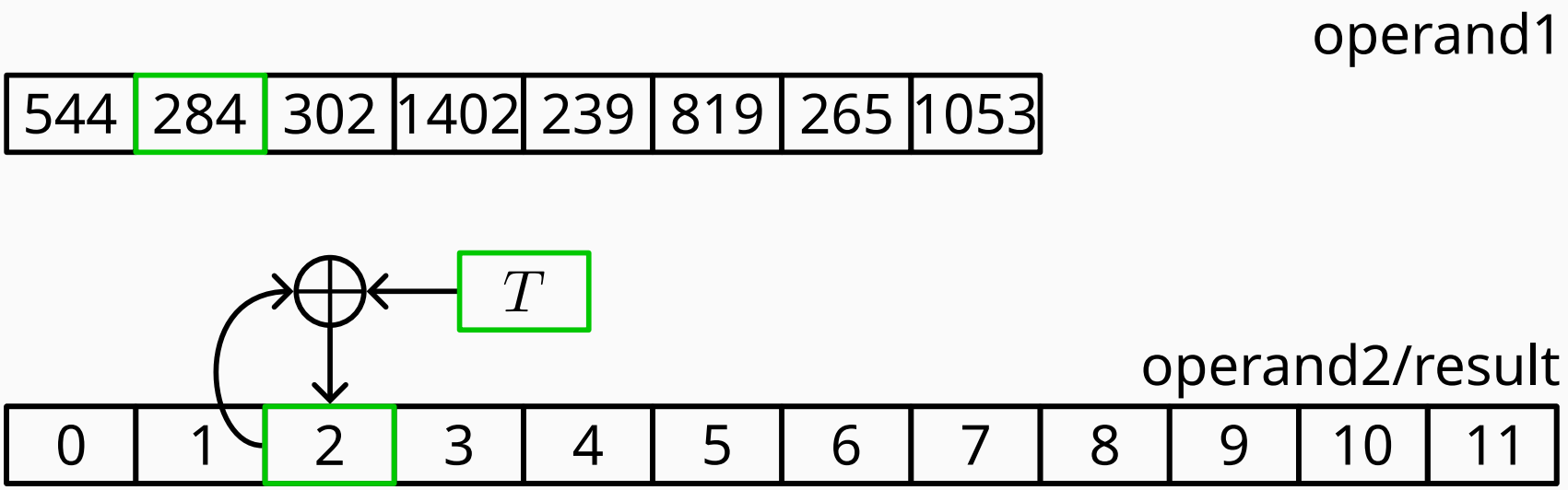
- determine the operand2 block index as $\lfloor i/B \rfloor$
- flip a single bit of that block by generating $T = 1 \ll (i \bmod B)$



Polynomial adder
Addition $R_w \times R \mapsto R$

In case operand1 in R_w and operand2 is in R:
For each index i in the vector of operand1:

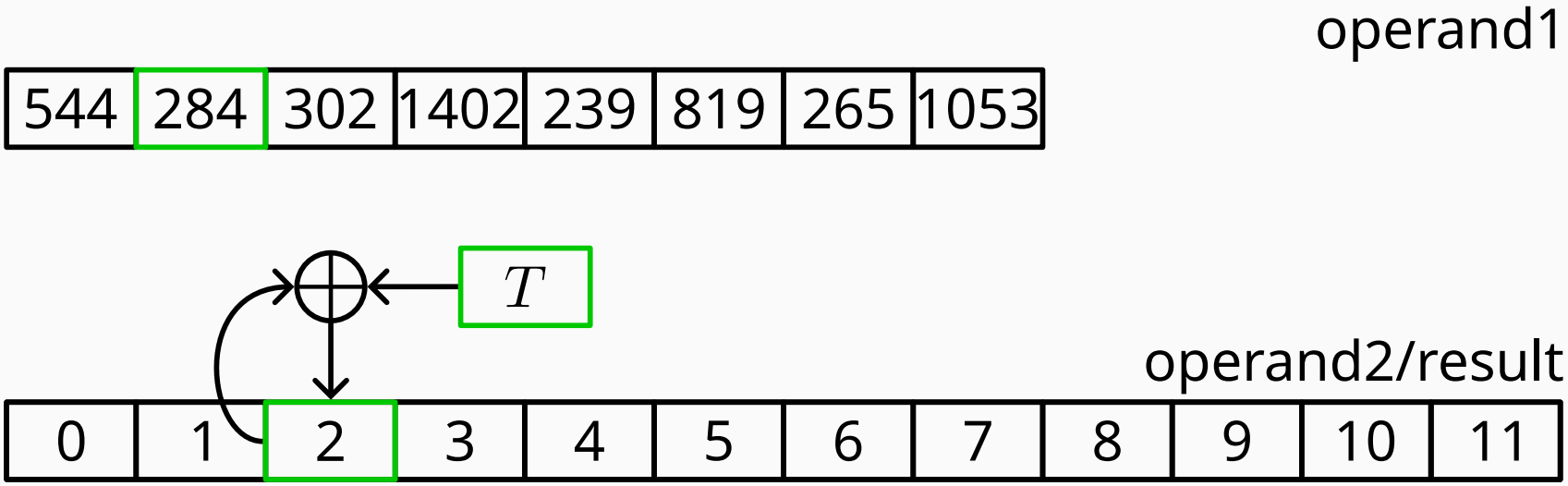
- determine the operand2 block index as $\lfloor i/B \rfloor$
- flip a single bit of that block by generating $T = 1 \ll (i \bmod B)$



Polynomial adder
Addition $R_w \times R \mapsto R$

In case operand1 in R_w and operand2 is in R:
For each index i in the vector of operand1:

- determine the operand2 block index as $\lfloor i/B \rfloor$
- flip a single bit of that block by generating $T = 1 \ll (i \bmod B)$
- cannot be easily pipelined due to read-after-write dependency!



Polynomial multiplier: $\mathrm{R} \times \mathrm{R}_w$

- One operand is always in R_w
- The low weight of polynomial ($\approx \sqrt{p}$) makes the schoolbook shift-and-add approach interesting: $\Theta(p^{1.5})$ asymptotic complexity

Polynomial multiplier: $\mathrm{R} \times \mathrm{R}_w$

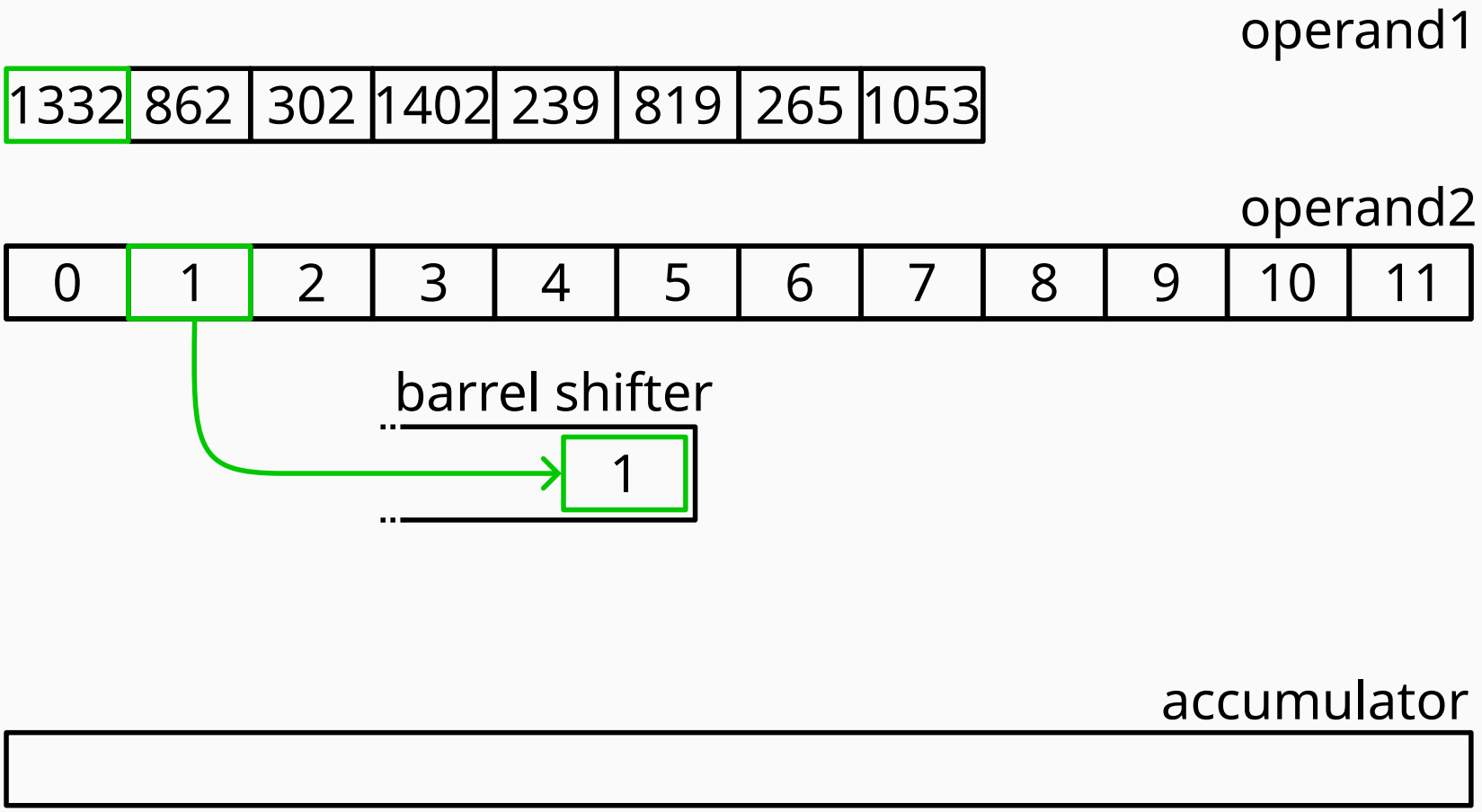
- One operand is always in R_w
- The low weight of polynomial ($\approx \sqrt{p}$) makes the schoolbook shift-and-add approach interesting: $\Theta(p^{1.5})$ asymptotic complexity

- There are faster algorithms based on the NTT with better asymptotic complexity, but:
  - the polynomial ring is not compatible with any NTT algorithm
  - memory access pattern is challenging to optimize

Polynomial multiplier: $R \times R_w$
Single index processed

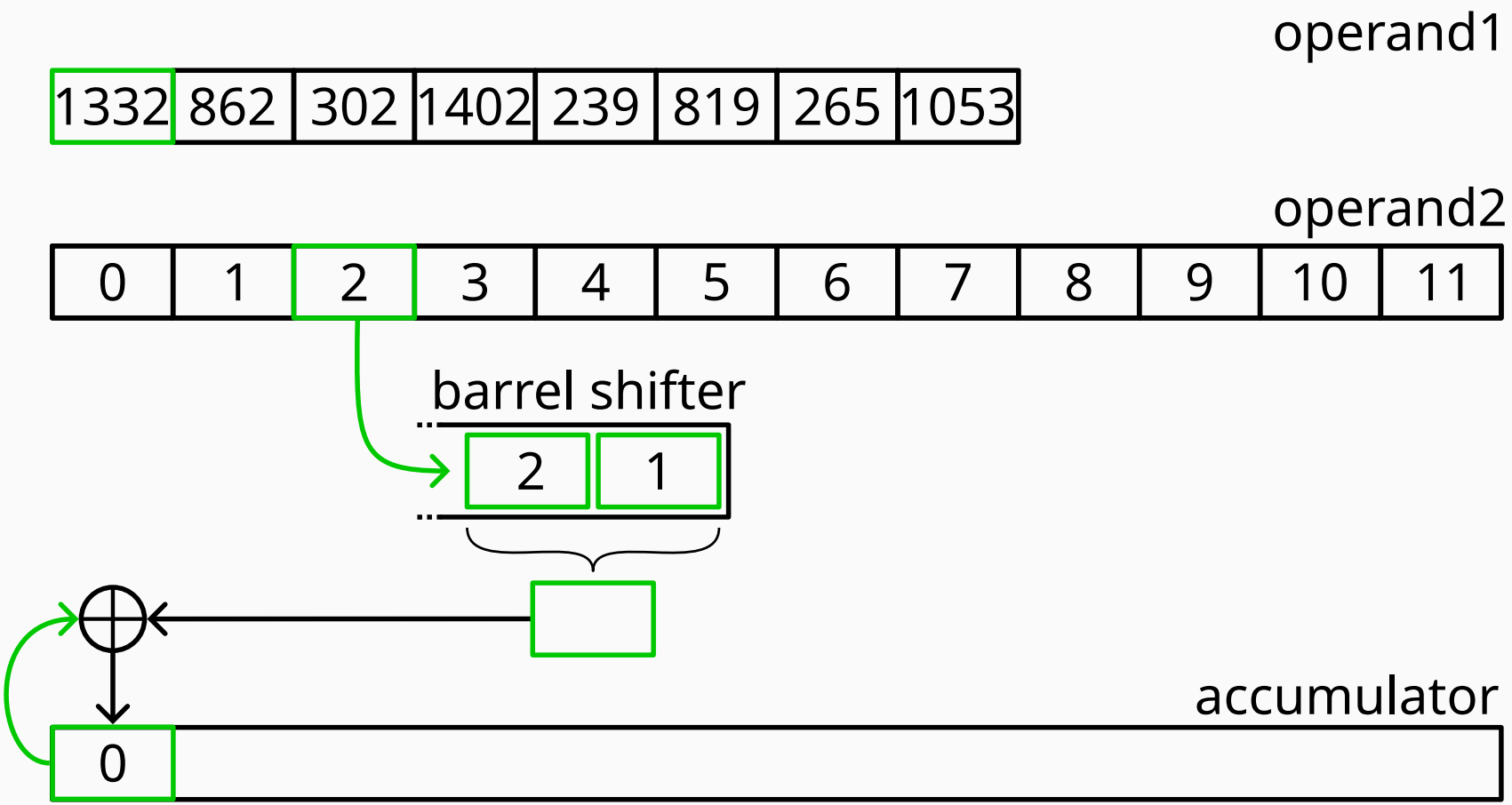
$$\text{start block} = \lfloor (p - i)/B \rfloor$$

