UNIVERSITY OF BIRMINGHAM

High-speed Instruction-set Coprocessor for Lattice-based Key Encapsulation Mechanisms: Saber in Hardware

Sujoy Sinha Roy and Andrea Basso

CHES 2020

Motivation

Saber is (now) a **round 3 finalist** for the NIST PQC standardization process.

NIST [MAA+20] reported that

> *"SABER is one of the most promising KEM schemes to be considered for standardization at the end of the third round."*

Saber's unique design choices

- Different implementation approaches from other lattice-based protocols
- Non-NTT based polynomial multipliers

The Saber protocol [DKSRV18]

Key Generation

$seed_A \leftarrow random()$

$\mathbf{A} = gen(seed_A)$

$\mathbf{s} \leftarrow small\_vec()$

$\mathbf{b} = \left\lfloor \frac{p}{q} \mathbf{A}^T \cdot \mathbf{s} \right\rceil$

$\xrightarrow{seed_A,\ \mathbf{b}}$

The Saber protocol [DKSRV18]

Key Generation

$$seed_A \leftarrow random()$$
$$\boldsymbol{A} = gen(seed_A)$$
$$\mathbf{s} \leftarrow small\_vec()$$
$$\boldsymbol{b} = \left\lfloor \frac{p}{q} \boldsymbol{A}^T \cdot \mathbf{s} \right\rceil$$

$seed_A$, $\mathbf{b}$ →

← $\mathbf{b}'$, c_m

Encryption

$$\begin{aligned} \boldsymbol{A} &= gen(seed_A) \\ \mathbf{s}' &\leftarrow small\_vec() \\ \boldsymbol{b}' &= \left\lfloor \frac{p}{q} \boldsymbol{A} \cdot \mathbf{s}' \right\rceil \\ c_m &= \left\lfloor \frac{T}{p} \boldsymbol{b}^T \mathbf{s}' + \frac{T}{2} m \right\rceil \end{aligned}$$

The Saber protocol [DKSRV18]

Key Generation

$$seed_A \leftarrow random()$$
$$\boldsymbol{A} = gen(seed_A)$$
$$\mathbf{s} \leftarrow small\_vec()$$
$$\boldsymbol{b} = \left\lfloor \frac{p}{q} \boldsymbol{A}^T \cdot \mathbf{s} \right\rceil$$

$seed_A$, $\mathbf{b}$ →

Encryption

$$\boldsymbol{A} = gen(seed_A)$$
$$\mathbf{s'} \leftarrow small\_vec()$$
$$\boldsymbol{b'} = \left\lfloor \frac{p}{q} \boldsymbol{A} \cdot \mathbf{s'} \right\rceil$$
$$c_m = \left\lfloor \frac{T}{p} \boldsymbol{b}^T \mathbf{s'} + \frac{T}{2} m \right\rceil$$

← $\mathbf{b'}$, c_m

Decryption

$$v = \boldsymbol{b'}^T \mathbf{s}$$
$$m = \left\lfloor \frac{2}{q} \left(v - \frac{p}{T} c_m\right) \right\rceil$$

The Saber protocol [DKSRV18]

Key Generation

$$seed_A \leftarrow random()$$
$$\boldsymbol{A} = gen(seed_A)$$
$$\mathbf{s} \leftarrow small\_vec()$$
$$\boldsymbol{b} = \left\lfloor \frac{p}{q} \mathbf{A}^T \cdot \mathbf{s} \right\rceil$$

$seed_A$, $\mathbf{b}$ →

Encryption

$$\boldsymbol{A} = gen(seed_A)$$
$$\mathbf{s'} \leftarrow small\_vec()$$
$$\boldsymbol{b'} = \left\lfloor \frac{p}{q} \mathbf{A} \cdot \mathbf{s'} \right\rceil$$
$$c_m = \left\lfloor \frac{T}{p} \mathbf{b}^T \mathbf{s'} + \frac{T}{2} m \right\rceil$$

← $\mathbf{b}'$, c_m

Decryption

$$v = \boldsymbol{b'}^T \mathbf{s}$$
$$m = \left\lfloor \frac{2}{q} \left(v - \frac{p}{T} c_m \right) \right\rceil$$

Key Encapsulation Mechanism

Saber.KEM is obtained via the Fujisaki-Okamoto (FO) transform.

