POLITECNICO DI MILANO

A Flexible ASIC-oriented Design for a Full NTRU Accelerator

Francesco Antognazza, Alessandro Barenghi, Gerardo Pelosi, Ruggero Susella

ASP-DAC 2023, Tokyo

Background

NTRU KEM hard problem:

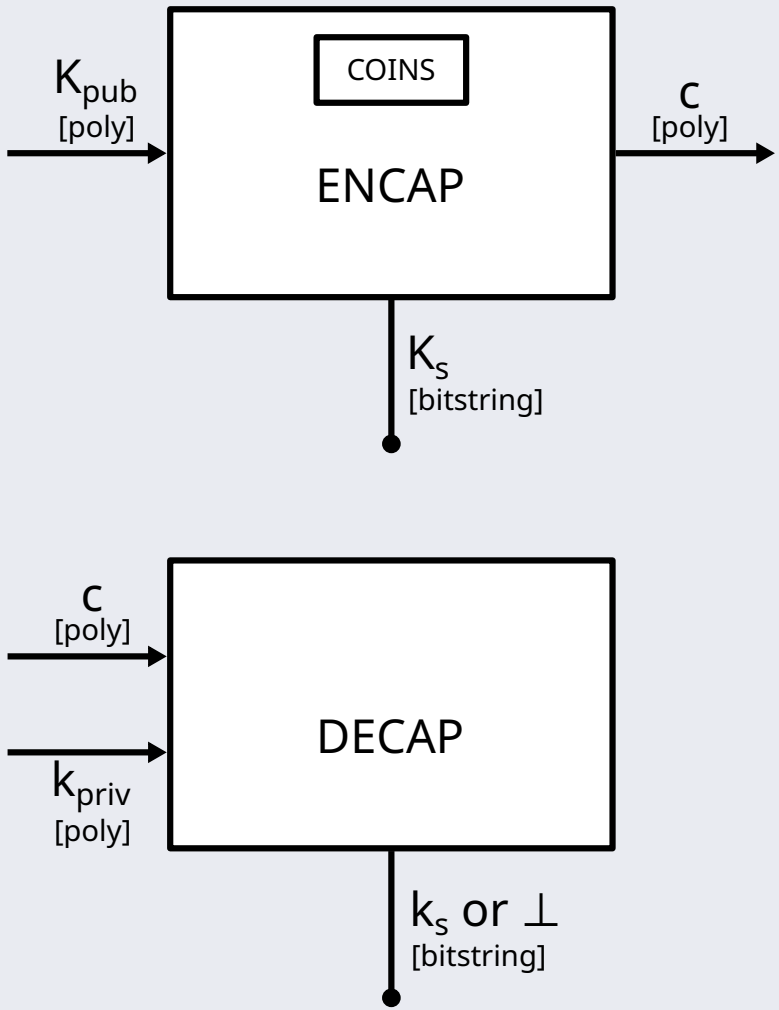
- given a poly $\mathbf{h} = \mathbf{g}^{-1} \cdot \mathbf{f}$, find $\mathbf{f}$ and $\mathbf{g}$ with small coefficients
- can be reduced to solve SVP or CVP over a lattice

Main features:

- faster than RSA
- 699-1230 B public key/ciphertext
- patent-free

Standards and applications:

- IEEE 1363.1 (NTRU ver. 2008)
- mainline in OpenSSH (NTRU Prime NIST 2022)
- ongoing IETF RFC preparation (NTRU HPS/HRSS NIST 2022)



Background

Challenges and goals

- NTRU candidate in NIST PQC contest: NTRU HPS (AES-{128,192,256} equiv. params) and NTRU HRSS (AES-192 equiv. params.)
- flexible architecture to perform a design space exploration
  - completely decoupled modules to easily replace any algorithm
  - modules scaling performance with the memory bus widths
- target an ASIC oriented design

Results improving state-of-the-art

- the latency and Area×Time products of our speed-oriented FPGA designs outperform current state-of-the-art solutions
- area optimized design only 20% larger than the inner SHA-3 module

  e.g. for encap ntruhps2048677
  462 kCC @ 750 MHz vs. 820 kCC @ 24 MHz ARM C-M4 [1]
- speed oriented design against Intel Xeon E3-1220 (Haswell) [2]

  more than $1.47\times$ ***(encap) and*** $2.19\times$ ***(decap) for NTRU HPS***

Outline

1. NTRU parameters
2. Encapsulation and decapsulation algorithms
3. Arithmetic modules in the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q/\langle x^n - 1\rangle$
   - adder
   - multiplier
   - lift/embed
   - sampler
4. Scheduling of encap and decap inner operations
5. Design Space Exploration results
6. Conclusions

NTRU parameters

NTRU parameters

NTRU makes use of arithmetic in the quotient polynomial ring

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

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

$\mathbf{f} \in \mathcal{R}, \mathbf{f} = f_{n-1} x^{n-1} + \ldots + f_0$ equiv. to $\mathbf{f} \in \mathbb{Z}^n, \mathbf{f} = (f_{n-1}, \cdots, f_0)$

NTRU parameters

NTRU makes use of arithmetic in the quotient polynomial ring

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

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

$\mathbf{f} \in \mathcal{R}, \mathbf{f} = f_{n-1} x^{n-1} + \ldots + f_0$ equiv. to $\mathbf{f} \in \mathbb{Z}^n, \mathbf{f} = (f_{n-1}, \cdots, f_0)$
The inner workings of the scheme are over:
$\mathcal{R}_q = \mathbb{Z}_q[x] / \langle \Phi_1 \Phi_n \rangle \qquad \mathcal{S}_q = \mathbb{Z}_q[x] / \langle \Phi_n \rangle \qquad \mathcal{S}_p = \mathbb{Z}_p[x] / \langle \Phi_n \rangle$

NTRU parameters

NTRU makes use of arithmetic in the quotient polynomial ring

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

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

$\mathbf{f} \in \mathcal{R}, \mathbf{f} = f_{n-1} x^{n-1} + \ldots + f_0$ equiv. to $\mathbf{f} \in \mathbb{Z}^n, \mathbf{f} = (f_{n-1}, \cdots, f_0)$
The inner workings of the scheme are over:
$\mathcal{R}_q = \mathbb{Z}_q[x] / \langle \Phi_1 \Phi_n \rangle \qquad \mathcal{S}_q = \mathbb{Z}_q[x] / \langle \Phi_n \rangle \qquad \mathcal{S}_p = \mathbb{Z}_p[x] / \langle \Phi_n \rangle$

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

NTRU parameters

NTRU makes use of arithmetic in the quotient polynomial ring

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

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

$\mathbf{f} \in \mathcal{R}, \mathbf{f} = f_{n-1} x^{n-1} + \ldots + f_0$ equiv. to $\mathbf{f} \in \mathbb{Z}^n, \mathbf{f} = (f_{n-1}, \cdots, f_0)$
The inner workings of the scheme are over:
$\mathcal{R}_q = \mathbb{Z}_q[x] / \langle \Phi_1 \Phi_n \rangle \qquad \mathcal{S}_q = \mathbb{Z}_q[x] / \langle \Phi_n \rangle \qquad \mathcal{S}_p = \mathbb{Z}_p[x] / \langle \Phi_n \rangle$

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

Further constraints to make a deterministic cryptographic scheme:

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

NTRU parameters

NTRU makes use of arithmetic in the quotient polynomial ring

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

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

$\mathbf{f} \in \mathcal{R}, \mathbf{f} = f_{n-1} x^{n-1} + \ldots + f_0$ equiv. to $\mathbf{f} \in \mathbb{Z}^n, \mathbf{f} = (f_{n-1}, \cdots, f_0)$
The inner workings of the scheme are over:
$\mathcal{R}_q = \mathbb{Z}_q[x] / \langle \Phi_1 \Phi_n \rangle \qquad \mathcal{S}_q = \mathbb{Z}_q[x] / \langle \Phi_n \rangle \qquad \mathcal{S}_p = \mathbb{Z}_p[x] / \langle \Phi_n \rangle$

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

Further constraints to make a deterministic cryptographic scheme:

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

The NTRU KEM scheme employs $k_{\text{pub}} = \mathbf{h}$, with $\mathbf{h} = \mathbf{g}^{-1} \cdot \mathbf{f}$, where $\mathbf{f}$ and $\mathbf{g}$ are small polys and $\mathbf{h}$ is a large poly, $k_{\text{priv}} = \mathbf{f}$