$$\text{shift amount} = \lfloor (p - i) \bmod B \rfloor$$



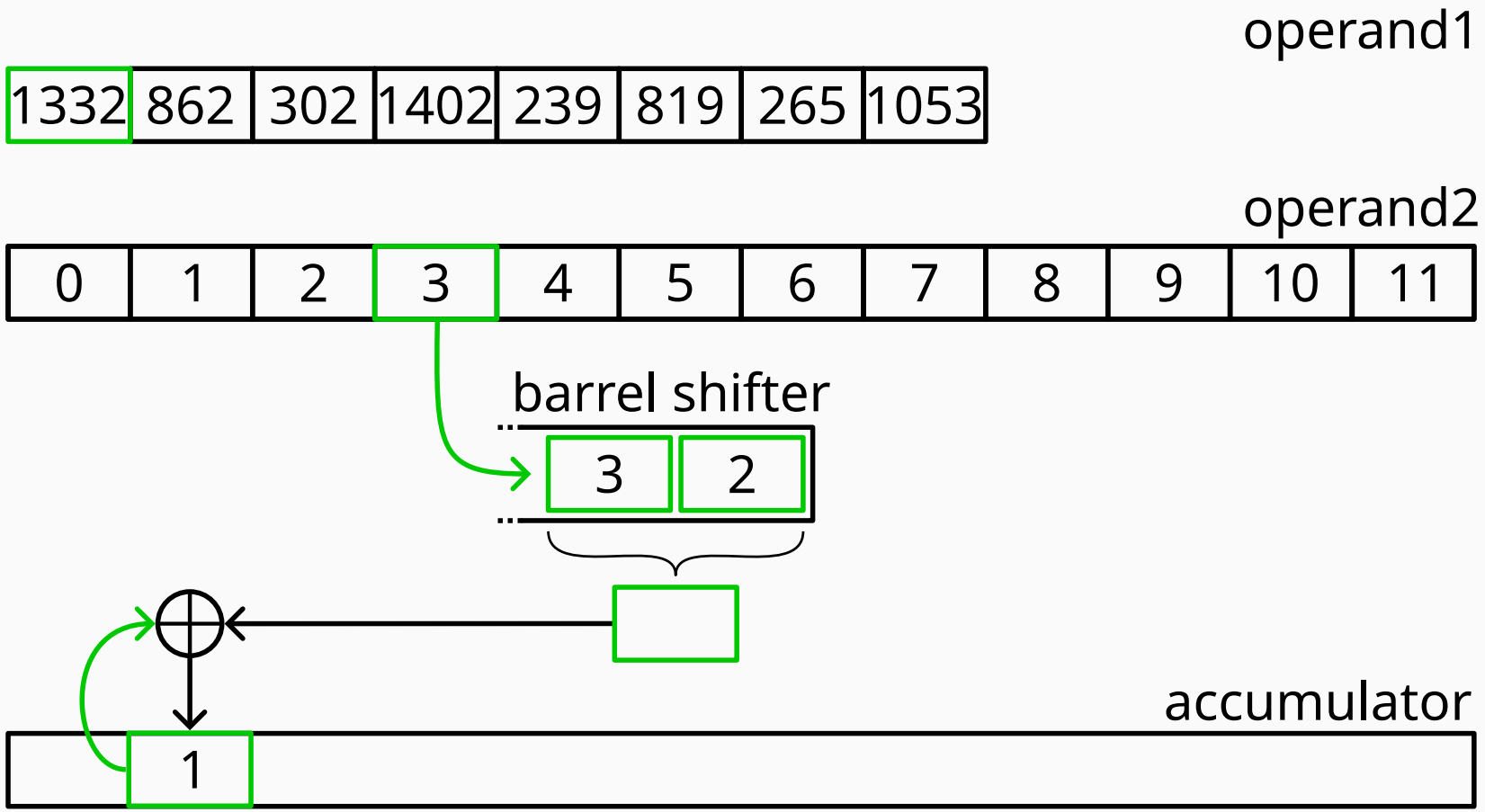
Polynomial multiplier: $R \times R_w$
Single index processed

$$\text{start block} = \lfloor (p - i)/B \rfloor$$
$$\text{shift amount} = \lfloor (p - i) \bmod B \rfloor$$



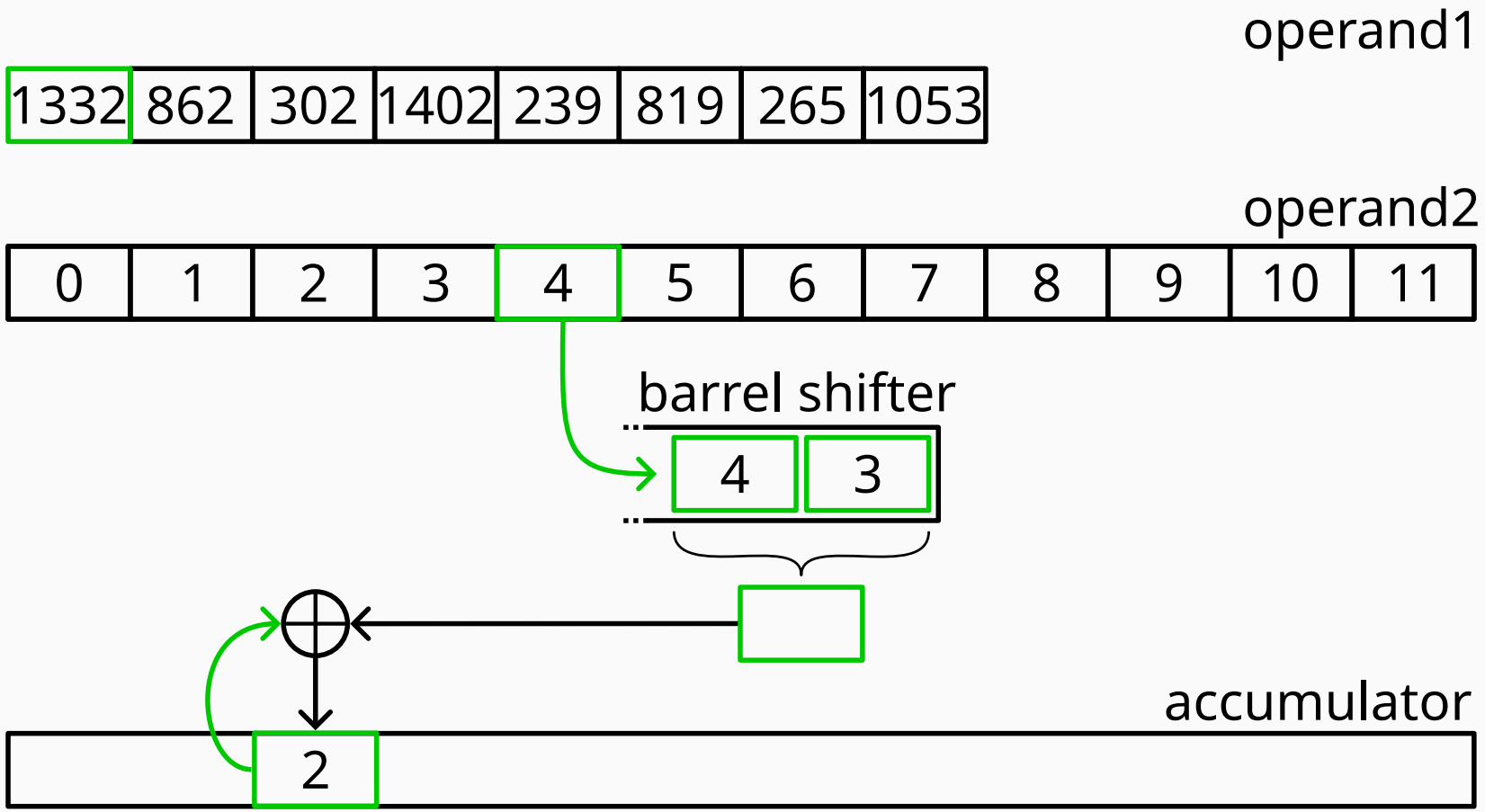
Polynomial multiplier: $R \times R_w$
Single index processed

$$\text{start block} = \lfloor (p - i)/B \rfloor$$
$$\text{shift amount} = \lfloor (p - i) \bmod B \rfloor$$



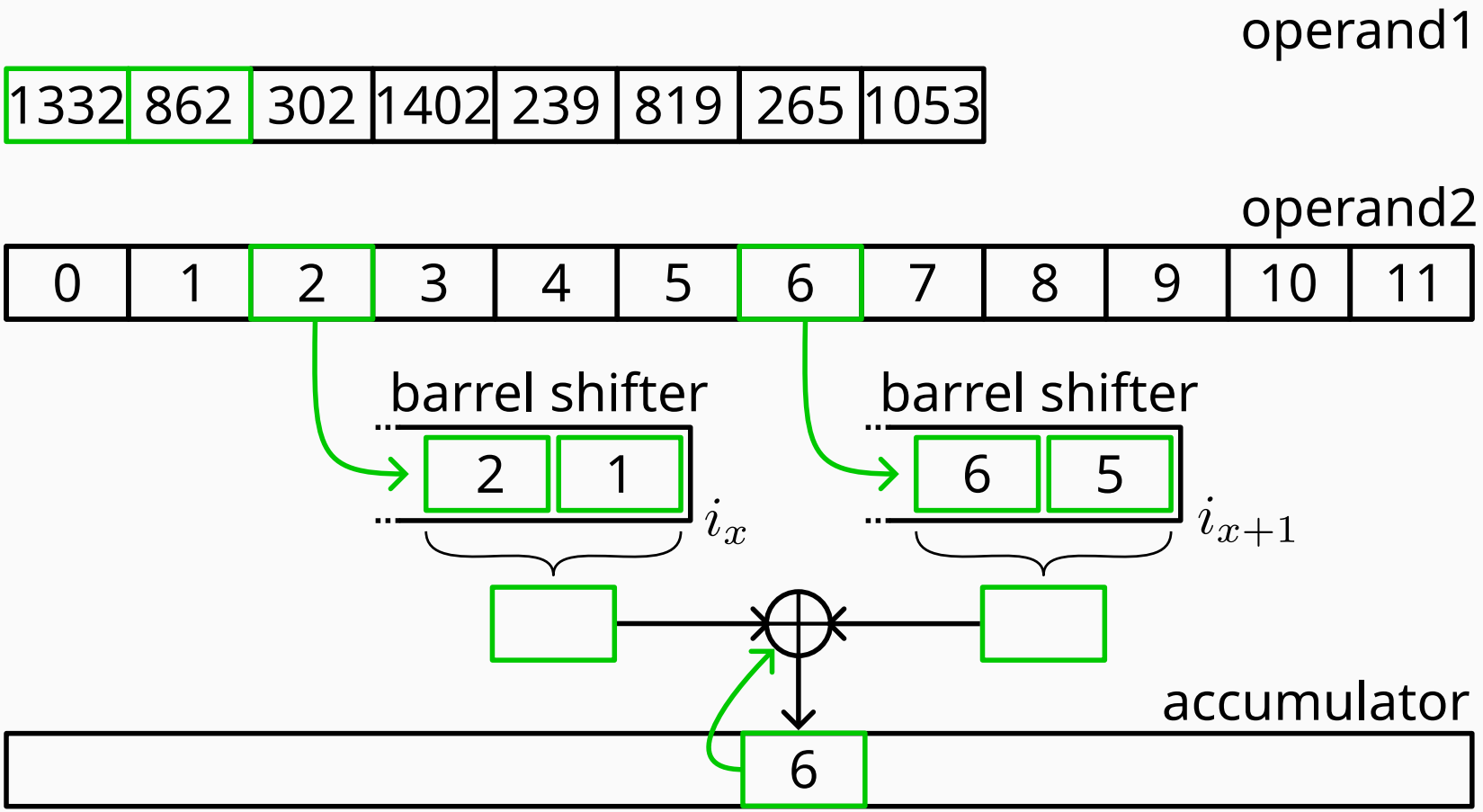
Polynomial multiplier: $R \times R_w$
Single index processed

$$\text{start block} = \lfloor (p - i)/B \rfloor$$
$$\text{shift amount} = \lfloor (p - i) \bmod B \rfloor$$



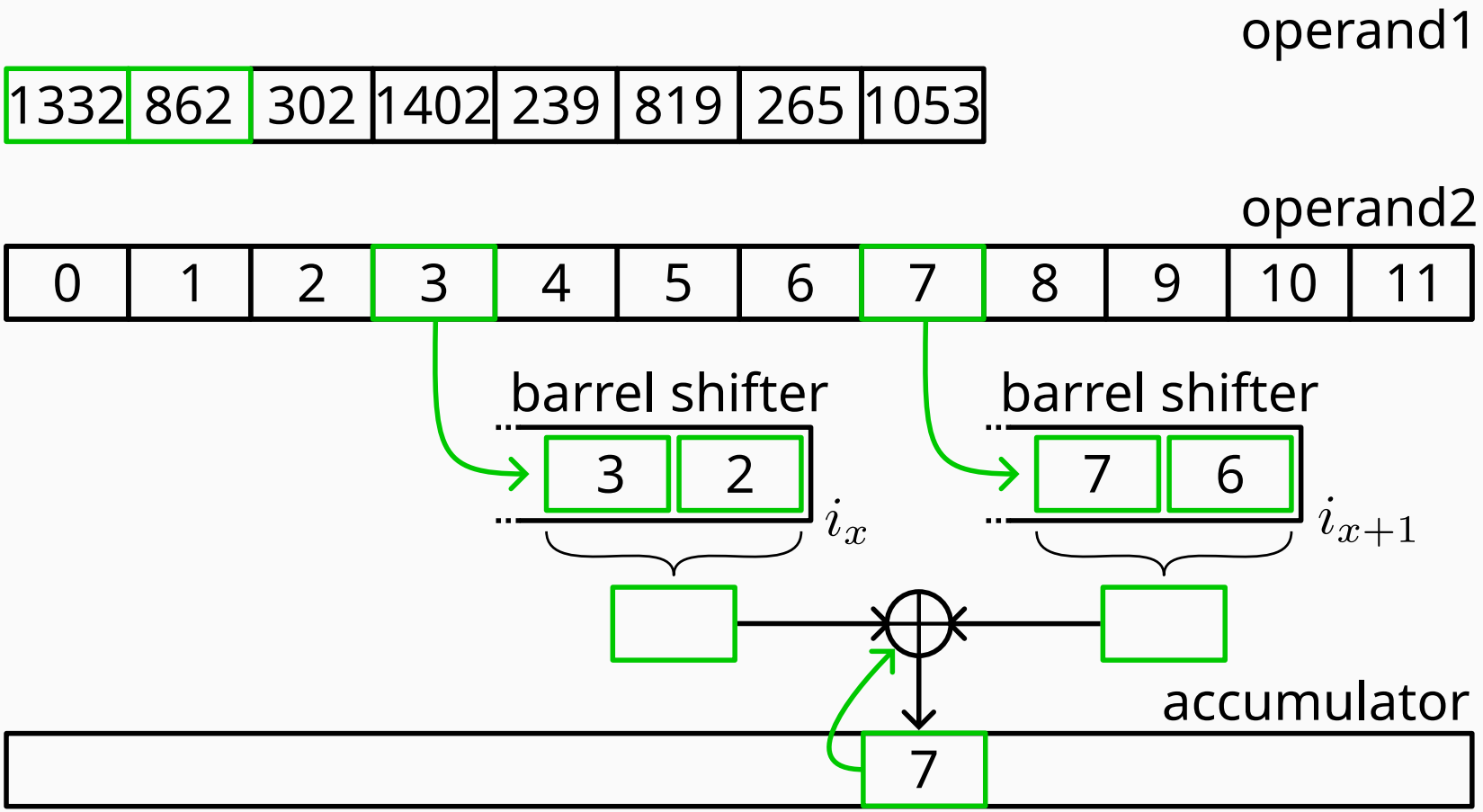
Polynomial multiplier: $R \times R_w$
Multiple indexes processed

$$\text{start block} = \lfloor (p - i)/B \rfloor$$
$$\text{shift amount} = \lfloor (p - i) \bmod B \rfloor$$



Polynomial multiplier: $R \times R_w$
Multiple indexes processed

$$\text{start block} = \lfloor (p - i)/B \rfloor$$
$$\text{shift amount} = \lfloor (p - i) \bmod B \rfloor$$



Sample random polynomials uniformly

Dense random polynomials in R

The elements of the binary vector $h \in R$ are generated by the SHAKE-256 algorithm (a SHA-3 eXtensible Output Function) expanding the small 320-bits public seed.