Implementation-wise, the FO consists mainly of SHA/SHAKE calls.

Performance bottlenecks

The majority of computations involve

1. SHA/SHAKE

2. Computing polynomial multiplication

Performance bottlenecks

The majority of computations involve

1. SHA/SHAKE
   - 70/80% of computations in software
   - Keccak is very fast in hardware
   - High-speed implementation by the Keccak team
   - Serialized SHA(KE) in Saber $\longrightarrow$ one core
2. Computing polynomial multiplication

Performance bottlenecks

The majority of computations involve

1. SHA/SHAKE
   - 70/80% of computations in software
   - Keccak is very fast in hardware
   - High-speed implementation by the Keccak team
   - Serialized SHA(KE) in Saber $\longrightarrow$ one core
2. Computing polynomial multiplication

Performance bottlenecks

The majority of computations involve

1. SHA/SHAKE
   - 70/80% of computations in software
   - Keccak is very fast in hardware
   - High-speed implementation by the Keccak team
   - Serialized SHA(KE) in Saber ⟶ one core
2. Computing polynomial multiplication
   - **The main focus of this work**

Polynomial multiplication in Saber

The main characteristics

- Module-LWR
  - Different module ranks for different security levels
  - All polynomials have degree 255
- Small secrets
  - Secret polynomial coefficients in $[-3, 3]$, $[-4, 4]$ or $[-5, 5]$
- Power-of-2 moduli
  - Multiplication modulo 2^{13} or 2^{10}
  - Free modular reduction
  - **No NTT**

Our polynomial multiplication approach

The alternatives to NTT

The Number Theoretic Transform (NTT) requires the modulus to be prime

In software: improved Toom-Cook ([BMKV20], also at CHES 2020)

In hardware:

- Toom-Cook/Karatsuba not convenient because recursive
- High parallelism
- Ad-hoc solutions

Our polynomial multiplication approach

The alternatives to NTT

The Number Theoretic Transform (NTT) requires the modulus to be prime

In software: improved Toom-Cook ([BMKV20], also at CHES 2020)

In hardware:

- Toom-Cook/Karatsuba not convenient because recursive
- High parallelism
- Ad-hoc solutions

$\Rightarrow$ **Schoolbook algorithm**

The schoolbook algorithm

The alternatives to NTT

Algorithm: Schoolbook algorithm

$acc(x) \leftarrow 0$

for $i = 0; i < 256; i{+}{+}$ **do**

  for $j = 0; j < 256; j{+}{+}$ **do**

    $acc[j] = acc[j] + b[j] \cdot a[i]$

  $b = b \cdot x \bmod \langle x^{256} + 1 \rangle$

return acc

The schoolbook algorithm

The alternatives to NTT

Algorithm: Schoolbook algorithm

$acc(x) \leftarrow 0$

for $i = 0; i < 256; i{+}{+}$ **do**

  for $j = 0; j < 256; j{+}{+}$ **do**

    $acc[j] = acc[j] + b[j] \cdot a[i]$

  $b = b \cdot x \bmod \langle x^{256} + 1 \rangle$ ← negacyclic shift

return acc

The schoolbook algorithm

The alternatives to NTT

Algorithm: Schoolbook algorithm

$acc(x) \leftarrow 0$

for $i = 0; i < 256; i{+}{+}$ **do**

  for $j = 0; j < 256; j{+}{+}$ **do**

    $acc[j] = acc[j] + b[j] \cdot a[i]$

  $b = b \cdot x \bmod \langle x^{256} + 1 \rangle$ ← negacyclic shift

return *acc*

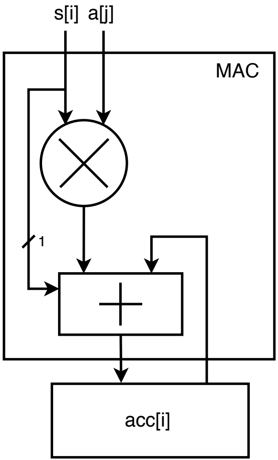
Advantages

- Simple implementation
- High flexibility
- Great performance

Multiply and ACcumulate (MAC) units

How to compute coefficient-wise operations

- Small secrets ⟶ bitshift & add multiplication
- Power-of-two moduli ⟶ no modular reduction