Encap and decap algorithms

Key Encapsulation Mechanism (KEM)

Encapsulation

Input: public key: h^{pkd}
Output: ciphertext c^{pkd}
session key: k kept by sender

$coins \xleftarrow{\$} \{0,1\}^{320}$
$(\mathbf{r}, \mathbf{m}) \leftarrow \text{SAMPLE\_}rm(coins)$ /* sample of two random small polynomials */
$r^{\text{pkd}} \leftarrow \text{PACK}_p(\mathbf{r})$, $m^{\text{pkd}} \leftarrow \text{PACK}_p(\mathbf{m})$
$k \leftarrow \text{SHA3-256}(r^{\text{pkd}} || m^{\text{pkd}})$
$\mathbf{h} \leftarrow \text{UNPACK}_q(h^{\text{pkd}})$
$\mathbf{m}' \leftarrow \texttt{Lift}(\mathbf{m})$ /* Lift to $\mathcal{R}_q$ ring */
$\mathbf{c} \leftarrow (\mathbf{r} \circledast \mathbf{h} + \mathbf{m}') \bmod (q, x^n - 1)$ /* small-by-large multiplication, addition */
$c^{\text{pkd}} \leftarrow \text{PACK}_q(c)$

return c^{pkd}, k

Polynomials	HPS	HRSS
r	variable-weight	variable-weight
m	fixed-weight (d)	variable-weight

Key Encapsulation Mechanism (KEM)

Decapsulation

Input: private key: f^{pkd}, f_p^{pkd}, h_q^{pkd}, 256-bit string s
ciphertext: c^{pkd}

Output: session key: k kept by receiver

$\mathbf{f} \leftarrow \text{UnPack}_p(f^{\text{pkd}})$, $\mathbf{f}_p \leftarrow \text{UnPack}_p(f_p^{\text{pkd}})$

$\mathbf{h}_q \leftarrow \text{UnPack}_q(h_q^{\text{pkd}})$, $\mathbf{c} \leftarrow \text{UnPack}_q(c^{\text{pkd}})$

$\mathbf{a} \leftarrow (\mathbf{c} \circledast \mathbf{f}) \bmod (q, x^n - 1)$ /* small-by-large multiplication */

$\mathbf{m} \leftarrow (\mathbf{a} \circledast \mathbf{f}_p) \bmod (p, \Phi_n)$ /* small-by-large multiplication */

$\mathbf{m}' \leftarrow \texttt{Lift}(\mathbf{m})$ /* Lift to $\mathcal{R}_q$ ring */

$\mathbf{r} \leftarrow ((\mathbf{c} - \mathbf{m}') \circledast \mathbf{h}_q) \bmod (q, \Phi_n)$ /* subtraction, large-by-large multiplication */

$r^{\text{pkd}} \leftarrow \text{Pack}_p(\mathbf{r})$, $m^{\text{pkd}} \leftarrow \text{Pack}_p(\mathbf{m})$

$k_1 \leftarrow \text{SHA3-256}(r^{\text{pkd}} || m^{\text{pkd}})$, $k_2 \leftarrow \text{SHA3-256}(s || c^{\text{pkd}})$

if $\mathbf{c} \not\equiv 0 \bmod (q, \Phi_1) \vee (\mathbf{r}, \mathbf{m}) \notin \mathcal{L}_r \times \mathcal{L}_m$ **then return** k_1 **else return** k_2

Arithmetic modules in the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q/\langle x^n - 1 \rangle$

Arithmetic in polynomial ring $\mathcal{R}_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$

Polynomial addition

Let $\mathbf{a}, \mathbf{b}$ be two polynomials in $\mathcal{R}_q$, their sum $\mathbf{c} = \mathbf{a} + \mathbf{b}$ has coefficients

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

Polynomial product (circular convolution)

Let $\mathbf{a}, \mathbf{b}$ be two polynomials in $\mathcal{R}_q$, their product $\mathbf{c} = \mathbf{a} \circledast \mathbf{b}$ has coefficients

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

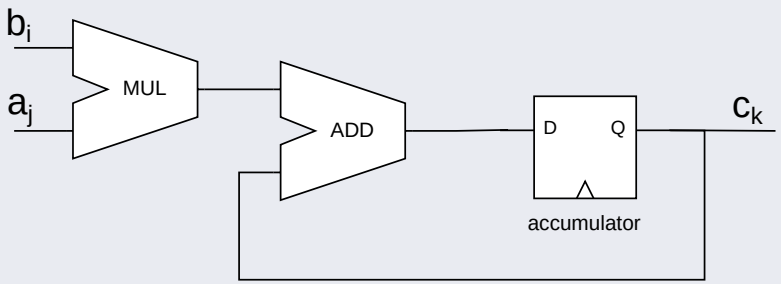
Operands and result are stored in three simple dual port memories

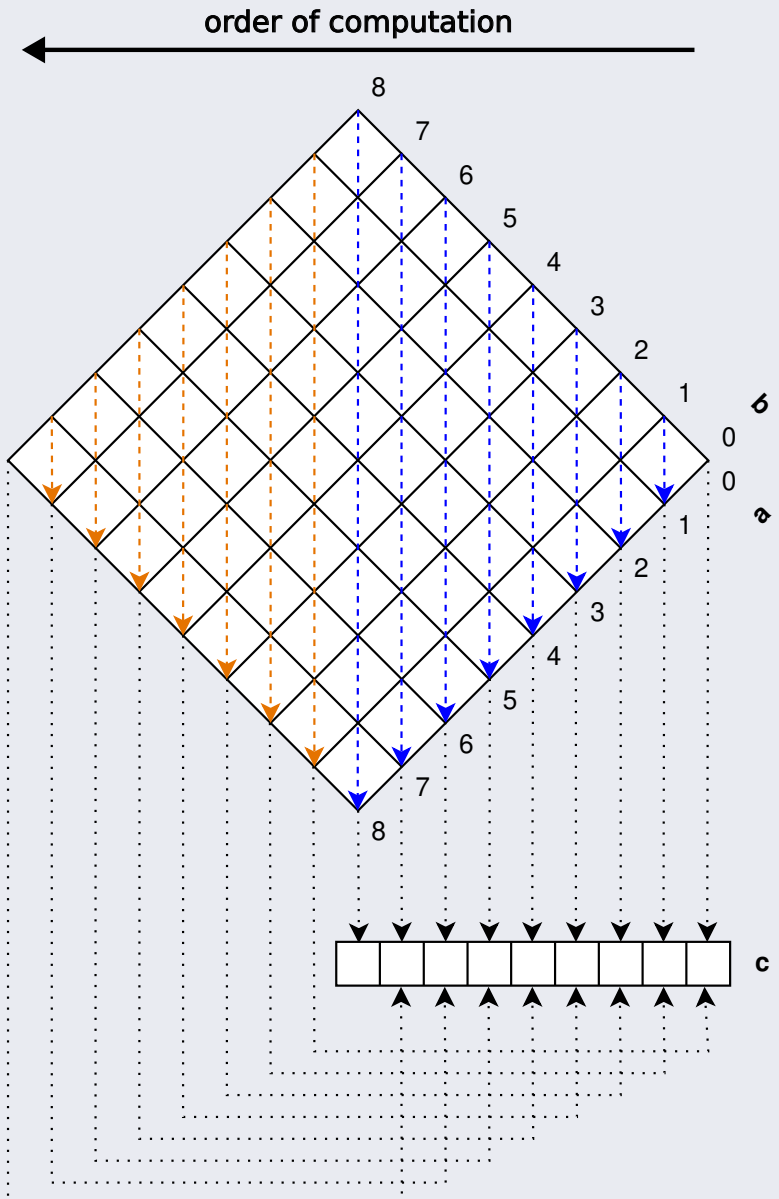
Polynomial multiplier in $\mathcal{R}_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$ - Comba algorithm

Computes results in $\mathcal{R}_q$ with minimum write access number

Input: $\mathbf{a} \in \mathcal{R}_q$, $\mathbf{b} \in \mathcal{R}_q$
Output: $\mathbf{c} \in \mathcal{R}_q \mid \mathbf{c} = \mathbf{a} \circledast \mathbf{b}$

for $i := 0$ **to** $(n-1)$ **do**
  $c_i \leftarrow \sum_{k=0}^{i} a_k \cdot b_{i-k}$
for $i := n$ **to** $(2n-2)$ **do**
  $c_{i-n} \leftarrow c_{i-n} + \sum_{k=i+1-n}^{n-1} a_k \cdot b_{i-k}$
return c