The output is trimmed to the correct bit size, padding with zero bits if the last block is underfilled.

Sample random polynomials uniformly

Sparse random polynomials in R_w

The HQC specification uses the constant-time algorithm from [16]:

- runs in constant-time
- uses of an exact amount of randomness ($32 \cdot w$ bits)
- requires a modulo operations between a 32-bit dividend and a generic 16-bit divisor

We used a straightforward *shift-and-subtract* pipelined algorithm, not requiring DSPs to perform the operation.

At synthesis time the number of pipeline stages can be selected to balance resources usage and timing closure.

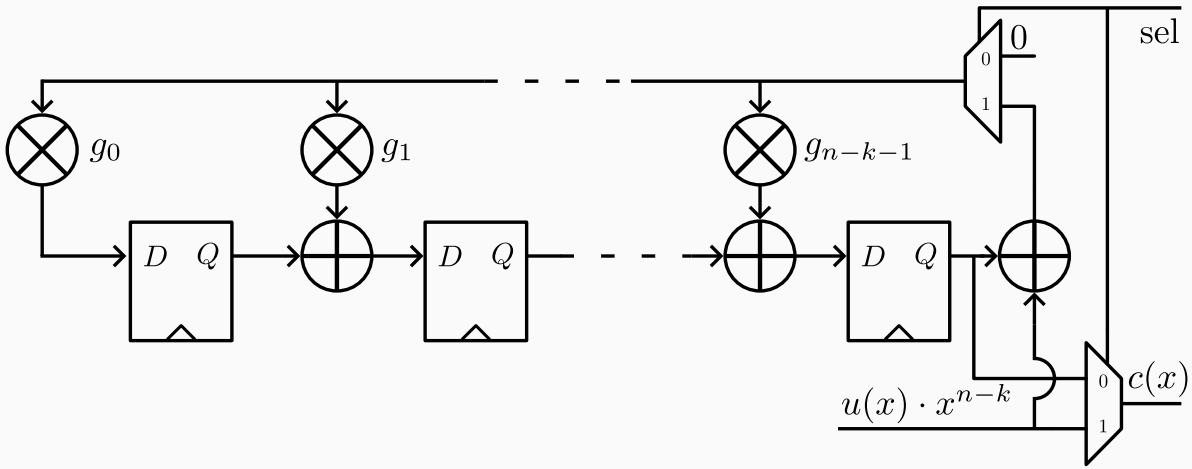
Public code: Reed Solomon
Encoder

The code treats a block of data as a set of $\mathbb{F}_{2^8}$ elements (symbols).

In a systematic encoding procedure the sequence of symbols of the message polynomial $u(x)$ are the prefix of the codeword, and the error correcting symbols are the suffix:

$$c(x) = x^{n_e - k_e} u(x) - \left(x^{n_e - k_e} u(x) \bmod g(x) \right)$$

A simple way to produce such special encoding is through a Linear Feedback Shift Register [17]:



Public code: Reed Solomon
Decoder

Consider a valid codeword $c(x)$ affected by an unknown error $e(x)$ which has up to t terms:

$$r(x) = c(x) + e(x)$$

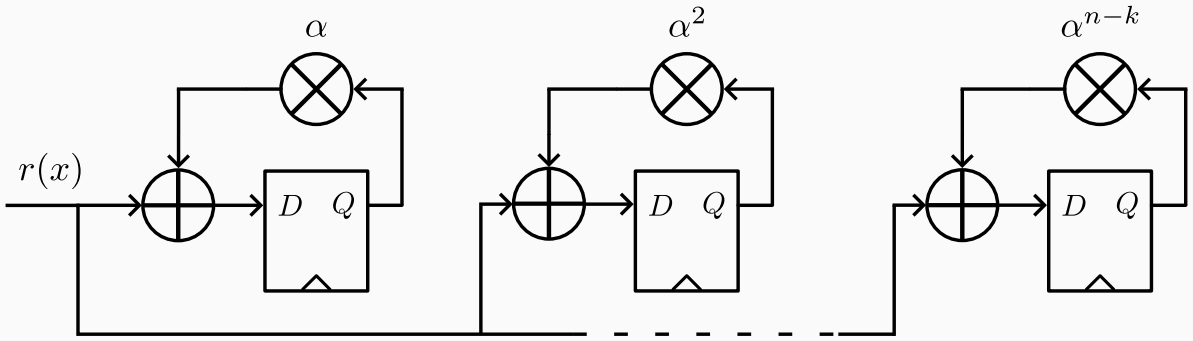
Decoding algorith overview

The decoder computes:

- the polynomial associated to the syndrome of the received word $r(x)$
- both positions and values of the coefficients of $e(x)$
- the error-free codeword is derived as $c(x) = r(x) - e(x)$.

Public code: Reed Solomon
Decoder

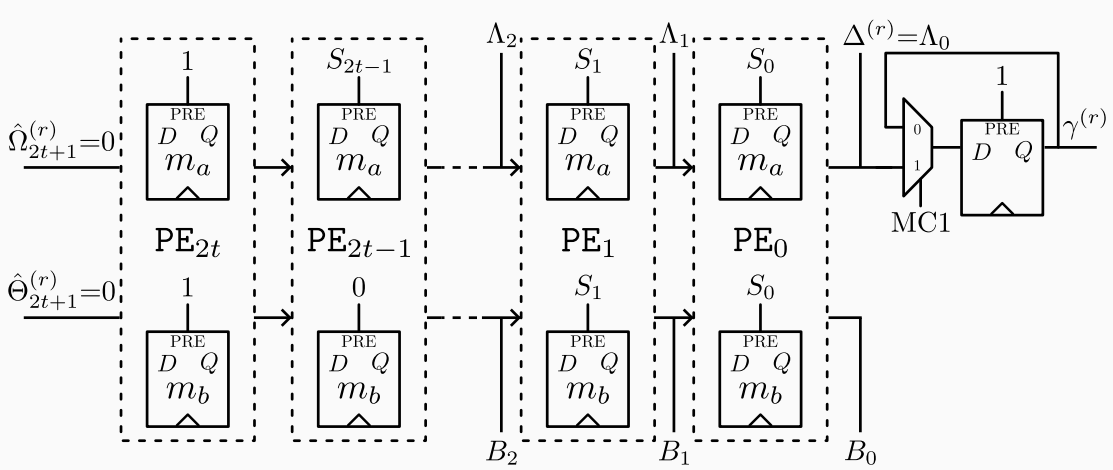
First, the received polynomial $r(x)$ is evaluated at each root α^i of the generator polynomial $g(x)$ using the Horner's method, determining the *syndrome polynomial* $S(x)$

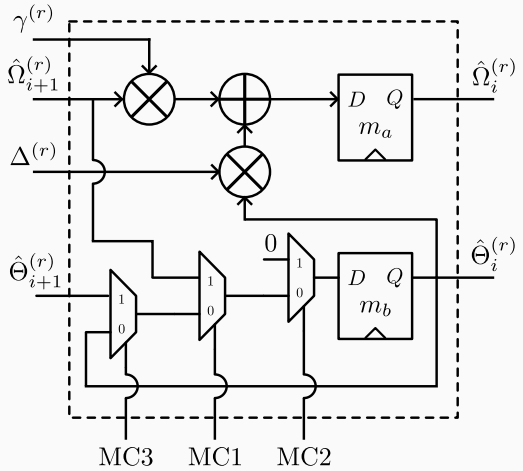


Public code: Reed Solomon
Decoder

We employed the design of the Enhanced Parallel Inversionless Berlekamp-Massey Algorithm (ePIBMA) introduced in [18].

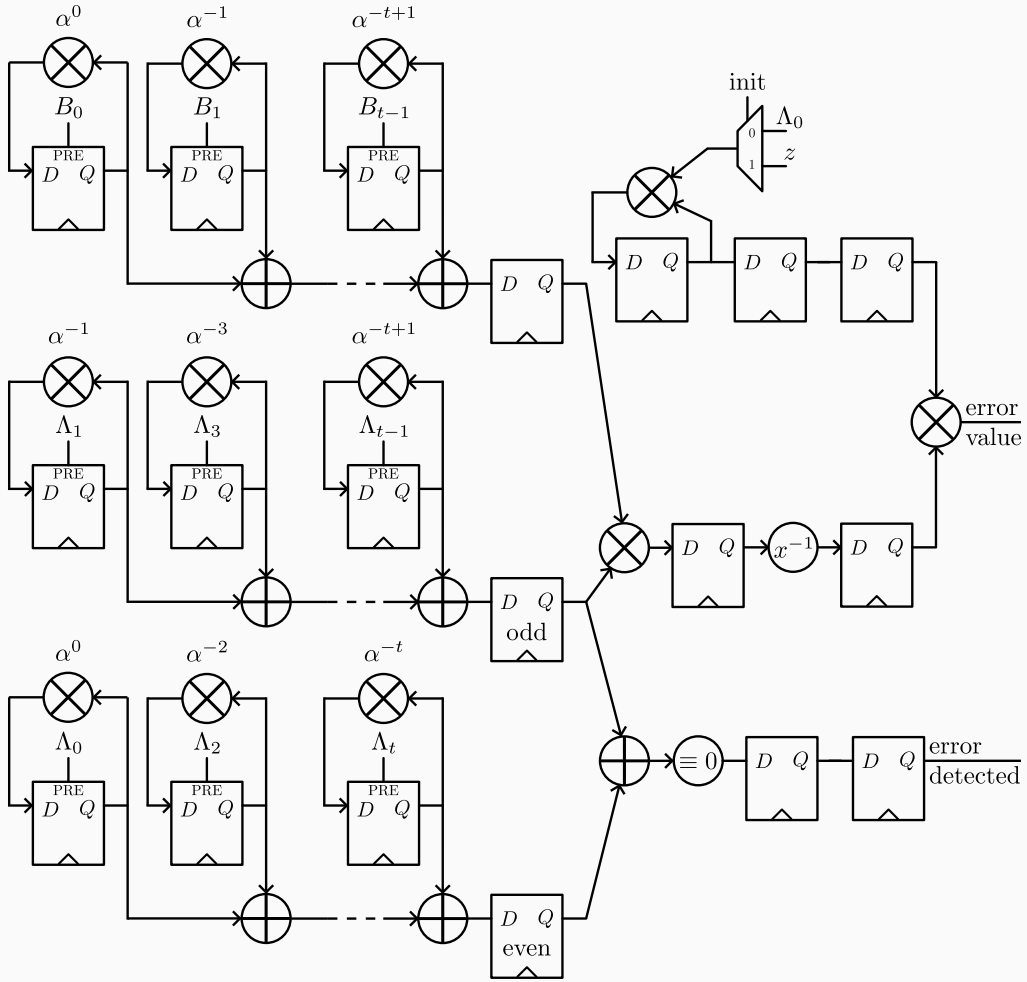
The *error locator polynomial* $\Lambda(x)$ and the *auxiliary polynomial* $B(x)$ are derived from the syndrome polynomial $S(x)$





Public code: Reed Solomon
Decoder

Similarly, we used the Enhanced Chien Search and Error Evaluator design from [18] to compute the *error evaluator polynomial* $\Omega(x)$ from $\Lambda(x)$ and $B(x)$.



Public code: Reed Muller
Encoder

To derive the 128-bit codewords corresponding to each 8-bit input message, we follow the traditional message vector multiplied by the generator matrix G_{RM}.

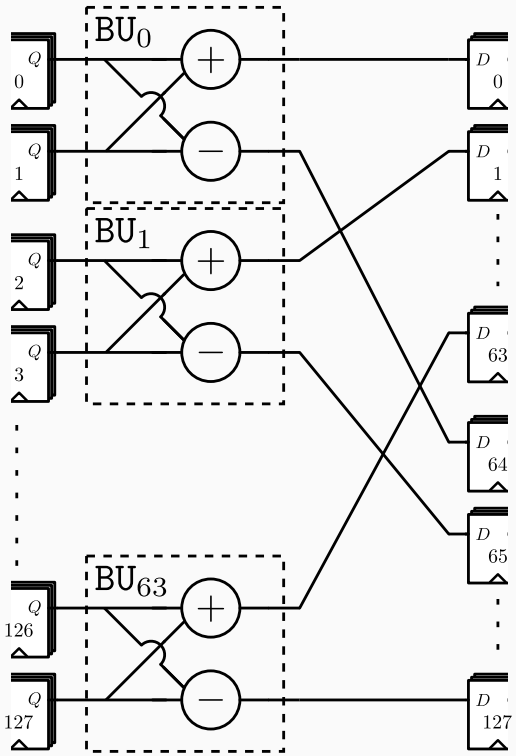
$$G_{RM} = \begin{bmatrix} \texttt{0xAAAAAAAA} & \texttt{0xAAAAAAAA} & \texttt{0xAAAAAAAA} & \texttt{0xAAAAAAAA} \\ \texttt{0xCCCCCCCC} & \texttt{0xCCCCCCCC} & \texttt{0xCCCCCCCC} & \texttt{0xCCCCCCCC} \\ \texttt{0xF0F0F0F0} & \texttt{0xF0F0F0F0} & \texttt{0xF0F0F0F0} & \texttt{0xF0F0F0F0} \\ \texttt{0xFF00FF00} & \texttt{0xFF00FF00} & \texttt{0xFF00FF00} & \texttt{0xFF00FF00} \\ \texttt{0xFFFF0000} & \texttt{0xFFFF0000} & \texttt{0xFFFF0000} & \texttt{0xFFFF0000} \\ \texttt{0x00000000} & \texttt{0xFFFFFFFF} & \texttt{0x00000000} & \texttt{0xFFFFFFFF} \\ \texttt{0x00000000} & \texttt{0x00000000} & \texttt{0xFFFFFFFF} & \texttt{0xFFFFFFFF} \\ \texttt{0xFFFFFFFF} & \texttt{0xFFFFFFFF} & \texttt{0xFFFFFFFF} & \texttt{0xFFFFFFFF} \end{bmatrix}$$

Working with 32-bits words, the presence of repeated words in G_{RM} yields some identical intermediate values during the multiplication.

Consequently, the size of multiplexers and the number of XOR gates were decreased substantially.

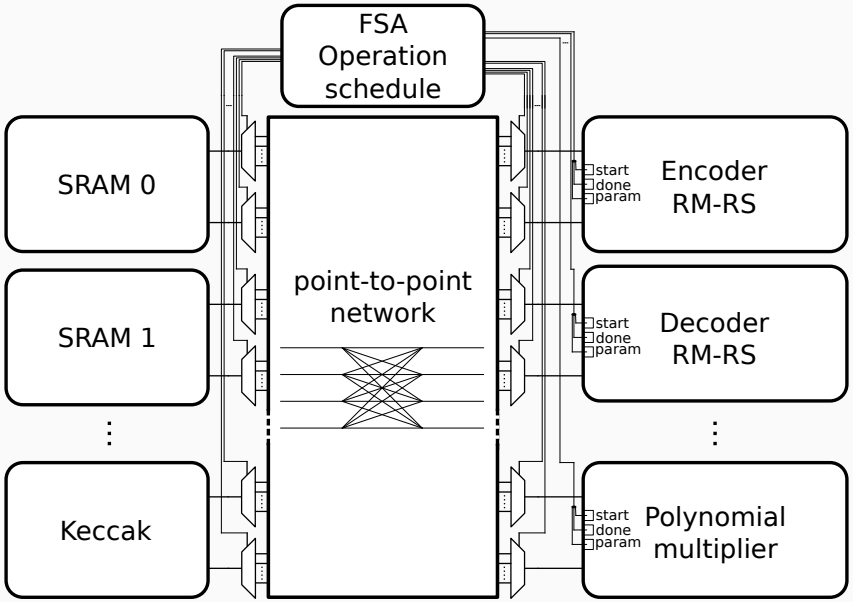
Public code: Reed Muller
Decoder

The operation is carried out by a Maximum Likelihood (ML) decoder computing a fast Hadamard transform [19]



We find the maximum absolute value with a pipelined comparator tree computing pairwise maxima, acting on a tunable-sized input vector.