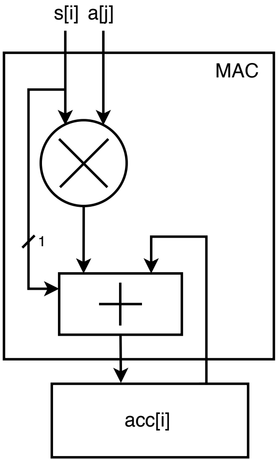
Multiply and ACcumulate (MAC) units

How to compute coefficient-wise operations

- Small secrets ⟶ bitshift & add multiplication
- Power-of-two moduli ⟶ no modular reduction

$$\Downarrow$$

A MAC unit requires little area (50 LUTs)



Multiply and ACcumulate (MAC) units

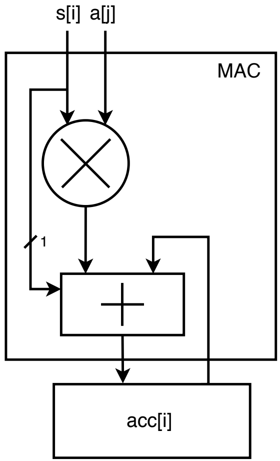
How to compute coefficient-wise operations

- Small secrets ⟶ bitshift & add multiplication
- Power-of-two moduli ⟶ no modular reduction

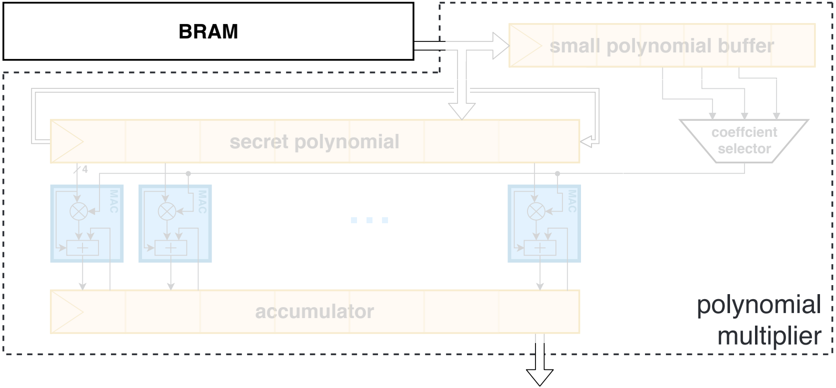
⇓

A MAC unit requires little area (50 LUTs)