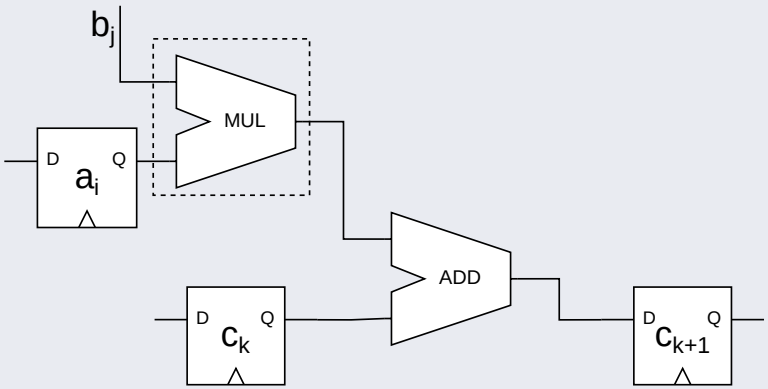
Polynomial multiplier in $\mathcal{R}_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$ - x-net algorithm

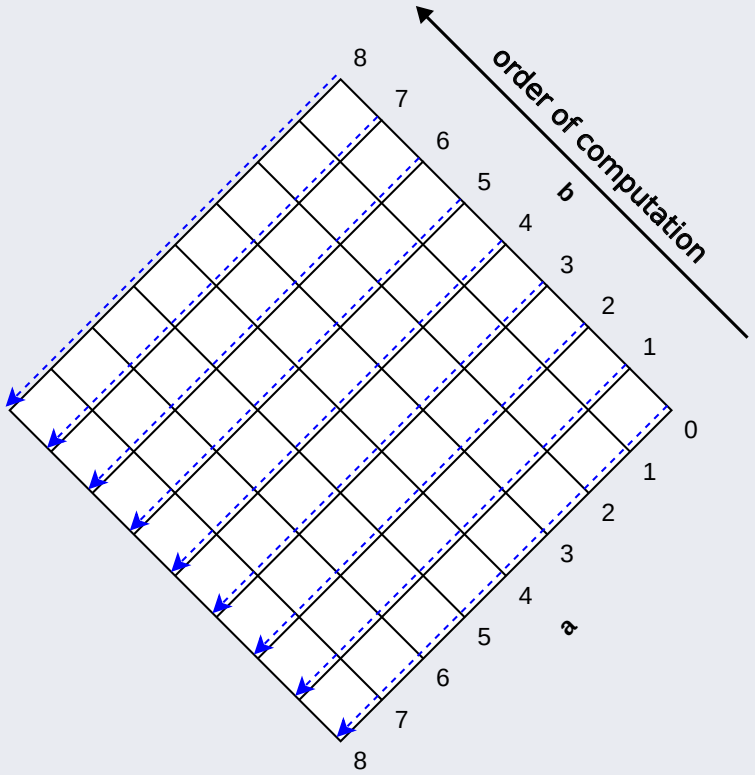
c $=$ **a** $\circledast$ **b**, **a** copied in FFs
c in FFs copied to memory

One coeff. of **b** processed per CC by n MAC units

Input: **a** $\in \mathcal{R}_q$, **b** $\in \mathcal{R}_q$
Output: **c** $\in \mathcal{R}_q \mid$ **c** $=$ **a** $\circledast$ **b**

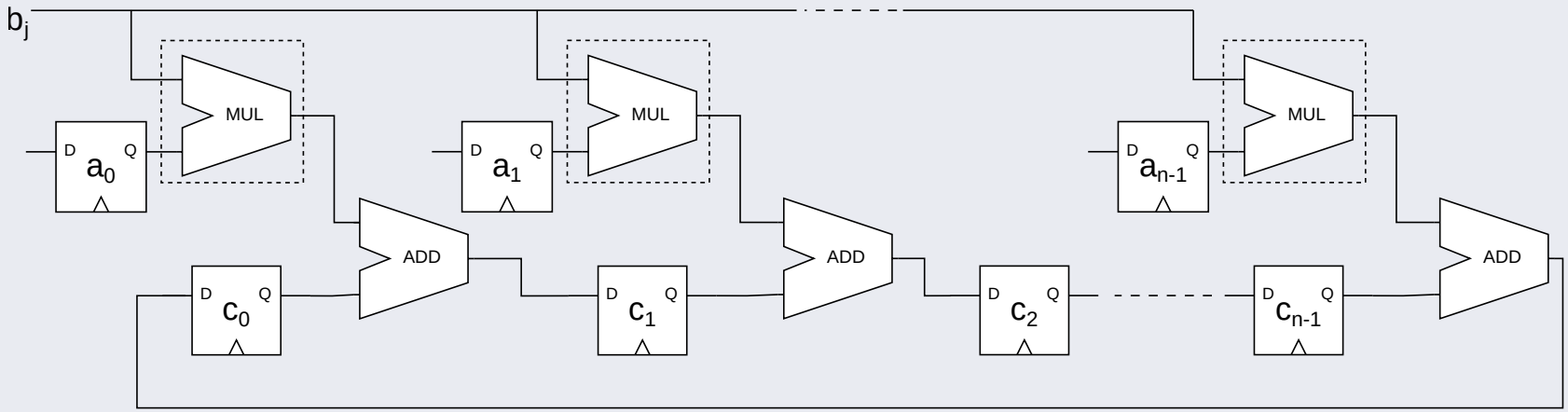
forall c_i **in c do** $c_i \leftarrow 0$
for $j := n-1$ **to** 0 **do**
  parallel for $i := 0$ **to** $n-1$ **do**
    $c_{(i+j) \bmod n} \leftarrow c_{(i+j) \bmod n} + a_i \cdot b_j$
return c





Polynomial multiplier in $\mathcal{R}_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$ - x-net algorithm

Reduction in $\mathcal{R}_q$, *i.e.*mod $x^n - 1$, is performed at every CC by adopting a LFSR structure with a trivial feedback network



The modular multiplication between polynomials with *n* coefficients is performed in *n* CC

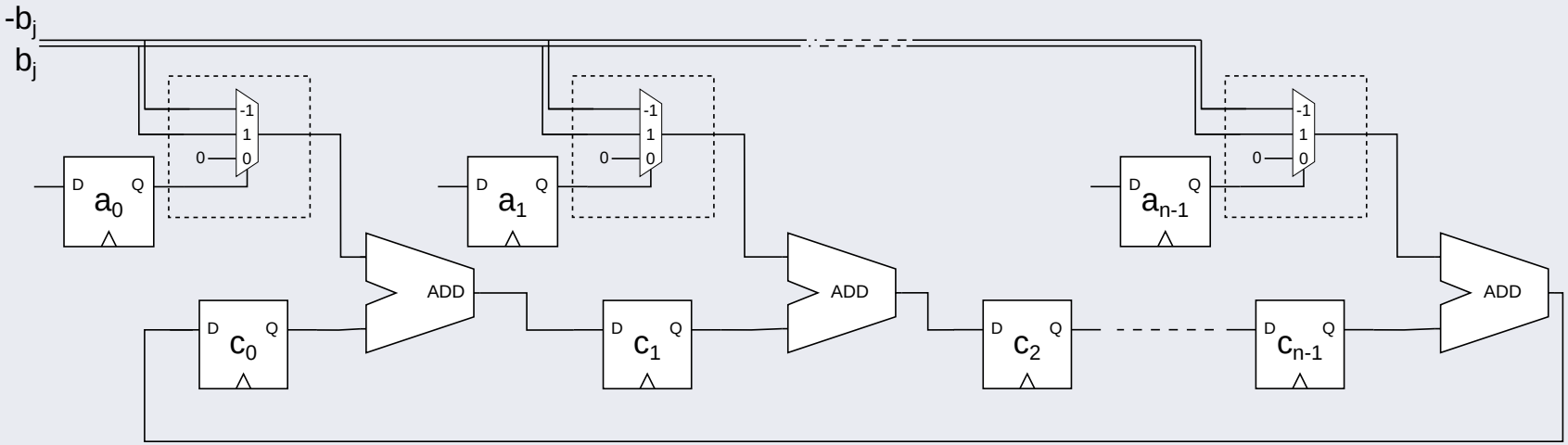
Realizing a small by large polynomial multiplication

$$\mathbf{c} = \mathbf{a} \circledast \mathbf{b} \qquad \mathbf{a} \in \mathcal{R}_p = \mathbb{Z}_3[x]/\langle x^n - 1 \rangle \qquad \mathbf{b}, \mathbf{c} \in \mathcal{R}_q$$