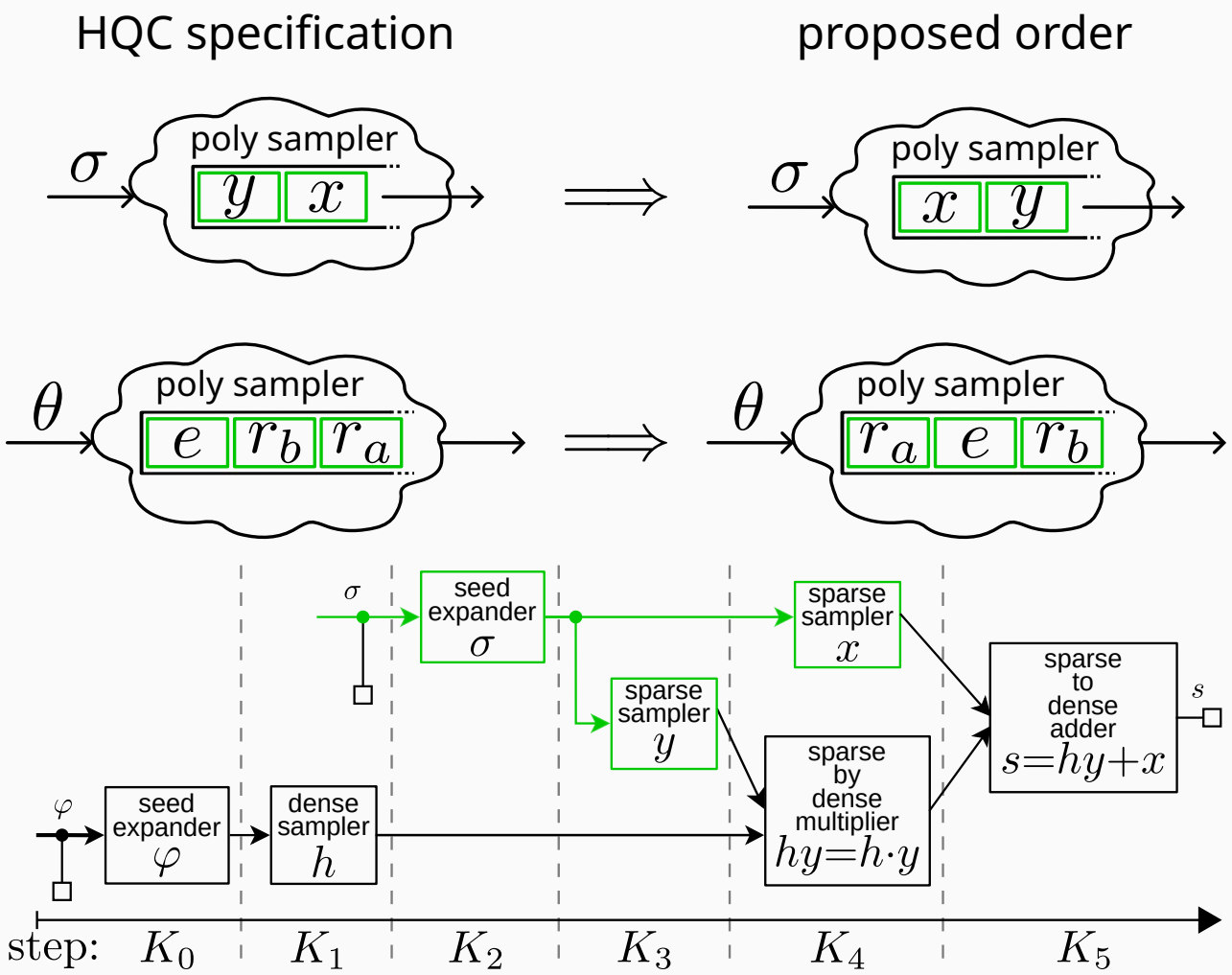
Operation schedule
Top-Level Design configuration



Top-Level Design (TLD) configuration implementing a KEM primitive.

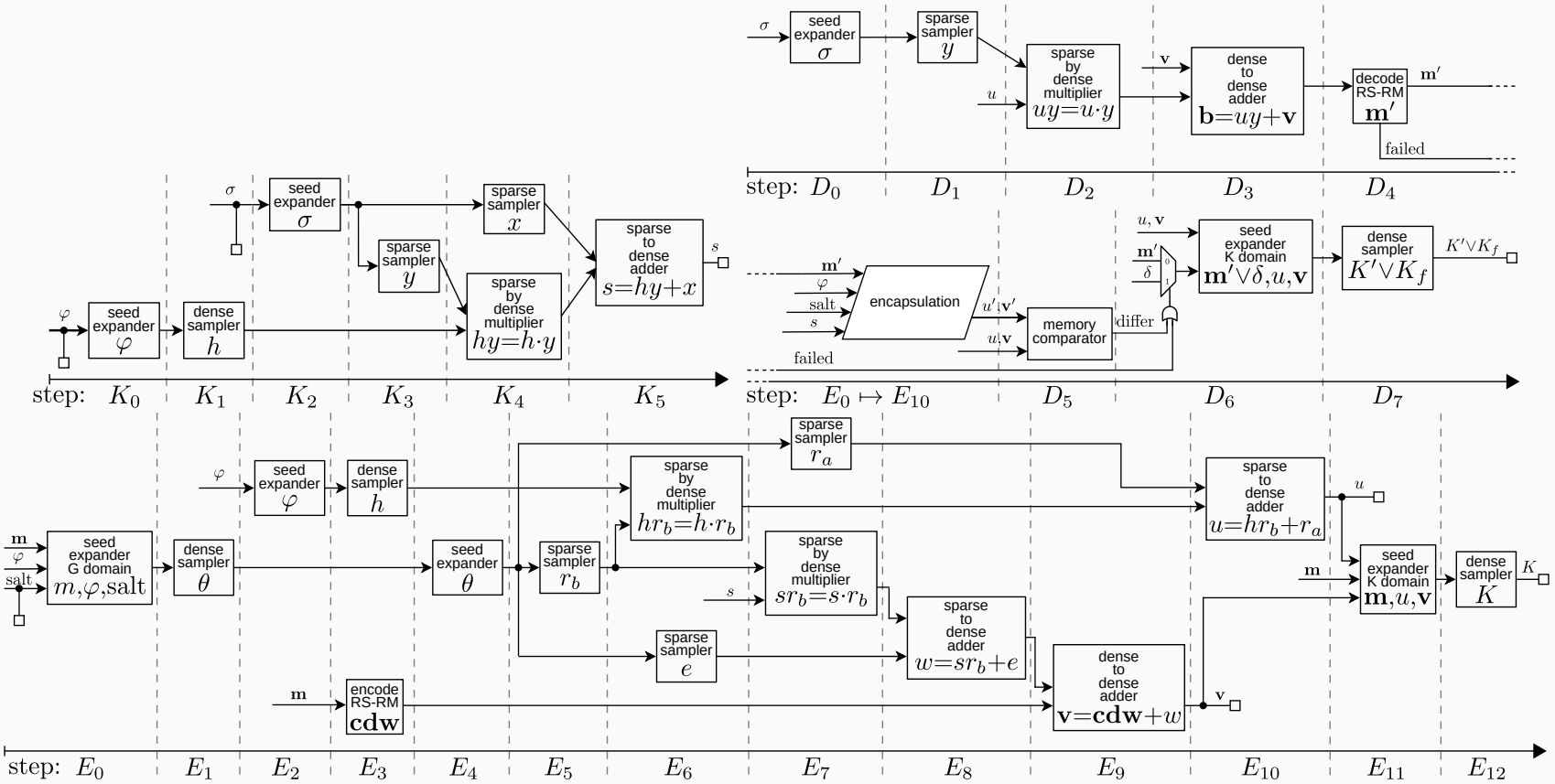
We employed five True Dual-Port memories, which are connected in every moment to a specific compute unit (e.g., polynomial multiplier) by global Finite State Automata (FSA) following a fixed schedule of operations.

Operation schedule
Sampling order optimization



Performance gains from 13% to 32% over the entire cryptographic primitive **without any cost or security implications**

Operation schedule
HQC.KEM schedules



Experimental results

Designed in SystemVerilog, tested with CocoTB following the Universal Verification Methodology (UVM).

Synthesized on an AMD Artix-7 `xc7a200tfbg484-3` FPGA, and validated it on a Digilent's Arty A7-100T employing the (modified) official Known Answer Tests (KAT) via a UART module.

The source code is available on Zenodo.

Experimental results
Top-level: Key Generation

HQC keygen top-modules w/o SHAKE256 (5520 LUTs, 2810 FFs).

Parameter set	Design	Area eSlice	Frequency MHz	Latency μs	Area-Time product
hqc128	[20]	1879	179	88	165
	[21] (HLS, perf.)	2849	150	270	768
	This work	4267	208	**30**	**127**
hqc192	[20]	1866	189	222	415
	This work	4348	207	**72**	**314**
hqc256	[20]	1866	188	437	817
	This work	4272	201	**138**	**591**

Experimental results
Top-level: Encapsulation

HQC encapsulation top-modules w/o SHAKE256 (5520 LUTs, 2810 FFs).

Parameter set	Design	Area eSlice	Frequency MHz	Latency μs	Area-Time product
hqc128	[20] (balanced)	2701	179	186	504
	[20] (high speed)	3377	179	125	423
	[21] (HLS, perf.)	4575	152	586	2682
	This work	4326	168	**79**	**343**
hqc192	[20] (balanced)	2990	182	496	1484
	[20] (high speed)	3785	196	292	1106
	This work	4468	175	**180**	**803**
hqc256	[20] (balanced)	3123	182	973	3039
	[20] (high speed)	3901	196	553	2160
	This work	4412	187	**313**	**1382**

Experimental results
Top-level: Decapsulation

HQC decapsulation top-modules w/o SHAKE256 (5520 LUTs, 2810 FFs).

Parameter set	Design	Area eSlice	Frequency MHz	Latency μs	Area-Time product
hqc128	[20] (balanced)	4806	192	251	1206
	[20] (high speed)	5556	179	207	1154
	[21] (HLS, perf.)	6130	152	1270	7787
	This work	5956	167	**119**	**709**
hqc192	[20] (balanced)	5309	186	676	3590
	[20] (high speed)	6051	186	498	3018
	This work	7068	161	**287**	**2026**
hqc256	[20] (balanced)	5549	186	1335	7408
	[20] (high speed)	6289	186	966	6076
	This work	8098	151	**570**	**4614**

Experimental results
Top-level: unified design

Comparison of the unified designs compatible with all KEM primitives for parameter set guaranteeing a security similar to AES-128[1].

Design			Area	Freq.	KeyGeneration		Encapsulation		Decapsulation	
scheme	ref.	variant	eSlice	MHz	μs	AT	μs	AT	μs	AT
Kyber	[22]	–	4147	220	10	40	15	62	20	85
Kyber	[23]	–	2758	161	24	65	32	87	42	115
HQC	This work	–	12471	143	39	488	82	1027	128	1597
HQC	[20]	balanced	10406	164	96	1000	204	2122	294	3059
HQC	[20]	high-speed	11167	178	89	989	126	1407	209	2333
HQC	[24]	–	20564	178	112	2311	225	4621	393	8087
BIKE	[25]	lightweight	5930	121	3826	22691	446	2646	6950	41216
BIKE	[25]	trade-off	9717	100	1870	18171	280	2721	4210	40909
BIKE	[25]	high-speed	15344	113	1681	25800	133	2037	1168	17924
C. McEliece	[26]	lightweight	36007	112	1161	41794	1518	54653	79286	2854841
C. McEliece	[26]	high-speed	48173	113	265	12789	885	42631	8584	413520

[1] Highlighted designs support also security margins of AES-192 and AES-256.

Experimental results
ASIC synthesis

ASIC synthesis of top-level HQC.KEM designs using *Yosys* and *OpenROAD* with *FreePDK45* tech library and typical process corner 1.1V @ 25°C.

Design	Frequency (MHz)	Area $\mathbf{mm}^2$
Key generation	502	0.235
Encapsulation	390	0.280
Decapsulation	457	0.469
Unified	419	0.496

Experimental results
Comparison with a SW execution

Comparing the execution runtime of the three KEM primitives with a software execution of the reference C code on a Rockchip RK3288 ARM Cortex-A17 CPU at 1.8GHz, our hardware accelerator is from 15.6× to 19.8× faster.

This target was chosen because it is produced with a 28nm process node technology similar to the one used for the AMD Artix-7 XC7A35T, and it has a similar bulk price of 20 US $.

Producing an ASIC version of the design on a modern process node, the performance advantage will inevitably grow due to the higher working frequencies.

CROSS: Codes and Restricted Objects Signature Scheme

Introduction to CROSS

The Codes and Restricted Objects Signature Scheme (CROSS) scheme is a digital signature algorithm proposed in the additional NIST call for digital signatures relying on the NP-complete Restricted Syndrome Decoding Problem (R-SDP).

It is built applying the Fiat-Shamir construction to transform the CROSS-ID Zero-Knowledge (ZK) identification protocol into an non-interactive one using a one-way function and achieving the EUF-CMA security.

- $\mathsf{H} \in \mathbb{F}_p^{(n-k)\times n}$ of a p-ary $[n, k, d]$ public random linear code
- $\mathsf{e} \in E^n \subset \mathbb{F}_p^n$ the secret error vector from a subgroup of $\mathbb{F}_p^n$
- $\mathsf{s} = \mathsf{e}\mathsf{H}^\top \in \mathbb{F}_p^{n-k}$ the syndrome associated to the error e

The subgroup E is generated by the public element g of order z:

$$\langle g \rangle = \left\{ g^i \mid i \in \{1, \ldots, z\} \right\} = E \subset \mathbb{F}_p^*$$

Given $\mathsf{pk} = (\mathsf{H}, \mathsf{s})$, find a vector $\mathsf{e} \in E^n$ such that $\mathsf{s} = \mathsf{e}\mathsf{H}^\top$.

The removal of the fixed-weight constraint in favor of the restriction of the ambient space of the error vector allowed to speed-up the computation and reduce the signature size.