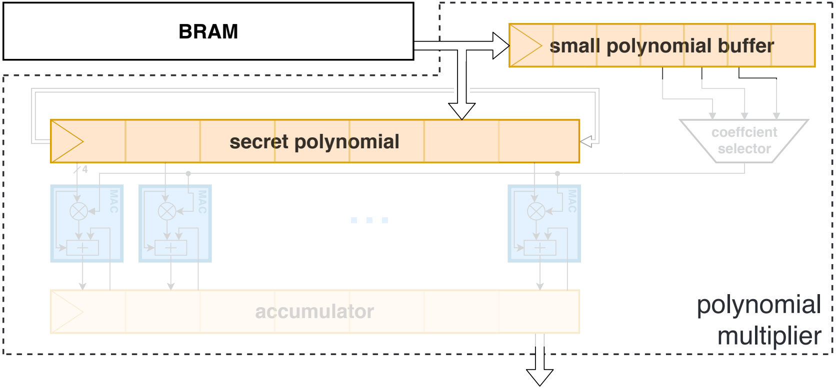
We use 256 MACs in parallel



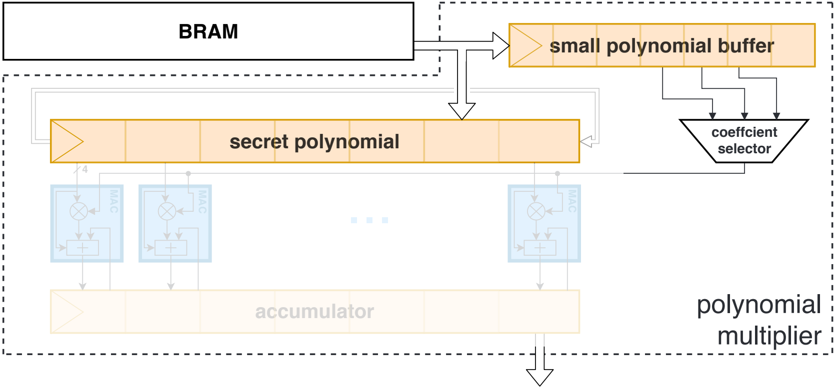
The polynomial multiplier



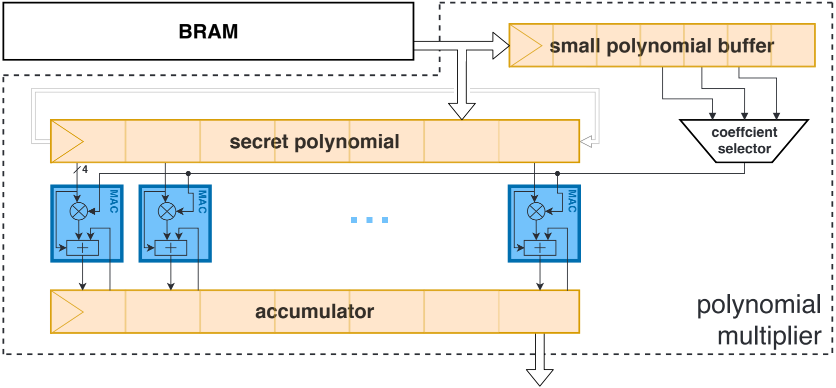
The polynomial multiplier



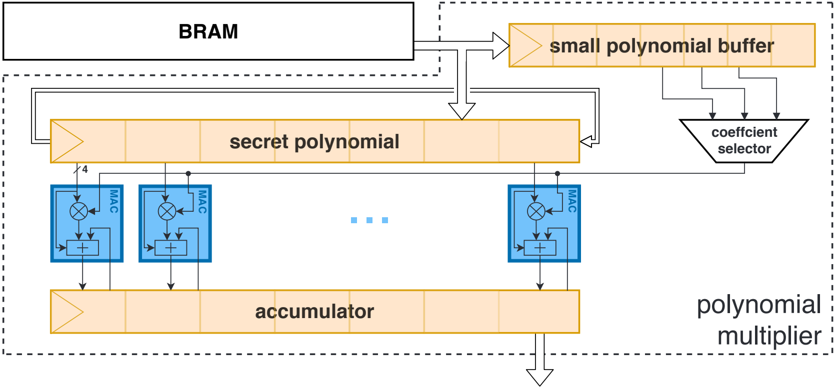
The polynomial multiplier



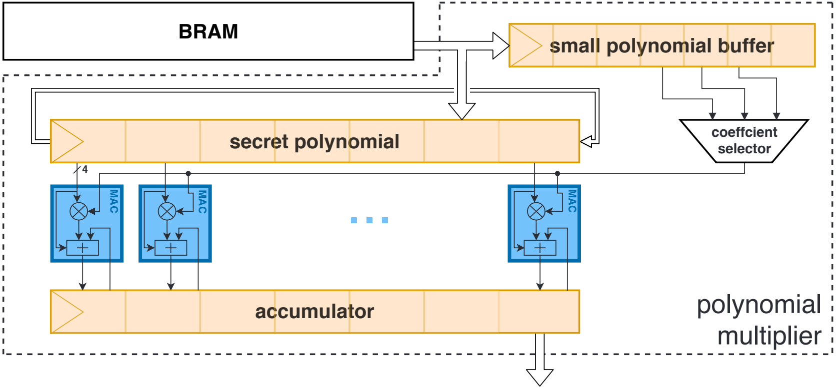
The polynomial multiplier



The polynomial multiplier



The polynomial multiplier



Performance

A full polynomial multiplication can be computed in 256 cycles!