- replace the scalar multiplier with a MUX selecting among $\{-b_i, 0, b_i\}$

x-net based multiplier

- copy the small polynomial **a** locally into the LFSR
- load multiple **a** coeff. per clock cycle
- compute once $-b_i$ and distribute $\{-b_i, b_i\}$ to the mul. units
- mitigate net delay of distributing a single **b** coeff. per CC by replicating control and data registers



Lift and embed functions

Lift

NTRU HPS and HRSS actually use three polynomial rings:

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

The `Lift` operation maps elements $\mathbf{a} \in \mathcal{S}_p$ in larger rings $\mathcal{R}_q$ such that

$$\mathbf{a}' \leftarrow \texttt{Lift}(\mathbf{a}) \Rightarrow \mathbf{a}' \bmod (p, \Phi_n) = \mathbf{a}$$

In HPS a `Lift` is the sign extension of the coefficients, whereas in HRSS

$$\texttt{Lift} : \mathbf{a} \rightarrow \Phi_1 \cdot ((\mathbf{a}/\Phi_1) \bmod (p, \Phi_n))$$

Embed

Two maps are used, taking an element a from the larger ring $\mathcal{R}_q$ to the smaller ones $\mathcal{S}_q$ and $\mathcal{S}_p$, performing $\mathbf{a} \bmod (q, \Phi_n)$ and $\mathbf{a} \bmod (p, \Phi_n)$, respectively

Computing lift $\mathcal{S}_p \mapsto \mathcal{R}_q$

We implemented the algorithm of the NTRU HRSS paper of Hülsing et al. [3], which computes `Lift` as a sequence of additions

Input: $\mathbf{a} \in \mathcal{S}_p, p = 3$
Output: $\mathbf{b} \in \mathcal{R}_q \mid \mathbf{b} \bmod (p, \Phi_n) = \mathbf{a}$

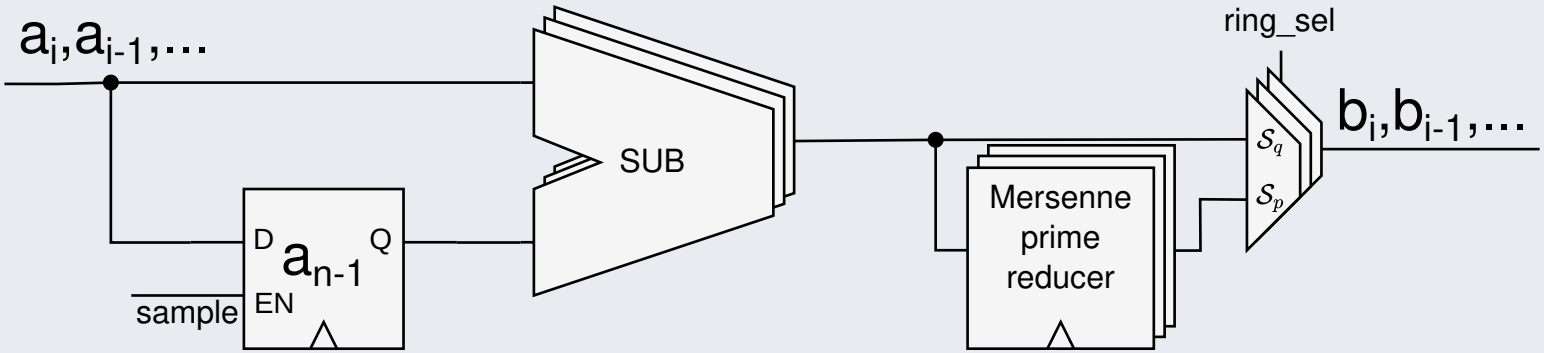
for $i := 0$ **to** $(n-2)$ **do**
  $c_i \leftarrow (1-i) \bmod p$ ▷ $\mathbf{c} \leftarrow 1/\Phi_1 \bmod (p, \Phi_n)$ for NTRU parameters; dynamically generated
for $i := 0$ **to** $(p-1)$ **do**
  $d_i \leftarrow \langle x^i \bar{\mathbf{c}}, \mathbf{a} \rangle$ ▷ 3 inner-product as sum or sub of $\mathbf{a}$ coeff.; $x^i \bar{\mathbf{c}} \in \{-1, 0, 1\}$ as $p = 3$
for $i := p$ **to** $(n-1)$ **do**
  $d_i \leftarrow d_{i-p} - \sum_{j=0}^{p-1} a_{i-j}$
$d_0 \leftarrow d_0 - d_{n-1} \bmod p$
$b_0 \leftarrow -d_0$
for $i := 1$ **to** $(n-1)$ **do**
  $d_i \leftarrow d_i - d_{n-1} \bmod p$
  $b_i \leftarrow d_{i-1} - d_i \bmod q$ ▷ multiplication by Φ_1
return b

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

Computing embed $\mathcal{R}_q \mapsto \mathcal{S}_q$ and $\mathcal{R}_q \mapsto \mathcal{S}_p$

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

If $\mathcal{S} = \mathcal{S}_p$ the coefficient-wise reductions modulo p are computed with a pipelined Mersenne prime reduction algorithm (with $p = 3$ it exhibits smallest area)



Random polynomial in $\mathcal{S}_p$ from a uniform distribution of coins

Polynomials	HPS	HRSS
r	variable-weight	variable-weight
m	fixed-weight (d)	variable-weight

Random polynomial in $\mathcal{S}_p$ from a uniform distribution of coins

Polynomials	HPS	HRSS
r	variable-weight	variable-weight
m	fixed-weight (d)	variable-weight

Variable-weight small polynomials

Two strategy for sampling each small ternary coefficients:

- reduce an 8-bit number **modulo** 3 through a Mersenne prime algorithm (constant execution time, approximated uniform distribution)
- **rejection** of the single invalid encoding in a 2-bit number (fewer bits from PRNG, perfect uniform distribution, variable execution time)

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

Random polynomial in $\mathcal{S}_p$ from a uniform distribution of coins

Polynomials	HPS	HRSS
r	variable-weight	variable-weight
m	fixed-weight (d)	variable-weight

Variable-weight small polynomials

Two strategy for sampling each small ternary coefficients:

- reduce an 8-bit number **modulo** 3 through a Mersenne prime algorithm (constant execution time, approximated uniform distribution)
- **rejection** of the single invalid encoding in a 2-bit number (fewer bits from PRNG, perfect uniform distribution, variable execution time)

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

Fixed-weight small polynomials

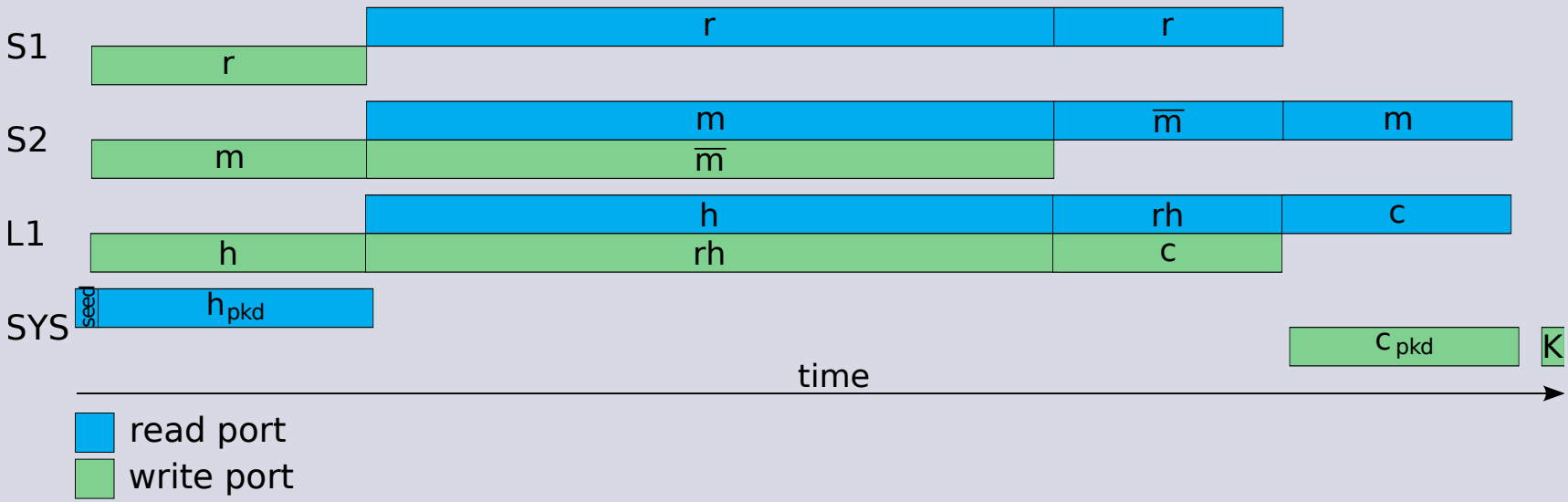
Generate a polynomial with the first d coefficients set as 1, and the following d coefficients set as -1, then scramble it