Algebraic structure

- $E = \langle g \rangle = \left\{ g^i \mid i \in \{1, \ldots, z\} \right\} \subset \mathbb{F}_p^*$

Vector arithmetic modulo a prime $p \in \{127, 509\}$: $+$, $-$, $\odot$, $g^{(\cdot)}$

Algebraic structure

- $E = \langle g \rangle = \left\{ g^i \mid i \in \{1, \ldots, z\} \right\} \subset \mathbb{F}_p^*$

Vector arithmetic modulo a prime $p \in \{127, 509\}$: $+$, $-$, $\odot$, $g^{(\cdot)}$

$\mathsf{a} \in E^n$ compactly represented by an element $\bar{\mathsf{a}} \in \mathbb{F}_z^n$, $z \in \{7, 127\}$

$(E^n, \odot)$ isomorphic to $(\mathbb{F}_z^n, +)$: $\mathsf{a} \odot \mathsf{b} = g^{\bar{\mathsf{a}}+\bar{\mathsf{b}}}$, $\quad \mathsf{a}^{-1} = g^{-\bar{\mathsf{a}}}$

Algebraic structure

- $E = \langle g \rangle = \left\{ g^i \mid i \in \{1, \ldots, z\} \right\} \subset \mathbb{F}_p^*$
- $G = \langle \{\mathsf{b}_1, \ldots, \mathsf{b}_m\} \rangle = \{ \bigodot_{i=1}^{m} \mathsf{b}_i{}^{u_i} \mid \mathsf{b}_i \in E^n \wedge u_i \in \mathbb{F}_z^\star \wedge m < n \} \subset E^n$

Vector arithmetic modulo a prime $p \in \{127, 509\}$: $+$, $-$, $\odot$, $g^{(\cdot)}$

$\mathsf{a} \in E^n$ compactly represented by an element $\bar{\mathsf{a}} \in \mathbb{F}_z^n$, $z \in \{7, 127\}$

$(E^n, \odot)$ isomorphic to $(\mathbb{F}_z^n, +)$: $\mathsf{a} \odot \mathsf{b} = g^{\bar{\mathsf{a}} + \bar{\mathsf{b}}}$, $\quad \mathsf{a}^{-1} = g^{-\bar{\mathsf{a}}}$

$\overline{\mathsf{M}} = \left[\bar{\mathsf{b}}_1, \ldots, \bar{\mathsf{b}}_m \right]^\top$ with exponents of the bases, $\quad \mathsf{a} = g^{\bar{\mathsf{a}}_G \overline{\mathsf{M}}}, \bar{\mathsf{a}}_G \in \mathbb{F}_z^m$

Algebraic structure

- $E = \langle g \rangle = \left\{ g^i \mid i \in \{1, \ldots, z\} \right\} \subset \mathbb{F}_p^*$
- $G = \langle \{\mathsf{b}_1, \ldots, \mathsf{b}_m\} \rangle = \{ \bigodot_{i=1}^{m} \mathsf{b}_i^{u_i} \mid \mathsf{b}_i \in E^n \wedge u_i \in \mathbb{F}_z^\star \wedge m < n \} \subset E^n$

Vector arithmetic modulo a prime $p \in \{127, 509\}$: $+$, $-$, $\odot$, $g^{(\cdot)}$

$\mathsf{a} \in E^n$ compactly represented by an element $\bar{\mathsf{a}} \in \mathbb{F}_z^n$, $z \in \{7, 127\}$

$(E^n, \odot)$ isomorphic to $(\mathbb{F}_z^n, +)$: $\mathsf{a} \odot \mathsf{b} = g^{\bar{\mathsf{a}} + \bar{\mathsf{b}}}$, $\quad \mathsf{a}^{-1} = g^{-\bar{\mathsf{a}}}$

$\overline{\mathsf{M}} = \left[\bar{\mathsf{b}}_1, \ldots, \bar{\mathsf{b}}_m \right]^\top$ with exponents of the bases, $\quad \mathsf{a} = g^{\bar{\mathsf{a}}_G \overline{\mathsf{M}}}, \bar{\mathsf{a}}_G \in \mathbb{F}_z^m$

The linear transitive maps $\mathsf{v} : E^n \mapsto E^n$ and $\mathsf{v}_G : G \mapsto G$ are simply $\mathsf{v}(\mathsf{a}) = \mathsf{v} \odot \mathsf{a} = g^{\bar{\mathsf{v}} + \bar{\mathsf{a}}}$

The underlying hard problem is tweaked by selecting:

- the error vector ambient space E^n or G, for R-SDP and R-SDP(G) respectively
- a different parametrization of the random $[n, k, d]$ random linear code
- the number of CROSS-ID ZK protocol repetitions t

Background
CROSS-ID ZK protocol

CROSS-ID ZK protocol

Require: $\mathsf{sk} = e \in \mathsf{G} \subseteq \mathsf{E}^n$

$\mathsf{pk} = \left(\mathsf{H} \in \mathbb{F}_p^{(n-k)\times n}, \mathsf{s} = \mathsf{eH}^\top \in \mathbb{F}_p^{n-k}\right)$

PROVER $\mathcal{P}$		VERIFIER $\mathcal{V}$
$\mathtt{seed} \xleftarrow{\$} \{0,1\}^\lambda$		
	Commitment	
$\mathsf{e}', \mathsf{u}' \leftarrow \mathsf{CSPRNG}(\mathtt{seed}, \mathsf{G}\times\mathbb{F}_p^n)$ $\mathsf{v} \leftarrow \mathsf{e} \odot (\mathsf{e}')^{-1}$ $\mathsf{u} \leftarrow \mathsf{v} \odot \mathsf{u}'$ $\mathsf{s}' \leftarrow \mathsf{uH}^\top$ $\mathtt{cmt}_0 \leftarrow \mathsf{Hash}(\mathsf{s}' \| \mathsf{v})$ $\mathtt{cmt}_1 \leftarrow \mathsf{Hash}(\mathsf{u}' \| \mathsf{e}')$	$\xrightarrow[\text{commitment}]{\mathtt{cmt}_0, \mathtt{cmt}_1}$	
	First challenge	
	$\xleftarrow[1^{\text{st}}\text{ challenge}]{\mathtt{chall}_1}$	$\mathtt{chall}_1 \xleftarrow{\$} \mathbb{F}_p^*$
$\mathsf{y} \leftarrow \mathsf{u}' + \mathtt{chall}_1 \mathsf{e}'$ $\mathtt{dig}_\mathsf{y} \leftarrow \mathsf{Hash}(\mathsf{y})$	$\xrightarrow[1^{\text{st}}\text{ response}]{\mathtt{dig}_\mathsf{y}}$	
	Second challenge	
	$\xleftarrow[2^{\text{nd}}\text{ challenge}]{\mathtt{chall}_2}$	$\mathtt{chall}_2 \xleftarrow{\$} \{0,1\}$
if $\mathtt{chall}_2 = 0$ then $\mathtt{resp} \leftarrow \mathsf{y} \| \mathsf{v}$ else $\mathtt{resp} \leftarrow \mathtt{seed}$	$\xrightarrow[2^{\text{nd}}\text{ response}]{\mathtt{resp}}$	
	Verification	

		if $\mathtt{chall}_2 = 0$ then
		$\mathsf{y}' \leftarrow \mathsf{v} \odot \mathsf{y}$
		$\mathsf{s}' \leftarrow \mathsf{y}'\mathsf{H}^\top - \mathtt{chall}_1 \mathsf{s}$
		if $\mathsf{Hash}(\mathsf{y}) \neq \mathtt{dig}_\mathsf{y} \vee \mathsf{Hash}(\mathsf{s}' \| \mathsf{v}) \neq \mathtt{cmt}_0 \vee \mathsf{v} \notin \mathsf{G}$ then fail
		else
		$(\mathsf{e}', \mathsf{u}') \leftarrow \mathsf{CSPRNG}(\mathtt{seed}, \mathsf{G}\times\mathbb{F}_p^n)$
		$\mathsf{y} \leftarrow \mathsf{u}' + \mathtt{chall}_1 \mathsf{e}'$
		if $\mathsf{Hash}(\mathsf{y}) \neq \mathtt{dig}_\mathsf{y} \vee \mathsf{Hash}(\mathsf{u}' \| \mathsf{e}') \neq \mathtt{cmt}_1$ then fail

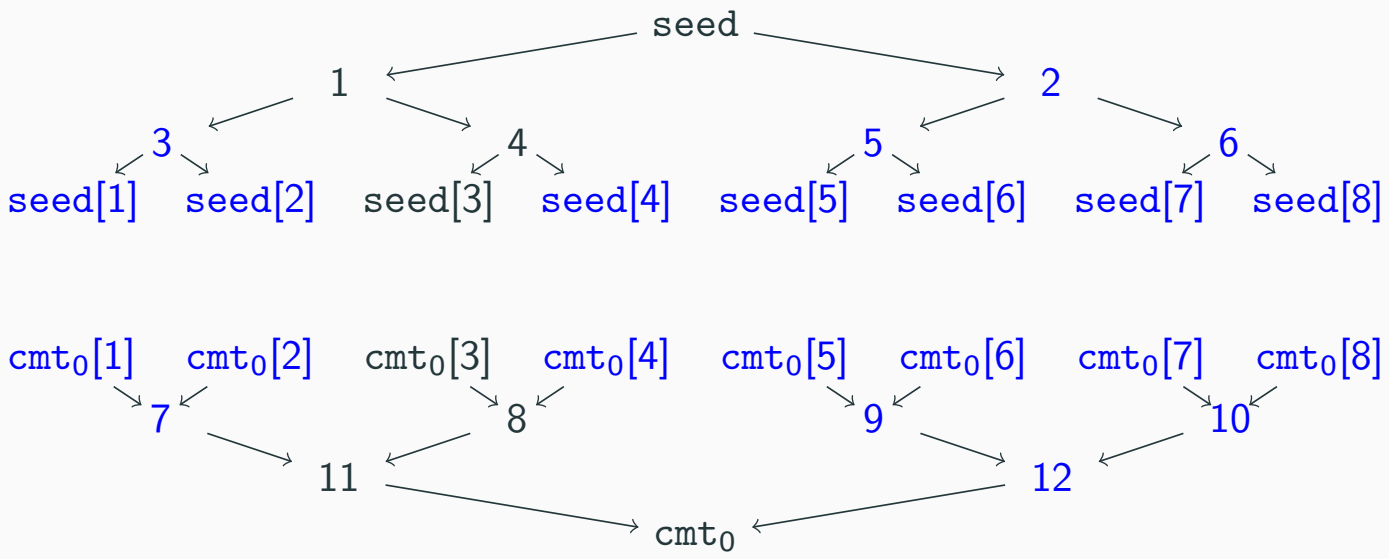
The t rounds are performed in parallel, and commitments are transmitted only if the Verifier cannot recompute them.

When the second challenge is unbalanced (w out of t binary challenges are 1), the Prover will send more frequently the round seed, which is far smaller than y||v.

Warning

This requires to increase the number of protocol repetitions t to maintain the same security, hence trades a higher execution runtime for a smaller signature

Round seeds can be generated via the expansion of a master seed in a binary tree shape.
Analogously, cmt0 of each round can be pre-combined during the signature generation using a Merkle tree.



CROSS seed and commitment trees

The element-wise modular operations are carried out in parallel: for $p, z \in \{7, 127, 509\}$, a 64-bits word encodes 21, 9, and 7 elements.

The mod7 and mod127 reduction uses the efficient Mersenne prime reduction, and employs only a few adders.
By contrast, for mod509 the Barrett reduction employs two integer multiplications (the first one having a large output domain).

The operation is repeated until all the n or $m < n$ vector elements are processed.

Leveraging the small values of p, z and the fixed public generator g, the scalar operation $g^a \bmod p$ is performed via a look-up table.

z table entries of $\log_2 p$ bits –> only 7 or 18 LUT FPGA units!

9, and 7 parallel operations can be performed each clock cycle, but an input FIFO is necessary due to the different read/write rates.

$$\mathsf{H}^\top = \left[\mathsf{V}^\top \mid \mathsf{I}_{n-k}\right] = \begin{bmatrix} V_{0,0}^\top & V_{0,1}^\top & \cdots & V_{0,n-k-1}^\top \\ V_{1,0}^\top & V_{1,1}^\top & \cdots & V_{1,n-k-1}^\top \\ \vdots & \vdots & \ddots & \vdots \\ V_{k-1,0}^\top & V_{k-1,1}^\top & \cdots & V_{k-1,n-k-1}^\top \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$\overline{\mathsf{M}} = \left[\overline{\mathsf{W}}, \bar{\mathsf{I}}_m\right] = \begin{bmatrix} \overline{W}_{0,0} & \overline{W}_{0,1} & \cdots & \overline{W}_{0,n-m-1} & 1 & 0 & \cdots & 0 \\ \overline{W}_{1,0} & \overline{W}_{1,1} & \cdots & \overline{W}_{1,n-m-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \overline{W}_{m-1,0} & \overline{W}_{m-1,1} & \cdots & \overline{W}_{m-1,n-m-1} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Both matrices are serialized row-wise.

Leveraging their systematic form to avoid part of the computation.

$$\bar{\mathrm{e}} = \bar{\mathrm{e}}_G \overline{\mathrm{M}} = \bar{\mathrm{e}}_G \left[\overline{\mathrm{W}}, \bar{\mathrm{I}}_m\right] = \left[\bar{\mathrm{e}}_G \overline{\mathrm{W}}, \bar{\mathrm{e}}_G\right]$$

Takes only $O(m(n-m)) < O(mn)$ multiplications in $\mathbb{F}_z$

$$\mathrm{s} = \mathrm{eH}^\top = \mathrm{e}\left[\mathrm{V} \mid \mathrm{I}_{n-k}\right] = [e_k, e_{k+1}, \ldots, e_n] + [e_0, e_1, \ldots, e_{k-1}]\,\mathrm{V}^\top$$

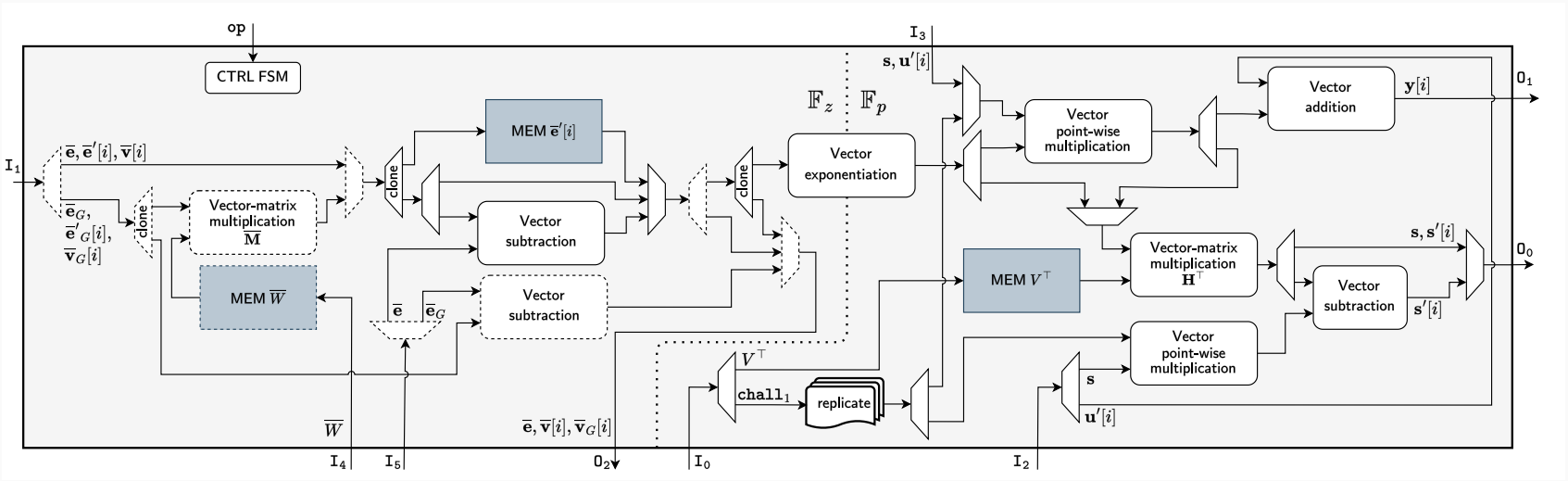
Takes only $O(k(n-k)) < O(n(n-k))$ multiplications in $\mathbb{F}_p$

Vector operator and result are transferred using 64-bits words. Matrices $\mathrm{V}^\top$ and $\overline{\mathrm{W}}$ are accessed using 64-bits, 192-bits, or enough bits in order to perform an entire row-by-coefficient computation.