The full architecture

An instruction-set coprocessor architecture

Advantages

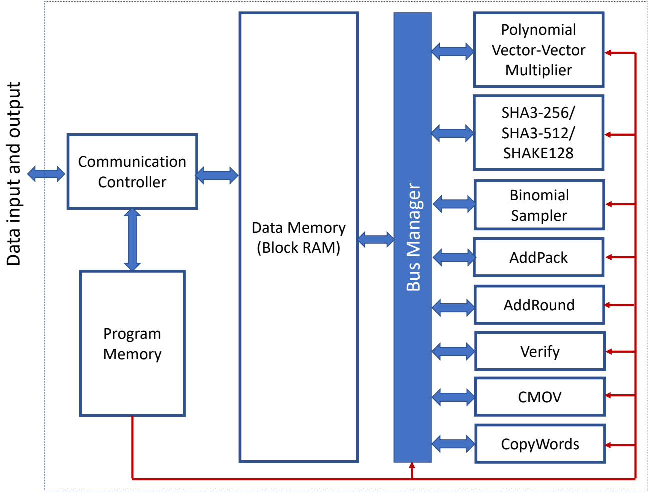
- Modularity

  ⇓
- Generic framework

  ⇓
- Other protocols
- Programmability

Disadvantages

- No parallelism



Design extendability

Unified architecture

- LightSaber
- Saber
- FireSaber

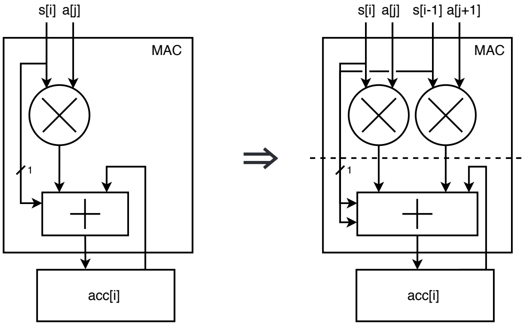
Design extendability

Unified architecture

- LightSaber
- Saber
- FireSaber

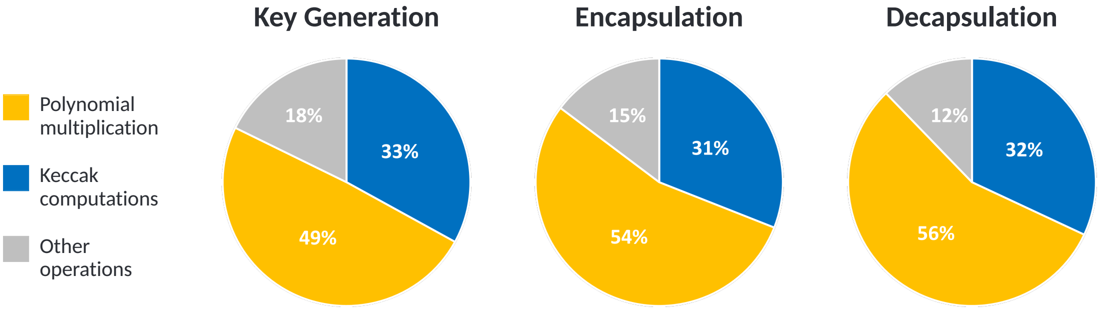
Performance/area trade-offs

- 512 multipliers
- ∼20% improvement in speed



Performance Results

Running on a Ultrascale+ XCZU9EG-2FFVB1156 FPGA

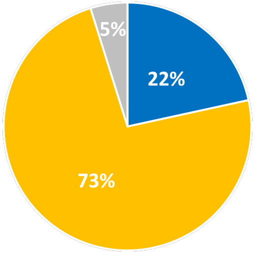


	Key Generation	Encapsulation	Decapsulation
Total cycles	5,453	6,618	8,034
Total time	**21.8 μs**	**26.5 μs**	**32.1 μs**
Throughput	45,872 op/s	37,776 op/s	31,118 op/s

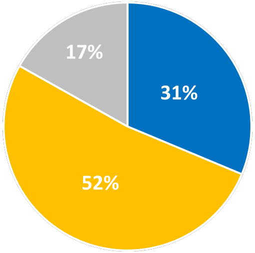
Area Results

Running on a Ultrascale+ XCZU9EG-2FFVB1156 FPGA

LUTs



Flip flops



DSPs

BRAM Tiles

	LUTs	Flip flops	DSPs	BRAM Tiles
Total	23,686	9,805	0	2
%	**8.6 %**	**1.8 %**	**0 %**	**0.2 %**

It is possible to fit 11 coprocessors, achieving a throughput of 504k / 416k / 342k op/s