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

Scheduling of encap and decap

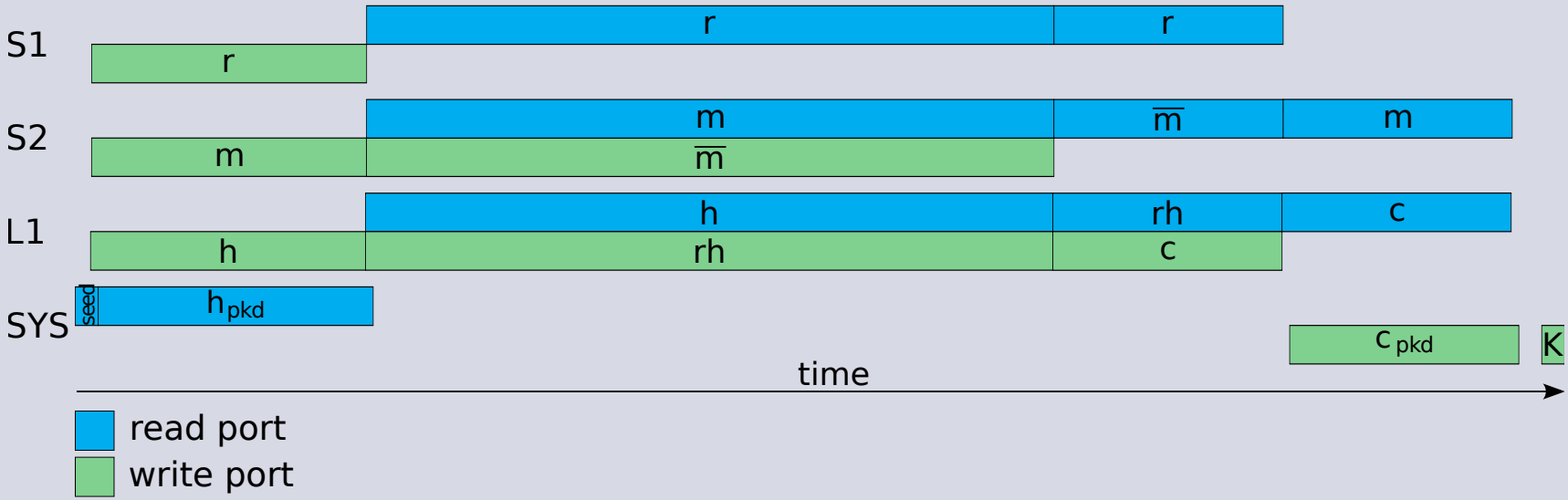
Memory constraints

Memory access constraint (memory port binding)

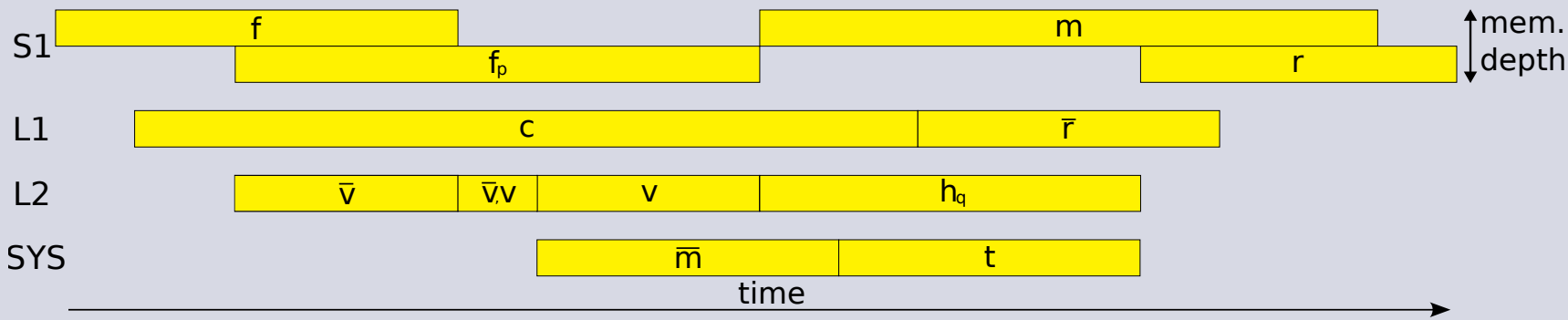


Memory constraints

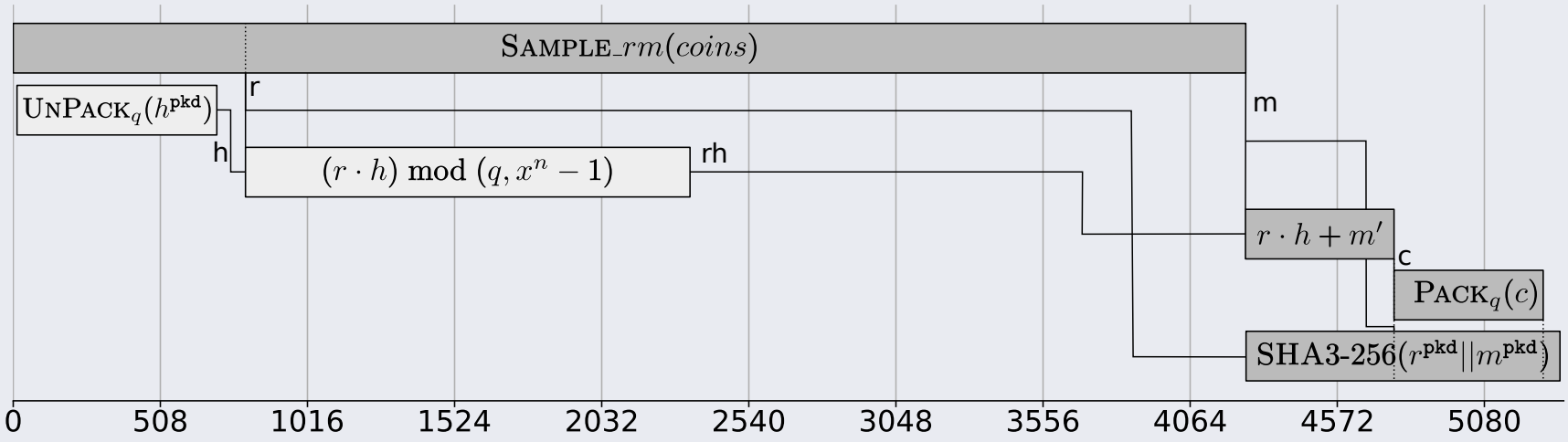
Memory access constraint (memory port binding)



Memory size constraint (variable liveness)