Different levels of computational parallelism with this design choice.

Hardware design
Arithmetic unit

Following the order of operations in the three DSA operations (KeyGeneration, Sign, Verify), we chained the arithmetic units to limit the memory transfers and maximize the performance.



We used two local memories to store the serialized matrices employed during the t CROSS-ID rounds, and cache the (expanded) round errors $\overline{\mathbf{e}}'$.

Experimental results

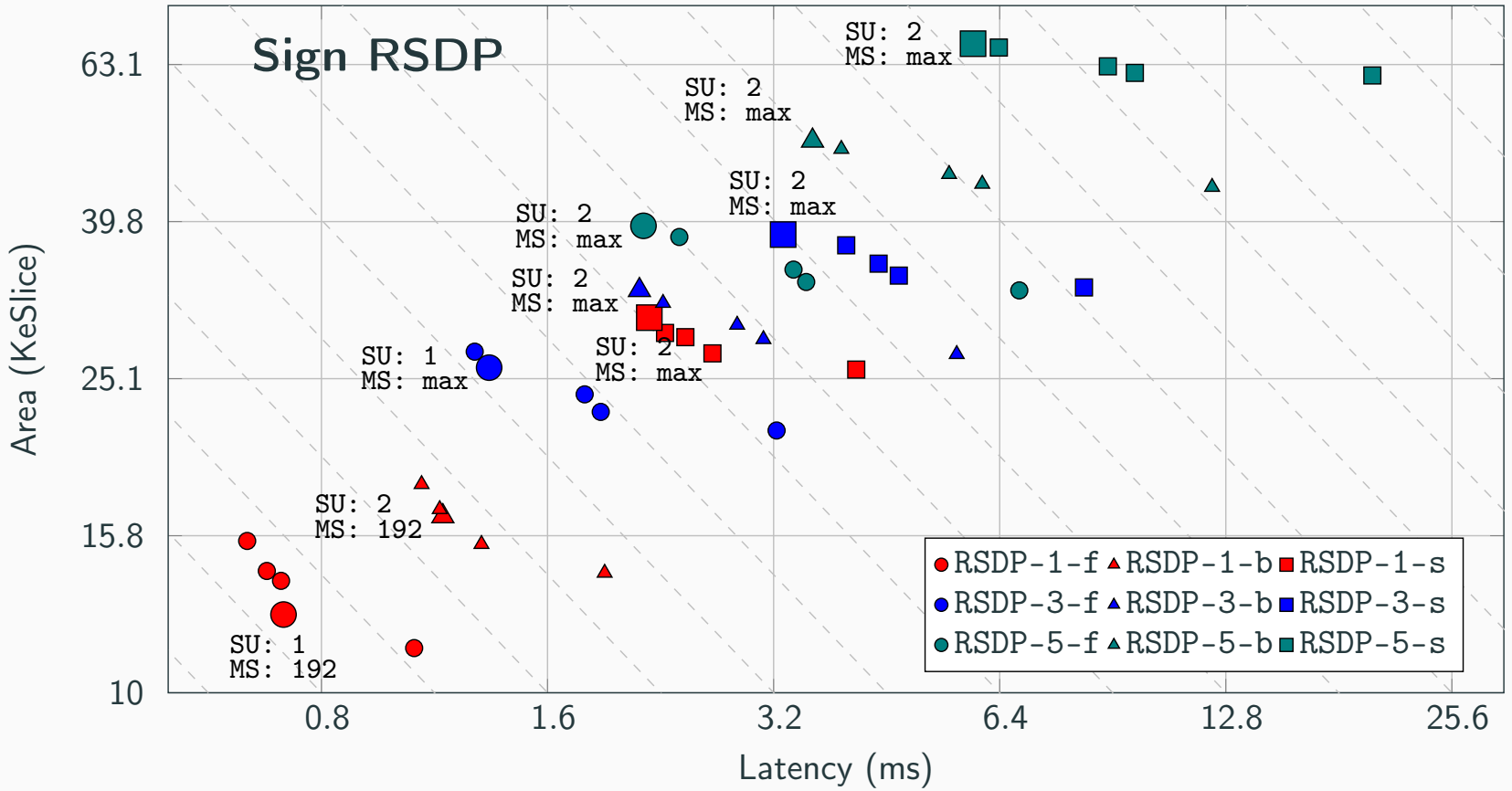
Designed in SystemVerilog, tested with CocoTB employing the reference software via ctypes function bindings.

Synthesized on various FPGAs of the AMD Artix-7 family, and validated it on a Digilent's Nexys Video employing the official Known Answer Tests (KAT) via a UART module.

We conducted a Design Space Exploration to determine Area/Latency trade points by varying:

- the parallelism of the matrix-vector multiplication
- the number of SHAKE-256 rounds per clock cycle

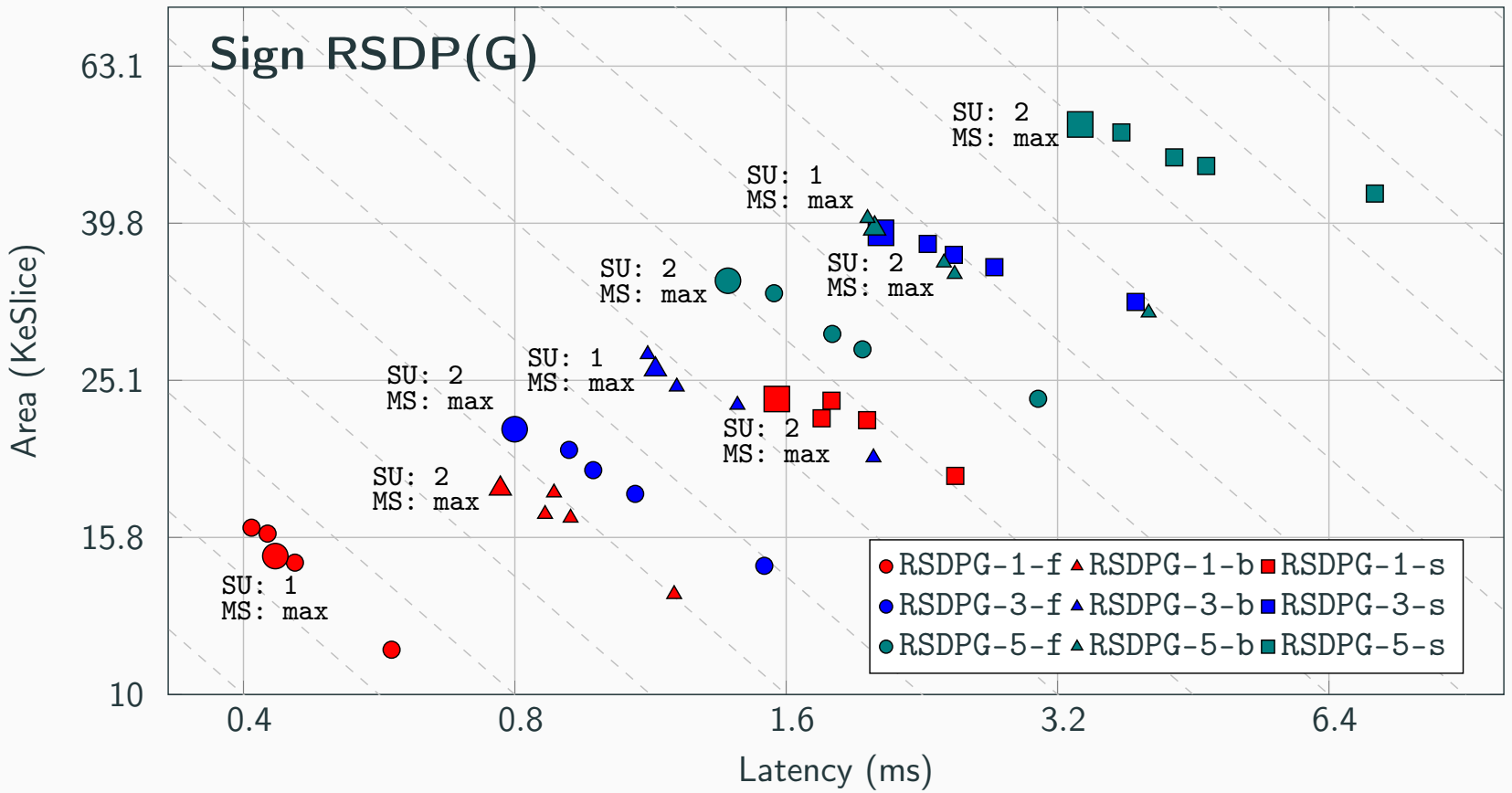
Experimental results
Artix-7 FPGA synthesis results[1]



Design Space Exploration

[1] Credits to Patrick Karl from the Technical University of Munich.

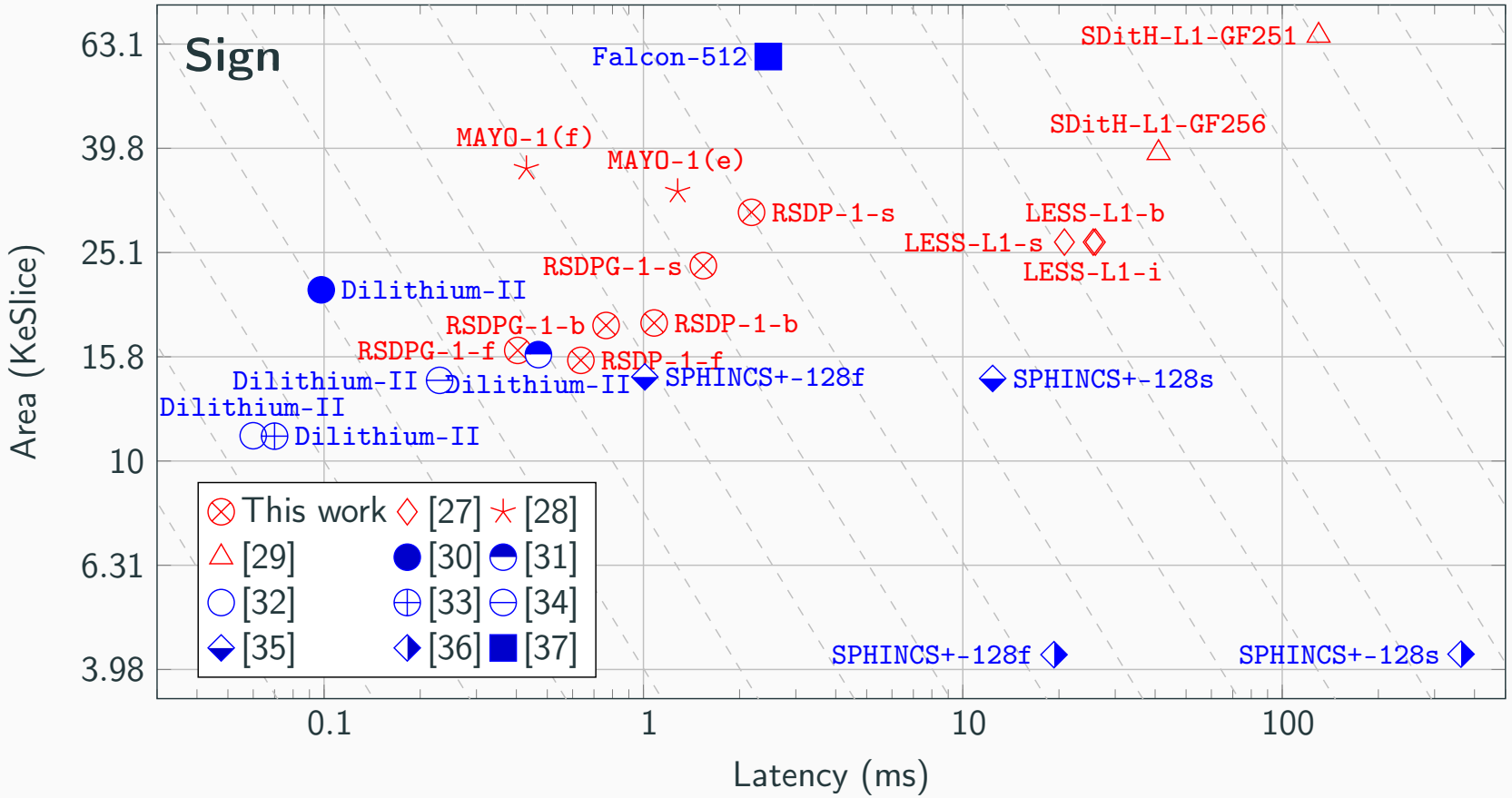
Experimental results
Artix-7 FPGA synthesis results[1]



Design Space Exploration

[1] Credits to Patrick Karl from the Technical University of Munich.

Experimental results
Artix-7 FPGA synthesis results[1]



Comparison with other DSA – Security level AES-128

[1] Credits to Patrick Karl from the Technical University of Munich.

Experimental results
Comparison with a SW execution[1]

Comparing the execution runtime of the three DSA primitives with a software execution of the reference C code on a Broadcom BCM2837B0 ARM Cortex-A53 CPU at 1.4GHz, our hardware accelerator produces a signature of a 59 bytes message from 2.3× to 22.7× faster.

This target was chosen due to being the only datapoint available right now in the SUPERCOP open cryptographic benchmarking tool for a low-end platform produced in a similar technology node and having a similar bulk price.

Producing an ASIC version of the design on a modern process node, the performance advantage will inevitably grow due to the higher working frequencies.

[1] Credits to Marco Gianvecchio from Politecnico di Milano.

Concluding remarks

Concluding remarks

We presented a systematic approach for the analysis, design, and optimization of hardware accelerators for three PQC algorithms: the lattice-based KEM.NTRU, the code-based KEM.HQC, and the code-based DSA.CROSS.

Concluding remarks

We presented a systematic approach for the analysis, design, and optimization of hardware accelerators for three PQC algorithms: the lattice-based KEM.NTRU, the code-based KEM.HQC, and the code-based DSA.CROSS.

Leveraging the data parallelisms offered by the sub-algorithms and the parallel schedule of operations in the KEM/DSA primitives, we set up design-space experiments to determine the solutions with interesting latency and efficiency figures.

Concluding remarks

We presented a systematic approach for the analysis, design, and optimization of hardware accelerators for three PQC algorithms: the lattice-based KEM.NTRU, the code-based KEM.HQC, and the code-based DSA.CROSS.

Leveraging the data parallelisms offered by the sub-algorithms and the parallel schedule of operations in the KEM/DSA primitives, we set up design-space experiments to determine the solutions with interesting latency and efficiency figures.

The produced hardware accelerators distinguished for their low primitive execution latency, higher efficiency, and high flexibility.