Comparisons to other work

Implementation	Platform	Time in μs			Frequency	Area			
		Key	Encps	Decps	(MHz)	LUT	FF	DSP	BRAM
Kyber [DFA+20]	Virtex-7	-	17.1	23.3	245	14k	11k	8	14
NewHope [ZYC+20]	Artix-7	40	62.5	24	200	6.8k	4.4k	2	8
FrodoKEM [HOKG18]	Artix-7	45K	45K	47K	167	7.7K	3.5K	1	24
SIKE [MLRB20]	Virtex-7*	8K	14K	15K	142	21K	14K	162	38
Saber [BMTK+20]	Artix-7*	3K	4K	3K	125	7.4K	7.3K	28	2
Saber [DFAG19]	UltraScale+*	-	60	65	322	13K	12K	256	4
Saber [this work]	UltraScale+	**21.8**	**26.5**	**32.1**	250	**24K**	**10K**	**0**	**2**

*: HW/SW codesign

Future work

Other protocols

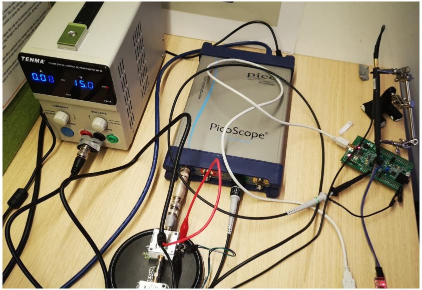
- Kyber and other lattice-based schemes
- Signature schemes?

Lightweight implementation

- Fewer multipliers

Side-channel resistance

- Masked implementation
- Handle small coefficients



Conclusion

A complete hardware architecture for Saber

- All three security levels: LightSaber, Saber and FireSaber
- Very high performance
- Still flexibile and with moderate area consumption

All code is available at `https://github.com/sujoyetc/SABER_HW`

Beyond Saber

- Generic framework for other protocols
- **High performance from non-NTT multiplier**

References I

[BMKV20] Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede.
Time-memory trade-off in Toom-Cook multiplication: an Application to Module-lattice based Cryptography.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(2):222–244, Mar. 2020.

[BMTK+20] Jose Maria Bermudo Mera, Furkan Turan, Angshuman Karmakar, Sujoy Sinha Roy, and Ingrid Verbauwhede.
Compact Domain-specific Co-processor for Accelerating Module Lattice-based Key Encapsulation Mechanism.
Accepted in DAC, 2020:321, 2020.

References II

[DFA+20] Viet Ba Dang, Farnoud Farahmand, Michal Andrzejczak, Kamyar Mohajerani, Duc Tri Nguyen, and Kris Gaj.
Implementation and benchmarking of round 2 candidates in the nist post-quantum cryptography standardization process using hardware and software/hardware co-design approaches.
Cryptology ePrint Archive, Report 2020/795, 2020.
https://eprint.iacr.org/2020/795.

[DFAG19] Viet B. Dang, Farnoud Farahmand, Michal Andrzejczak, and Kris Gaj.
Implementing and Benchmarking Three Lattice-Based Post-Quantum Cryptography Algorithms Using Software/Hardware Codesign.
In *International Conference on Field-Programmable Technology, FPT 2019, Tianjin, China, December 9-13, 2019*, pages 206–214. IEEE, 2019.

References III

[DKSRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren.
Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM, volume 10831, page 282–305.
Springer International Publishing, 2018.

[HOKG18] James Howe, Tobias Oder, Markus Krausz, and Tim Güneysu.
Standard Lattice-Based Key Encapsulation on Embedded Devices.
IACR Trans. Cryptogr. Hardw. Embed. Syst., 2018(3):372–393, 2018.

[MAA+20] Dustin Moody, Gorjan Alagic, Daniel C Apon, David A Cooper, Quynh H Dang, John M Kelsey, Yi-Kai Liu, Carl A Miller, Rene C Peralta, Ray A Perlner, et al.
Status report on the second round of the nist post-quantum cryptography standardization process.
NISTIR 8309, July 2020.

References IV

[MLRB20] Pedro Maat C. Massolino, Patrick Longa, Joost Renes, and Lejla Batina.
A Compact and Scalable Hardware/Software Co-design of SIKE.
IACR Trans. Cryptogr. Hardw. Embed. Syst., 2020(2):245–271, 2020.

[ZYC+20] Neng Zhang, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei, and Leibo Liu.
Highly efficient architecture of newhope-nist on fpga using low-complexity ntt/intt.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(2):49–72, Mar. 2020.