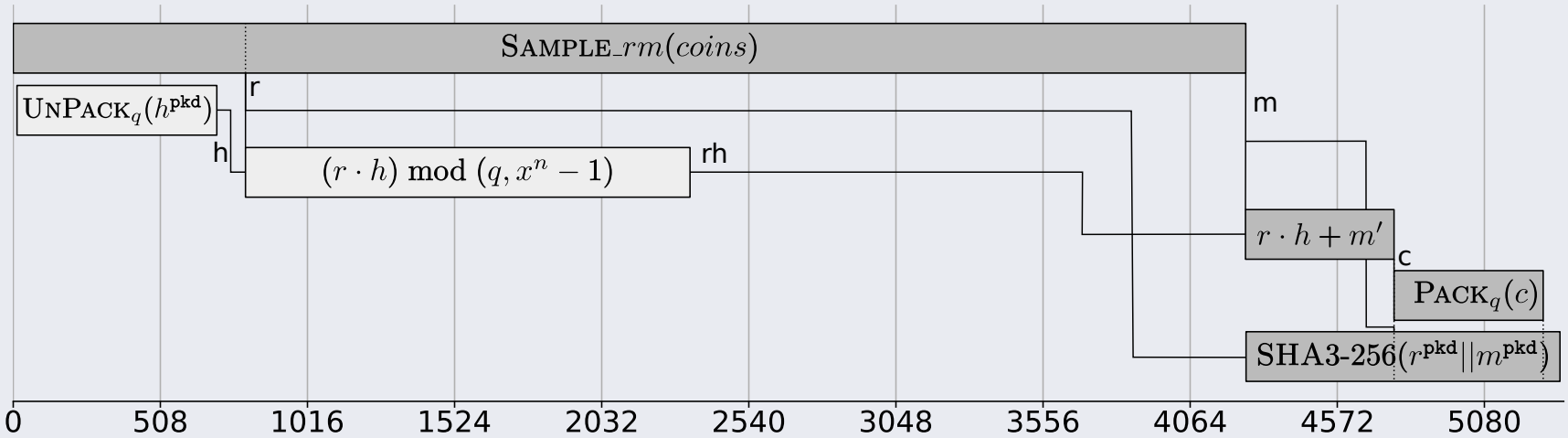
Encapsulation



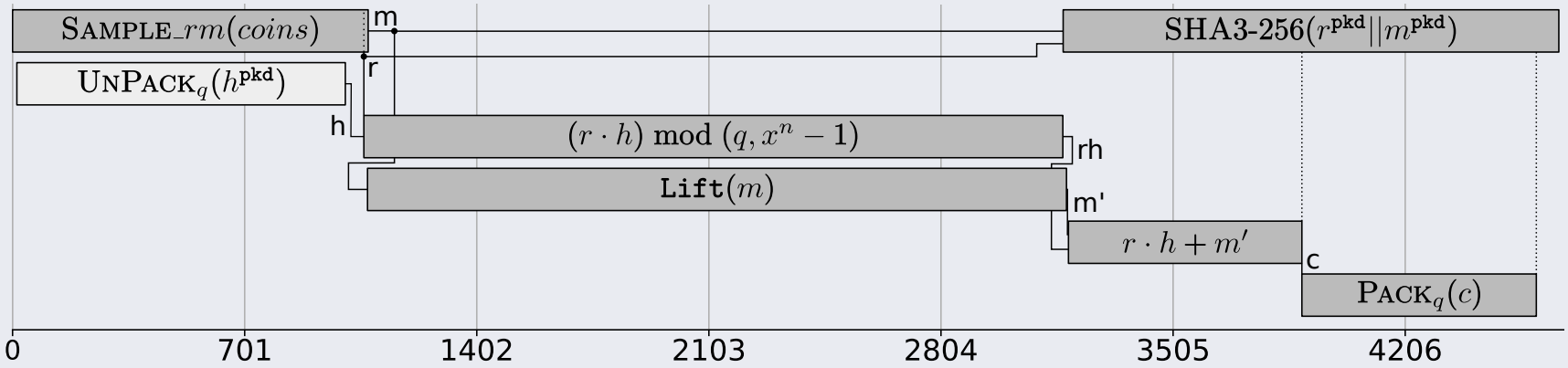
(a) NTRU HPS ($q = 2048, n = 509$)

Figure: Schedule of the NTRU KEM with a x-net multiplier (x axis represents the CC)

Encapsulation



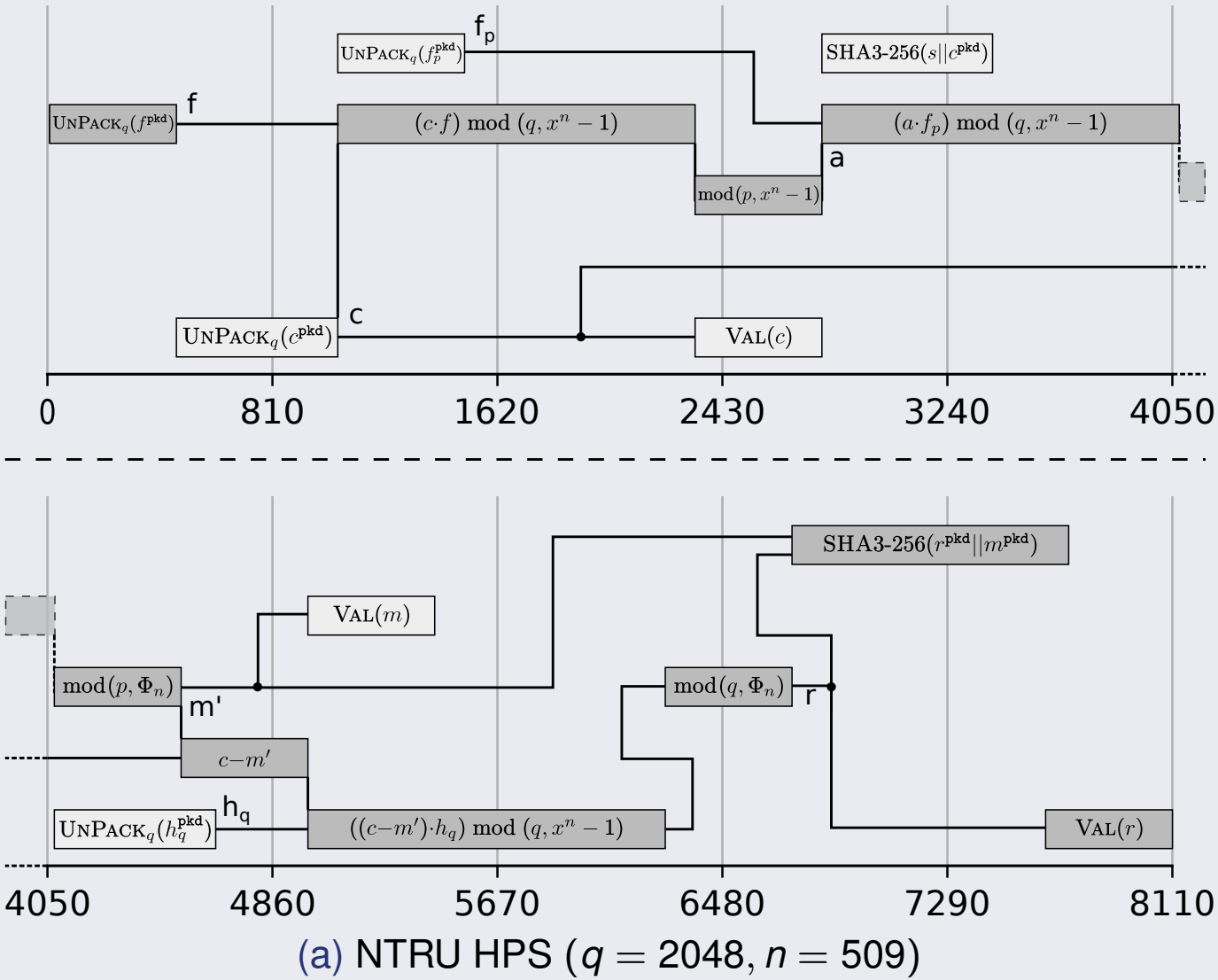
(a) NTRU HPS ($q = 2048, n = 509$)



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

Figure: Schedule of the NTRU KEM with a x-net multiplier (x axis represents the CC)

Decapsulation



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

Figure: Schedule of the NTRU HPS KEM with a x-net multiplier (x axis represents the CC)

Design Space Exploration results

Design space exploration on FPGA - parameters

We have conducted a Design Space Exploration for the encapsulation and decapsulation operations on a ZYNQ UltraScale+ FPGA to compare with the current state-of-the-art, separating the top-level modules of encap and decap

DSE parameters:

- all NTRU NIST parameter sets
- **x-net** and **Comba** polynomial multiplier algorithms
- variable-weight sampler based on **rejection** or **modulo** algorithms
- varying the memory access width per arithmetic unit component

Design space exploration on FPGA - NTRU KEM encapsulation

Table: **DSE of encapsulation** module for Security Level 3 (equiv. AES-192) NTRU HPS and HRSS. Parameters: multiplier architecture (Comba, *x*-net); transfer width (tw) for multiplier input operand (op1) and result (res), adder, session key generator (skg); polynomial sampler (Modulo, Rejection). Target frequencies reached: **400 MHz**, Area-Time product as latency (µs) $\times$ kSlice