POLITECNICO
MILANO 1863

Thanks for your attention!

Contatti

Via Giuseppe Ponzio, 34, Milano, 20133, Italy
+39 02 2399 9047
francesco.antognazza@polimi.it
https://antognazza.faculty.polimi.it/

References

[1] S. Jaques and A. G. Rattew, "Qram: A survey and critique," *arXiv preprint arXiv:2305.10310*, 2023. doi: 10.48550/arXiv.2305.10310. [Online]. Available: https://doi.org/10.48550/arXiv.2305.10310.

[2] Google Quantum AI and Collaborators, "Quantum error correction below the surface code threshold," en, *Nature*, pp. 1–3, Dec. 2024, Publisher: Nature Publishing Group, issn: 1476-4687. doi: 10.1038/s41586-024-08449-y. [Online]. Available: https://www.nature.com/articles/s41586-024-08449-y (visited on 01/10/2025).

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, G. L. Miller, Ed., ACM, 1996, pp. 212–219. doi: 10.1145/237814.237866. [Online]. Available: https://doi.org/10.1145/237814.237866.

[4] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, IEEE Computer Society, 1994, pp. 124–134. doi: 10.1109/SFCS.1994.365700. [Online]. Available: https://doi.org/10.1109/SFCS.1994.365700.

[5] ETSI, *TS 103 744 quantum-safe hybrid key exchanges*, [Online]. Available from: https://web.archive.org/web/20240702014511/https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf, (Archived on 7 Aug. 2024), 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf.

[6] ETSI, *GR QSC 006 Quantum-Safe Cryptography (QSC); Limits to Quantum Computing applied to symmetric key sizes*, [Online]. Available from: hhttps://web.archive.org/web/20241009041101/https://www.etsi.org/deliver/etsi_gr/QSC/001_099/006/01.01.01_60/gr_QSC006v010101p.pdf, (Archived on 13 Feb. 2025), 2017. [Online]. Available: https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=49740.

[7] ANSSI, *ANSSI views on the post-quantum cryptography transition (2023 follow up)*, [Online]. Available from: https://web.archive.org/web/20240127124757/https://cyber.gouv.fr/sites/default/files/document/follow_up_position_paper_on_post_quantum_cryptography.pdf, (Archived on 7 Aug. 2024), 2023. [Online]. Available: https://cyber.gouv.fr/sites/default/files/document/follow_up_position_paper_on_post_quantum_cryptography.pdf.

[8] BSI, *Study: Status of quantum computer development V2.1*, [Online]. Available from: https://web.archive.org/web/20250103231418/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/Entwicklungstand_QC_V_2_1.html, (Archived on 7 Jan. 2025), Jan. 2025. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/Entwicklungstand_QC_V_2_1.html.

[9] BSI, *BSI TR-02102-1: "cryptographic mechanisms: Recommendations and key lengths" version: 2024-1*, [Online]. Available from: https://web.archive.org/web/20240807092324/https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf, (Archived on 7 Aug. 2024), 2024. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf.

[10] D. Moody, R. Perlner, A. Regenscheid, A. Robinson, and D. Cooper, *Transition to Post-Quantum Cryptography Standards*. Nov. 2024. doi: 10.6028/nist.ir.8547.ipd. [Online]. Available: http://dx.doi.org/10.6028/NIST.IR.8547.ipd.

[11] G. Alagic, E. Barker, L. Chen, *et al.*, *Recommendations for key-encapsulation mechanisms*, Jan. 2025. doi: 10.6028/nist.sp.800-227.ipd. [Online]. Available: http://dx.doi.org/10.6028/NIST.SP.800-227.ipd.

[12] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, J. Buhler, Ed., ser. Lecture Notes in Computer Science, vol. 1423, Springer, 1998, pp. 267–288. doi: 10.1007/BFB0054868. [Online]. Available: https://doi.org/10.1007/BFb0054868.

[13] A. Hülsing, J. Rijneveld, J. M. Schanck, and P. Schwabe, "High-Speed Key Encapsulation from NTRU," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, W. Fischer and N. Homma, Eds., ser. Lecture Notes in Computer Science, vol. 10529, Springer, 2017, pp. 232–252. doi: 10.1007/978-3-319-66787-4_12. [Online]. Available: https://doi.org/10.1007/978-3-319-66787-4%5C_12.

[14] V. B. Dang, K. Mohajerani, and K. Gaj, "High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber," *IACR Cryptol. ePrint Arch.*, p. 1508, 2021. [Online]. Available: https://eprint.iacr.org/2021/1508.

[15] B. Peng, A. Marotzke, M. Tsai, B. Yang, and H. Chen, "Streamlined NTRU Prime on FPGA," *IACR Cryptol. ePrint Arch.*, p. 1444, 2021. [Online]. Available: https://eprint.iacr.org/2021/1444.

[16] N. Sendrier, "Secure sampling of constant-weight words – application to BIKE," *IACR Cryptol. ePrint Arch.*, p. 1631, 2021. [Online]. Available: https://eprint.iacr.org/2021/1631.

[17] S. Lin and D. J. C. Jr., *Error control coding - fundamentals and applications* (Prentice Hall computer applications in electrical engineering series). Prentice Hall, 1983, isbn: 978-0-13-283796-5.

[18] Y. Wu, "New Scalable Decoder Architectures for Reed-Solomon Codes," *IEEE Trans. Commun.*, vol. 63, no. 8, pp. 2741–2761, 2015. doi: 10.1109/TCOMM.2015.2445759. [Online]. Available: https://doi.org/10.1109/TCOMM.2015.2445759.

[19] Y. Be'ery and J. Snyders, "Optimal soft decision block decoders based on fast Hadamard transform," *IEEE Trans. Inf. Theory*, vol. 32, no. 3, pp. 355–364, 1986. doi: 10.1109/TIT.1986.1057189. [Online]. Available: https://doi.org/10.1109/TIT.1986.1057189.

[20] S. Deshpande, C. Xu, M. Nawan, K. Nawaz, and J. Szefer, "Fast and efficient hardware implementation of HQC," in *Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers*, C. Carlet, K. Mandal, and V. Rijmen, Eds., ser. Lecture Notes in Computer Science, vol. 14201, Springer, 2023, pp. 297–321. doi: 10.1007/978-3-031-53368-6_15. [Online]. Available: https://doi.org/10.1007/978-3-031-53368-6_15.

[21] C. Aguilar Melchor and et al., "Towards Automating Cryptographic Hardware Implementations: A Case Study of HQC," in *CBCrypto 2022, Trondheim, Norway, May 29-30, 2022*, ser. LCNS, vol. 13839, Springer, 2022, pp. 62–76. doi: 10.1007/978-3-031-29689-5_4. [Online]. Available: https://doi.org/10.1007/978-3-031-29689-5_4.

[22] V. B. Dang, K. Mohajerani, and K. Gaj, "High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber," *IEEE Trans. Computers*, vol. 72, no. 2, pp. 306–320, 2023. doi: 10.1109/TC.2022.3222954. [Online]. Available: https://doi.org/10.1109/TC.2022.3222954.

[23] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 2, pp. 328–356, 2021. doi: 10.46586/TCHES.V2021.I2.328-356. [Online]. Available: https://doi.org/10.46586/tches.v2021.i2.328-356.

[24] C. Li, S. Song, J. Tian, Z. Wang, and Ç. K. Koç, "An efficient hardware design for fast implementation of HQC," in *36th IEEE International System-on-Chip Conference, SOCC 2023, Santa Clara, CA, USA, September 5-8, 2023*, J. Becker, A. Marshall, T. Harbaum, A. Ganguly, F. Siddiqui, and K. McLaughlin, Eds., IEEE, 2023, pp. 1–6. doi: 10.1109/SOCC58585.2023.10257054. [Online]. Available: https://doi.org/10.1109/SOCC58585.2023.10257054.

[25] J. Richter-Brockmann, M. Chen, S. Ghosh, and T. Güneysu, "Racing BIKE: improved polynomial multiplication and inversion in hardware," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2022, no. 1, pp. 557–588, 2022. doi: 10.46586/TCHES.V2022.I1.557-588. [Online]. Available: https://doi.org/10.46586/tches.v2022.i1.557-588.

[26] P. Chen, T. Chou, S. Deshpande, *et al.*, "Complete and improved FPGA implementation of Classic McEliece," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2022, no. 3, pp. 71–113, 2022. doi: 10.46586/TCHES.V2022.I3.71-113. [Online]. Available: https://doi.org/10.46586/tches.v2022.i3.71-113.

[27] L. Beckwith, R. Wallace, K. Mohajerani, and K. Gaj, "A high-performance hardware implementation of the LESS digital signature scheme,", 2023, pp. 57–90. doi: 10.1007/978-3-031-40003-2_3.

[28] F. Hirner, M. Streibl, F. Krieger, A. C. Mert, and S. S. Roy, "Whipping the multivariate-based MAYO signature scheme using hardware platforms," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds., ACM, 2024, pp. 3421–3435. doi: 10.1145/3658644.3690258. [Online]. Available: https://doi.org/10.1145/3658644.3690258.

[29] S. Deshpande, J. Howe, J. Szefer, and D. Yue, "SDitH in hardware,", vol. 2024, no. 2, pp. 215–251, 2024. doi: 10.46586/tches.v2024.i2.215-251.

[30] L. Beckwith, D. T. Nguyen, and K. Gaj, "Hardware accelerators for digital signature algorithms dilithium and FALCON," *IEEE Des. Test*, vol. 41, no. 5, pp. 27–35, 2024. doi: 10.1109/MDAT.2023.3305156. [Online]. Available: https://doi.org/10.1109/MDAT.2023.3305156.

[31] G. Land, P. Sasdrich, and T. Güneysu, "A hard crystal - implementing Dilithium on reconfigurable hardware," in *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, V. Grosso and T. Pöppelmann, Eds., ser. Lecture Notes in Computer Science, vol. 13173, Springer, 2021, pp. 210–230. doi: 10.1007/978-3-030-97348-3_12. [Online]. Available: https://doi.org/10.1007/978-3-030-97348-3_12.

[32] C. Zhao, N. Zhang, H. Wang, *et al.*, "A compact and high-performance hardware architecture for CRYSTALS-Dilithium," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2022, no. 1, pp. 270–295, 2022. doi: 10.46586/TCHES.V2022.I1.270-295. [Online]. Available: https://doi.org/10.46586/tches.v2022.i1.270-295.

[33] Z. Wu, R. Chen, Y. Wang, Q. Wang, and W. Peng, "An efficient hardware implementation of Crystal-Dilithium on FPGA," in *Information Security and Privacy - 29th Australasian Conference, ACISP 2024, Sydney, NSW, Australia, July 15-17, 2024, Proceedings, Part II*, T. Zhu and Y. Li, Eds., ser. Lecture Notes in Computer Science, vol. 14896, Springer, 2024, pp. 64–83. doi: 10.1007/978-981-97-5028-3_4. [Online]. Available: https://doi.org/10.1007/978-981-97-5028-3%5C_4.

[34] X. Li, J. Lu, D. Liu, A. Li, S. Yang, and T. Huang, "A high speed post-quantum crypto-processor for Crystals-Dilithium," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 71, no. 1, pp. 435–439, 2024. doi: 10.1109/TCSII.2023.3304416. [Online]. Available: https://doi.org/10.1109/TCSII.2023.3304416.

[35] D. Amiet, L. Leuenberger, A. Curiger, and P. Zbinden, "FPGA-based sphincs$^+$ implementations: Mind the glitch," in *23rd Euromicro Conference on Digital System Design, DSD 2020, Kranj, Slovenia, August 26-28, 2020*, IEEE, 2020, pp. 229–237. doi: 10.1109/DSD51259.2020.00046. [Online]. Available: https://doi.org/10.1109/DSD51259.2020.00046.

[36] S. Deshpande, Y. Lee, C. Karakuzu, J. Szefer, and Y. Paek, "SPHINCSLET: An area-efficient accelerator for the full SPHINCS+ digital signature algorithm," *ACM Trans. Embed. Comput. Syst.*, Apr. 2025, Just Accepted, issn: 1539-9087. doi: 10.1145/3728469. [Online]. Available: https://doi.org/10.1145/3728469.

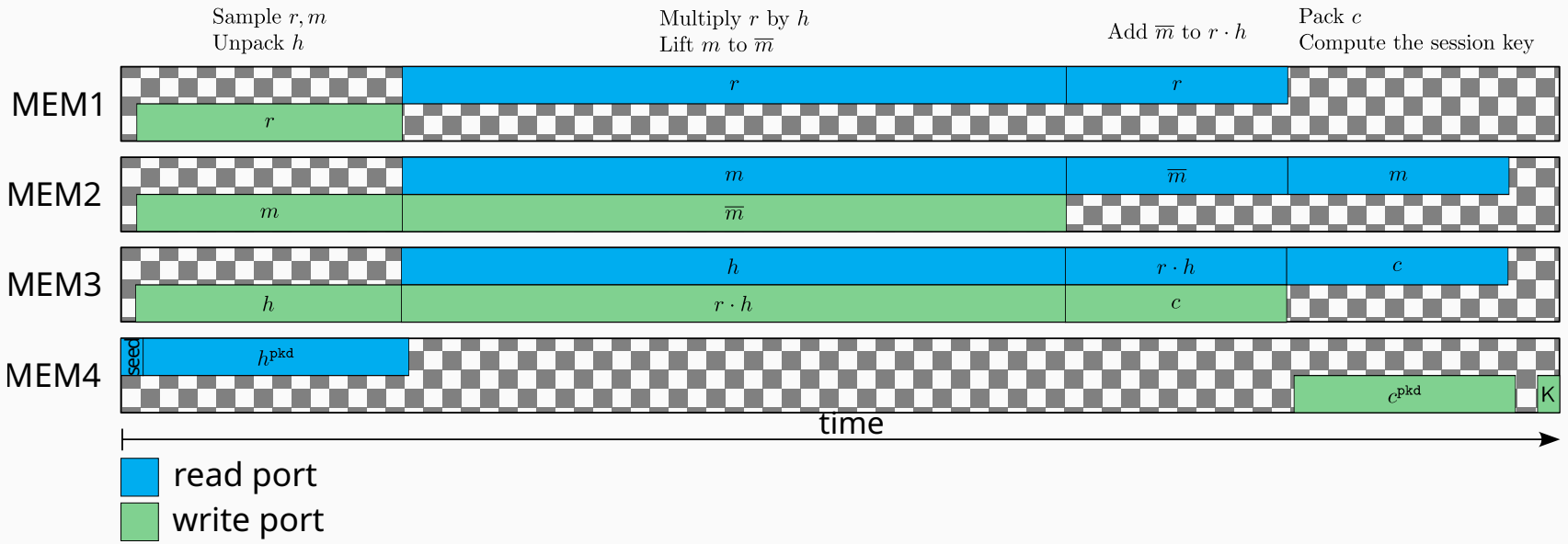
[37] Y. Ouyang, Y. Zhu, W. Zhu, *et al.*, "FalconSign: An efficient and high-throughput hardware architecture for Falcon signature generation," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2025, no. 1, pp. 203–226, 2025. doi: 10.46586/TCHES.V2025.I1.203-226. [Online]. Available: https://doi.org/10.46586/tches.v2025.i1.203-226.

[38] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of CRYSTALS-Dilithium," in *International Conference on Field-Programmable Technology, (IC)FPT 2021, Auckland, New Zealand, December 6-10, 2021*, IEEE, 2021, pp. 1–10. doi: 10.1109/ICFPT52863.2021.9609917. [Online]. Available: https://doi.org/10.1109/ICFPT52863.2021.9609917.

[39] W. Wang, S. Tian, B. Jungk, N. Bindel, P. Longa, and J. Szefer, "Parameterized hardware accelerators for lattice-based cryptography and their application to the HW/SW co-design of qTESLA," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 269–306, 2020. doi: 10.13154/TCHES.V2020.I3.269-306. [Online]. Available: https://doi.org/10.13154/tches.v2020.i3.269-306.

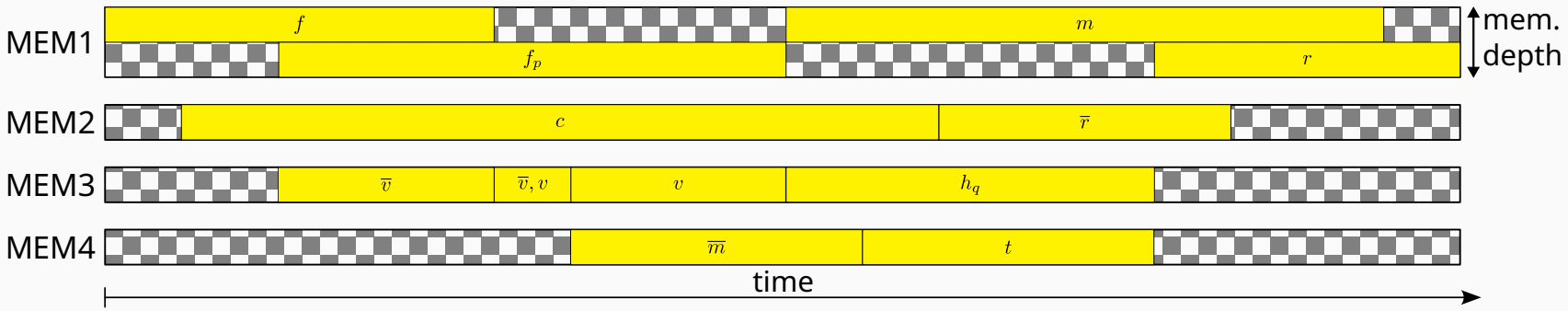
Backup slides

Memory constraints
Memory access constraint (memory port binding)



Binding of operations to memory ports for the **NTRU-HRSS encapsulation** algorithm

Memory constraints
Memory size constraint (variable liveness)



Variable liveness analysis for the **NTRU-HPS decapsulation** algorithm

CROSS – Experimental results
Artix-7 FPGA synthesis results[2]

Design	Parameter set	Area eSlice	Freq. MHz	KeyGen. μs	KeyGen. AT	Sign μs	Sign AT	Verify μs	Verify AT
This work	CROSS-RSDP-1-f	13087	119	35	467	764	10006	584	7653
	CROSS-RSDP-1-b	15889	113	37	596	1444	22951	1016	16157
	CROSS-RSDP-1-s	27462	115	36	1013	2858	78508	1960	53841
	CROSS-RSDP-3-f	22885	110	82	1886	1905	43609	1455	33310
	CROSS-RSDP-3-b	28283	108	84	2380	3230	91355	2067	58488
	CROSS-RSDP-3-s	34120	105	86	2949	4984	170081	3037	103653
	CROSS-RSDP-5-f	33591	105	153	5170	3762	126370	2843	95514
	CROSS-RSDP-5-b	44720	101	160	7157	6297	281646	3744	167460
	CROSS-RSDP-5-s	61796	101	160	9887	10153	627431	5564	343849
This work	CROSS-RSDPG-1-f	15452	100	10	163	601	9300	477	7376
	CROSS-RSDPG-1-b	17396	103	10	178	1216	21154	886	15421
	CROSS-RSDPG-1-s	23005	102	10	239	2433	55979	1762	40540
	CROSS-RSDPG-3-f	18377	96	22	413	1260	23162	1033	18998
	CROSS-RSDPG-3-b	23795	102	21	503	1604	38185	1201	28593
	CROSS-RSDPG-3-s	35344	99	21	770	3103	109699	2214	78267
	CROSS-RSDPG-5-f	27749	99	35	997	2223	61701	1860	51637
	CROSS-RSDPG-5-b	34512	96	37	1278	2960	102183	2222	76717
	CROSS-RSDPG-5-s	47332	98	36	1715	5133	242993	3613	171042
[30]	Dilithium-II	21330	185	12.2	260	98.3	2096	14.4	307
	Dilithium-III			22.6	482	166.7	3555	25.4	541
	Dilithium-V			30.3	646	196.3	4187	33.2	708
[31]	Dilithium-II	16091	163	115	1850	178/470	2864/7562	121	1947
	Dilithium-III	18230	145	228	4156	310/850	5651/15495	221	4028
	Dilithium-V	23788	140	363	8635	503/1042	11965/24787	377	8968
[38]	Dilithium-V	21597	116	121	2613	2520	54424	21	453
[35]	SPHINCS+-128f-s	14571	250/500	–	–	1010	14716	16	233
	SPHINCS+-192f-s	15838	250/500	–	–	1170	18530	19	300
	SPHINCS+-256f-s	17657	250/500	–	–	2520	44495	21	370

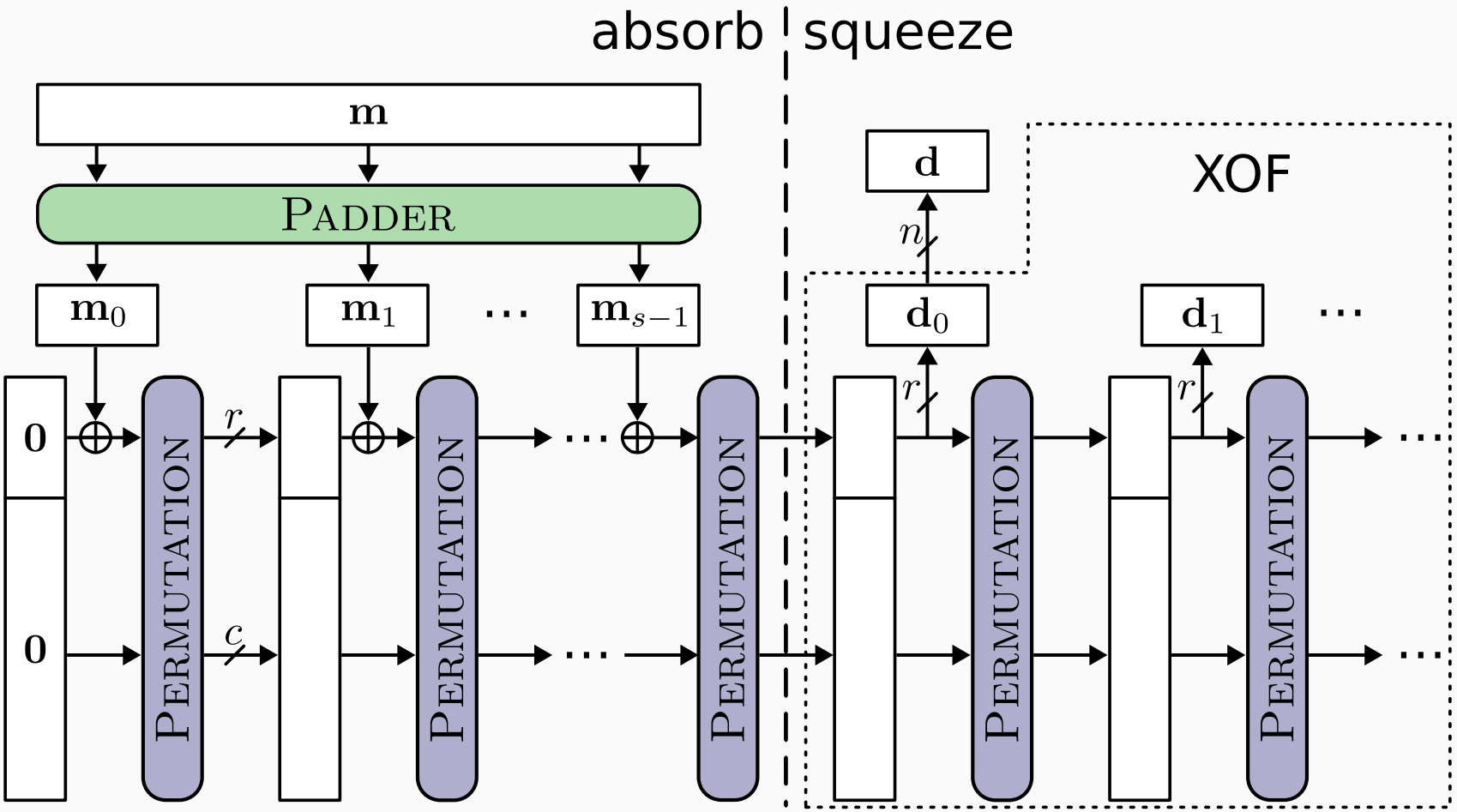
[2] Credits to Patrick Karl from the Technical University of Munich.

Assessment multiplier algorithms for HQC

We simulated different approaches for the $\mathbb{F}_2[x]$ multiplications required in the code-based HQC cryptoscheme.

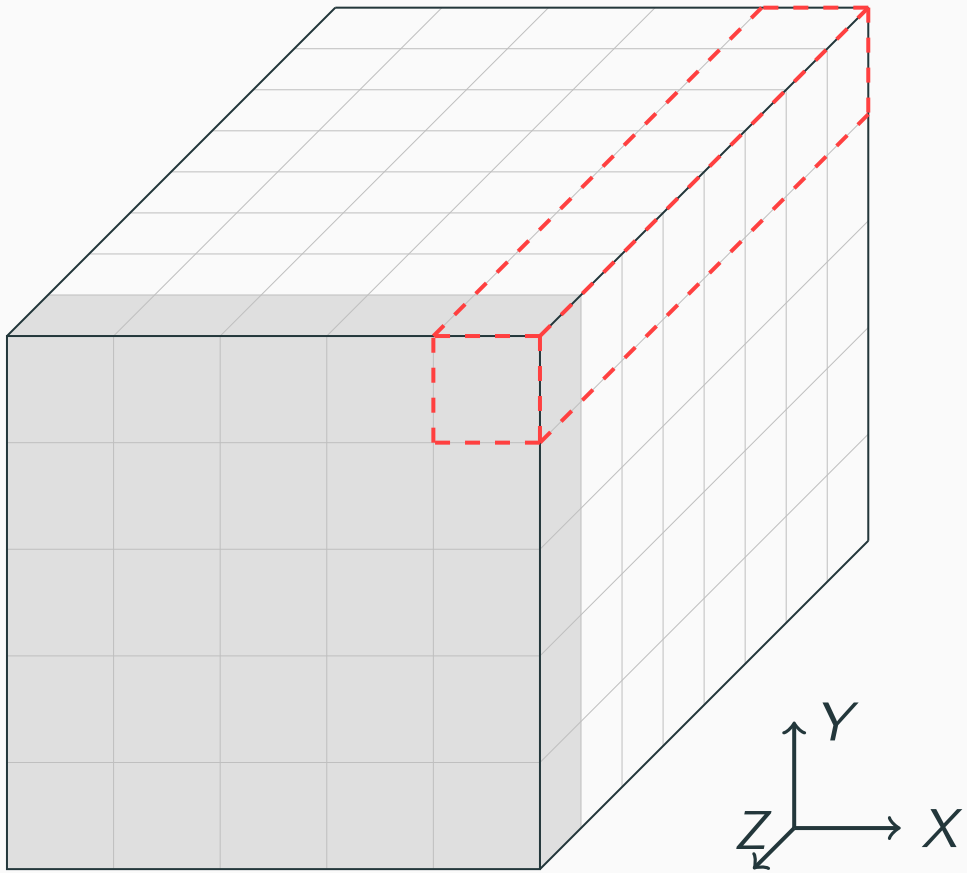
Block size	Mult. algorithm	Latency (kCC)		
		hqc-128	hqc-192	hqc-256
32	Comba	305.80	1256.64	3247.20
32	Karatsuba 5-terms + NTT	76.03	230.91	464.64
32	Karatsuba 5-terms + Comba	237.31	714.75	1432.32
32	Rotate and accumulate (d=1)	41.47	127.79	268.49
32	Rotate and accumulate (d=2)	20.73	63.89	134.24
32	Rotate and accumulate (d=4)	10.36	31.94	67.12
64	Comba	76.72	314.72	811.80
64	Karatsuba 5-terms + NTT	34.43	104.70	210.81
64	Karatsuba 5-terms + Comba	61.31	185.34	372.09
64	Rotate and accumulate (d=1)	20.77	63.95	134.24
64	Rotate and accumulate (d=2)	10.38	31.97	67.12
64	Rotate and accumulate (d=4)	5.19	15.98	33.56
128	Comba	19.32	78.96	203.40
128	Karatsuba 5-terms + NTT	15.42	46.97	94.65
128	Karatsuba 5-terms + Comba	16.32	49.66	100.03
128	Rotate and accumulate (d=1)	10.42	32.03	67.19
128	Rotate and accumulate (d=2)	5.21	16.01	33.60
128	Rotate and accumulate (d=4)	2.60	8.00	16.80

Keccak core



Sponge construction

Keccak core



Keccak state representation

Keccak core

Keccak FPGA cores synthesized on an Artix-7

Design			Area			Frequency	SHA3-256			
Ref.	Type	Variant	LUT	FF	eSlice	MHz	CC	μs	AT prod.	Gb/s
[39]	mid-range*	×1	811	490	203	178	2681	15.06	3.06	0.07
[20]	mid-range	×1	1437	498	360	163	2408	14.77	5.32	0.07
[39]	mid-range*	×2	908	450	227	163	1353	8.30	1.88	0.13
[20]	mid-range	×2	1558	466	390	167	1206	7.22	2.82	0.15
[39]	mid-range*	×4	1069	361	268	158	680	4.30	1.15	0.25
[20]	mid-range	×4	1625	370	407	157	604	3.85	1.57	0.28
[39]	mid-range*	×8	1466	270	367	164	337	2.05	0.75	0.52
[20]	mid-range	×8	1958	280	490	158	302	1.91	0.94	0.56
[39]	mid-range*	×16	2401	226	601	165	168	1.02	0.61	1.04
[20]	mid-range	×16	2819	236	705	164	150	0.91	0.64	1.17
[39]	mid-range*	×32	4436	180	1109	161	85	0.53	0.59	2.00
[20]	mid-range	×32	4797	191	1200	166	74	0.45	0.53	2.36
This work	high-speed	×1	5589	2744	1398	237	29	0.12	0.17	8.85
This work	high-speed	×2	10571	2736	2643	125	17	0.14	0.36	7.59
This work	high-speed	×3	12700	2719	3175	74	13	0.18	0.56	5.90
This work	high-speed	×4	15472	2717	3868	53	11	0.21	0.80	5.06
This work	high-speed	×6	22725	2715	5682	28	9	0.32	1.83	3.32
This work	high-speed	×8	20262	2714	5066	26	8	0.31	1.56	3.43
This work	high-speed	×12	28782	2713	7196	14	7	0.50	3.60	2.12
This work	high-speed	×24	55403	2714	13851	6	6	1.00	13.85	1.06