$\mathcal{R}_p \times \mathcal{R}_q$ multiplier			sampler		add	skg	lift	NTRU	Latency		Area				AT
arch	op1 tw	res tw	alg	tw	tw	tw	tw	variant	CC	µs	LUT	FF	Slice	BRAM	prod.
x-net	1	1	M	2	1	4	/	hps2048677	6435	16.08	23171	13772	4122	4.0	66
x-net	1	1	R	2	1	4	/	hps2048677	6465	16.16	23117	13716	4051	4.0	65
x-net	4	2	M	2	2	4	/	hps2048677	6097	15.24	23357	13824	4219	5.0	64
x-net	4	2	R	2	2	4	/	hps2048677	6127	15.31	23543	13805	4471	5.0	68
x-net	8	4	M	4	4	4	/	hps2048677	5928	14.82	24664	13902	4509	8.5	66
x-net	8	4	R	4	4	4	/	hps2048677	5947	14.86	24562	13760	4456	8.5	66
x-net	8	4	M	4	8	4	/	hps2048677	5843	14.60	24817	13923	4378	12.5	63
x-net	8	4	R	4	8	4	/	hps2048677	5862	14.65	24387	13817	4116	12.5	60
Comba	1	1	R	1	1	1	/	hps2048677	462079	1155.20	8221	4828	1215	1.5	1403
x-net	1	1	M	2	1	4	1	hrss701	4542	11.35	26350	15653	4499	6.5	51
x-net	1	1	R	2	1	4	1	hrss701	4542	11.35	26255	15534	4465	6.5	50
x-net	4	2	M	2	2	4	2	hrss701	3317	8.29	27881	15690	4809	6.5	39
x-net	4	2	R	2	2	4	2	hrss701	3317	8.29	27731	15634	4885	6.5	40
x-net	8	4	M	4	4	4	4	hrss701	2879	7.19	28561	16005	4949	9.0	35
x-net	8	4	R	4	4	4	4	hrss701	2879	7.19	27898	15743	4708	9.0	33
x-net	8	4	M	4	8	4	4	hrss701	2791	6.97	28563	15934	4821	13.0	33
x-net	8	4	R	4	8	4	4	hrss701	2791	6.97	28041	15689	4605	13.0	32
Comba	1	1	R	1	1	1	1	hrss701	495312	1238.28	7978	4917	1324	2.0	1639

For reference, the area occupied by the Keccak-512 module included in the result figures is 5368 LUTs and 2713 FFs

Design space exploration on FPGA - NTRU KEM decapsulation

Table: **DSE of decapsulation** module for Security Level 3 (equiv. AES-192) NTRU HPS and HRSS.
Parameters: multiplier architecture (Comba, *x*-net); transfer width (tw) for multiplier input operand (op1) and result (res), adder, session key generator (skg), validator (val).
Target frequencies reached: **350 MHz** (*x*-net) and **400 MHz** (Comba).
Area-Time product computed as latency (µs) $\times$ kSlice

$\mathcal{R}_q \times \mathcal{R}_q$ multiplier			add	skg	val	lift	NTRU	Latency		Area					AT
arch	op1 tw	res tw	tw	tw	tw	tw	variant	CC	µs	LUT	FF	Slice	DSP	BRAM	prod.
x-net	1	1	1	1	1	/	hps2048677	10048	30.65	15430	20648	4691	701	2.5	143
x-net	2	2	2	2	2	/	hps2048677	7686	21.95	22689	20059	4977	701	3.5	109
x-net	4	4	4	4	4	/	hps2048677	6163	17.60	23689	21286	5426	701	6.0	95
Comba	1	1	1	1	1	/	hps2048677	1385714	3958.98	8360	4705	1193	1	2.5	4723
x-net	1	1	1	1	1	1	hrss701	13351	38.14	17882	24300	5682	701	2.5	216
x-net	2	2	2	2	2	2	hrss701	9514	27.18	27342	24522	6197	701	3.5	168
x-net	4	4	4	4	4	4	hrss701	7606	21.73	28139	25101	6476	701	6.0	140
Comba	1	1	1	1	1	1	hrss701	1487552	4249.93	8436	4813	1292	1	2.5	5490

For reference, the area occupied by the Keccak-512 module included in the result figures is 5368 LUTs and 2713 FFs

First ASIC implementation results using a 40 nm library

Table: **encapsulation** reached **750** and **700 MHz** for area constrained and fast designs, respectively

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 µs
x-net	hps2048509	0.66	2.76	39.12	1.12	2.64	90.40	1.41	-	140.19	6.2
	hps2048677	0.68	2.97	39.16	1.11	2.67	120.44	1.41	-	170.44	8.4
	hps4096821	0.73	2.95	39.28	0.82	2.68	161.21	1.07	-	211.05	10.2
	hrss701	0.83	1.36	40.24	1.26	2.67	148.91	1.54	1.87	201.15	4.1
Com.	hps2048509	0.39	2.92	41.56	1.14	1.63	1.22	1.39	-	51.52	349.2
	hps2048677	0.42	3.01	40.06	1.13	1.65	1.30	1.40	-	50.30	616.1
	hps4096821	0.44	3.00	40.29	0.82	1.67	1.31	1.07	-	49.91	904.7
	hrss701	0.47	2.40	40.96	1.35	1.68	1.36	1.52	1.22	52.43	660.4

Table: **decapsulation** reached **750** and **650 MHz** for area constrained and fast designs, respectively

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 µs
x-net	hps2048509	0.78	40.60	0.68	2.67	268.95	1.02	1.54	0.21	-	320.06	7.1
	hps2048677	0.81	40.10	0.72	2.70	359.13	1.06	1.52	0.26	-	410.03	9.4
	hps4096821	0.81	38.96	0.71	2.66	495.69	1.07	1.17	0.28	-	517.80	11.5
	hrss701	0.91	40.71	0.72	2.71	507.23	1.05	1.66	0.25	1.72	561.02	11.7
Com.	hps2048509	0.42	41.46	0.67	1.64	1.59	1.08	1.51	0.31	-	51.45	1047.1
	hps2048677	0.45	40.70	0.71	1.68	2.38	1.05	1.48	0.29	-	50.74	1847.6
	hps4096821	0.46	40.14	0.72	1.68	2.49	1.07	1.17	0.30	-	50.08	2713.5
	hrss701	0.50	40.33	0.71	1.69	2.66	1.09	1.63	0.31	1.14	52.26	1983.4

Comparison with the state-of-the-art

Table: Comparison of our best speed-oriented solution to [4] and [5] on a ZYNQ UltraScale+

NTRU KEM encapsulation

Sec. lvl.	NTRU Variant	Work	Freq	Area					Latency		AT
				LUT	FF	Slice	DSP	BRAM	CC	µs	prod.
1	hps2048509	Our	400	19379	11663	3585	0	8.5	4384	10.9	39
3	hps2048677	Our	400	24664	13902	4509	0	8.5	5928	14.8	66
		[4]	250	26325	17568	4638	0	5	3687	14.8	68
	hrss701	Our	400	28396	15894	4699	0	9.0	2879	7.2	33
		[4]	300	31494	25120	6652	0	2.5	2219	7.4	49
	sntrup761	[5]	289	31996	22425	5381	6	4.5	5007	17.3	93
5	hps4096821	Our	400	29637	16634	4978	0	9.0	7181	17.9	89
		[4]	250	33698	30551	7370	0	5.5	4576	18.3	134

NTRU KEM decapsulation

Sec. lvl.	NTRU Variant	Work	Freq	Area					Latency		AT
				LUT	FF	Slice	DSP	BRAM	CC	µs	prod.
1	hps2048509	Our	350	20051	17379	4472	509	5.5	4678	13.3	59
3	hps2048677	Our	350	23689	21286	5426	677	6.0	6163	17.6	95
		[4]	300	29935	19511	5217	45	2.5	7522	25.1	130
	hrss701	Our	350	27790	24979	6257	701	6.0	7606	21.7	135
		[4]	300	37702	34441	8032	45	2.5	8826	29.4	236
	sntrup761	[5]	285	32301	22724	5432	9	3.5	10989	38.6	209
5	hps4096821	Our	350	29074	26474	6808	821	6.0	7521	21.4	146
		[4]	300	38642	33003	7785	45	2.5	10211	34.0	264

Conclusions

Conclusions

- designed the first fully-fledged ASIC-oriented implementation of the NTRU cryptoscheme presented at NIST post-quantum cryptography contest
- the HDL description has been developed with the main goal to ease the flexibility of the design in order to perform DSE
  - reduced time to complete a design with new trade-offs coming from the update of any inner component
- the latency and Area$\times$Time products of our speed-oriented FPGA designs outperform current state-of-the-art solutions
- the figures of merit of our solutions compare quite favorably with optimized software solutions on μC and general-purpose CPUs

Thanks for your attention

Francesco Antognazza
PhD student - Politecnico di Milano
email: francesco.antognazza@polimi.it

References I

[1] Matthias J. Kannwischer et al. *PQM4: Post-quantum crypto library for the ARM Cortex-M4*. https://github.com/mupq/pqm4.

[2] VAMPIRE lab. *SUPERCOP: System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives*. 2022. URL: https://bench.cr.yp.to/results-kem.html.

[3] Andreas Hülsing et al. "High-Speed Key Encapsulation from NTRU". In: *CHES 2017*. Vol. 10529. LNCS. 2017.

[4] Viet Ba Dang, Kamyar Mohajerani, and Kris Gaj. "High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber". In: *IACR Cryptol. ePrint Arch.* (2021). URL: https://eprint.iacr.org/2021/1508.

[5] Bo-Yuan Peng et al. "Streamlined NTRU Prime on FPGA". In: *IACR Cryptol. ePrint Arch.* (2021). URL: https://eprint.iacr.org/2021/1444.

References II

[6] Zhenhui Qin et al. “A Compact Full Hardware Implementation of PQC Algorithm NTRU”. In: *2021 International Conference on Communications, Information System and Computer Engineering (CISCE)*. 2021, pp. 792–797.

Arithmetic in polynomial ring $\mathcal{R}_q = \mathbb{Z}_q / \langle x^n - 1 \rangle$ - DSE results

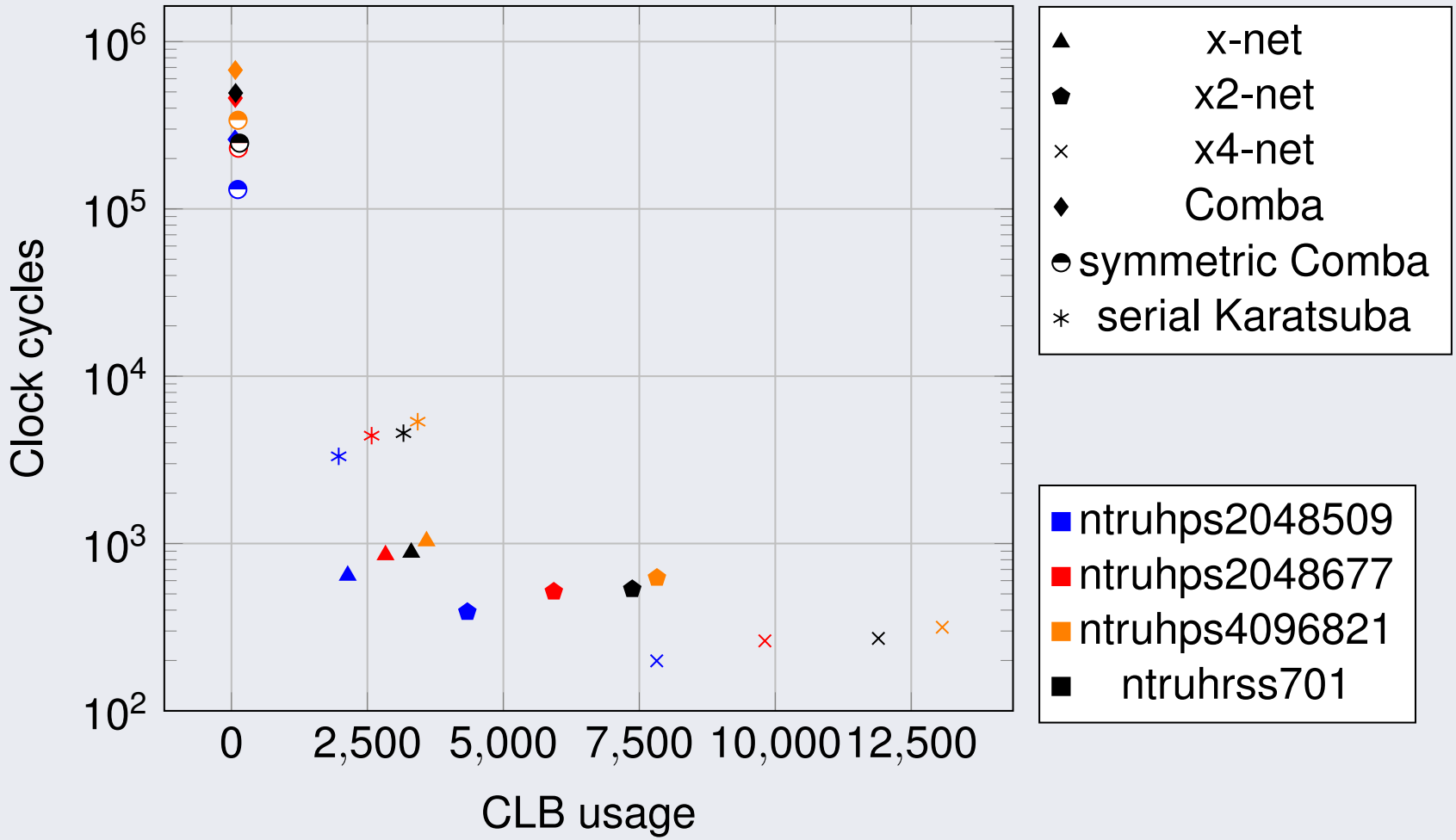


Figure: Time-Area chart comparing different polynomial multiplier architectures when implemented on a Xilinx UltraScale+ FPGA. For the x-net algorithm, we load 4 small coefficient each clock cycle.

Decrypted message validation and sanity checks (decaps)

Detects if a malformed ciphertext was received, protecting from attacks with an implicit rejection mechanism

Parallel checks performed

- α: checks for coefficients in $\{-1, 0, 1\}$
- β: counts the number of coefficients equal to 1
- γ: counts the number of coefficients equal to -1
- δ: accumulates the sum the first $n-1$ polynomial coefficients

$\mathbf{a} \in \mathcal{S}_p$	iif $\alpha == \top$
$\mathbf{a} \in \mathcal{S}_p \wedge \|\|\mathbf{a}\|\|$ valid	iif $(\alpha == \top) \wedge (\beta == w/2) \wedge (\gamma == w/2)$
$\mathbf{a} \bmod (q, \Phi_1) == 0$	iif $\delta == a_{n-1}$

More coefficients could be read out from memory each clock cycle to speed-up the validation

Comparison with the state-of-the-art

Table: Comparison of used algorithmms to perform multiplications, lifting of polynomials, sampling of coeffcients, and PRNG demands. Multiplier architectures: x-net (X), Comba (C), symmetric Comba (SC), serial Karatsuba (SK), Toom-Cook 3-way (TC3), odd-even Karatsuba (OEK)

Work	Scheme enc/dec	small-to-larg multiplier	larg-to-larg multiplier	variable-weight sampler	fixed-weight sampler	PRNG bit size	TRNG throughput	Lift algorithm
[4]	KEM	X	TC3 + OEK	modulo	merge-sort	31160	high	using SL multiplier
[6]	DPKE	X	X	n.a.	n.a.	n.a.	n.a.	n.a.
Our	KEM	X, C, SC, SK	X, C, SC, SK	rejection	Fisher-Yates	14482	12-14 bits/CC	multiplication-less
Our	KEM	X, C, SC, SK	X, C, SC, SK	modulo	Fisher-Yates	18860	18-26 bits/CC	multiplication-less

- TRNG bit size are calculated for NTRU HPS SL5 parameter set, which is the worst case scenario
- given the probabilistic nature of the Knuth and rejection sampling algorithms, we considered the worst-case scenario
- on average our throughput consumption is typically lower, and as low as 5 bits per clock cycle
- throughput for other works using inverted sorting is reasonably high due to requiring a block of 24600 bits of randomness immediately at the beginning of the algorithm