CHES 2020

Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT

Neng Zhang, Bohan Yang, Chen Chen,
Shouyi Yin, Shaojun Wei and Leibo Liu

Institute of Microelectronics, Tsinghua University, Beijing, China

Outline

- ☐ 1. Introduction
- ☐ 2. Low-Complexity NTT/INTT
- ☐ 3. Hardware Architecture
- ☐ 4. Implementation Results

1 Introduction

- **NewHope: a PQC algorithm for key encapsulation mechanism (KEM)**

NewHope-USENIX

NewHope-Simple

NewHope-NIST

- **A candidate in the 2nd round of NIST PQC standardization process, but not in the 3rd round**

- **Low-complexity NTT/INTT can be utilized by other algorithms.**

PQC: Crystals-Dilithium, qTesla, Falcon

FHE: LTV, BFV

1 Introduction

□ Main mathematical objects of NewHope

polynomials over the ring $\mathbb{R}_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle$			
q	12289	$\boldsymbol{\omega_N}$	Primitive N-th root of unit over Z_q
N	1024 or 512	$\boldsymbol{\gamma_{2N}}$	Square root of ω_N

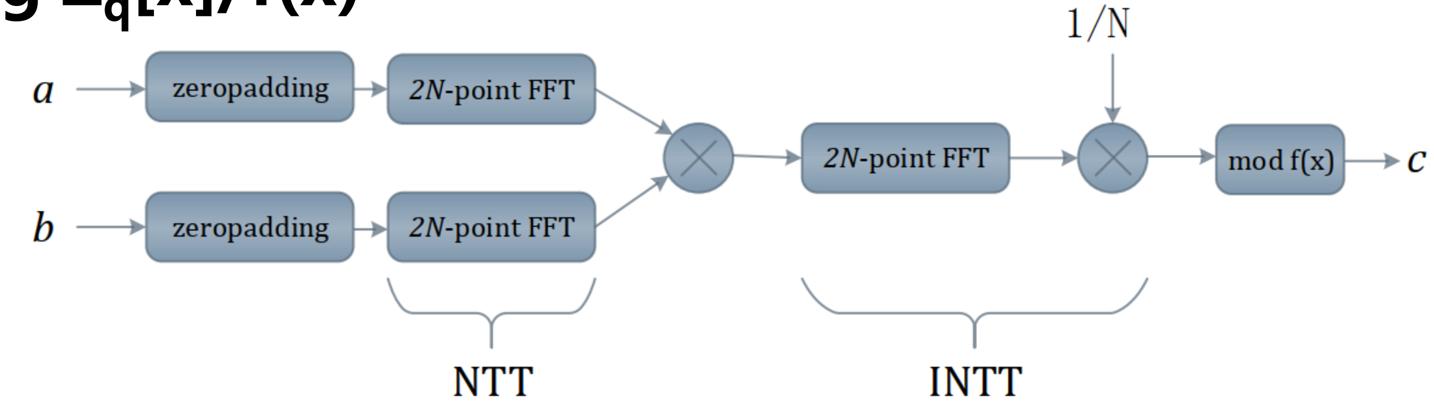
□ Encryption-based KEM

Key Generation	2 NTTs
Encryption	2 NTTs, 1 INTT
Decryption	1 INTT

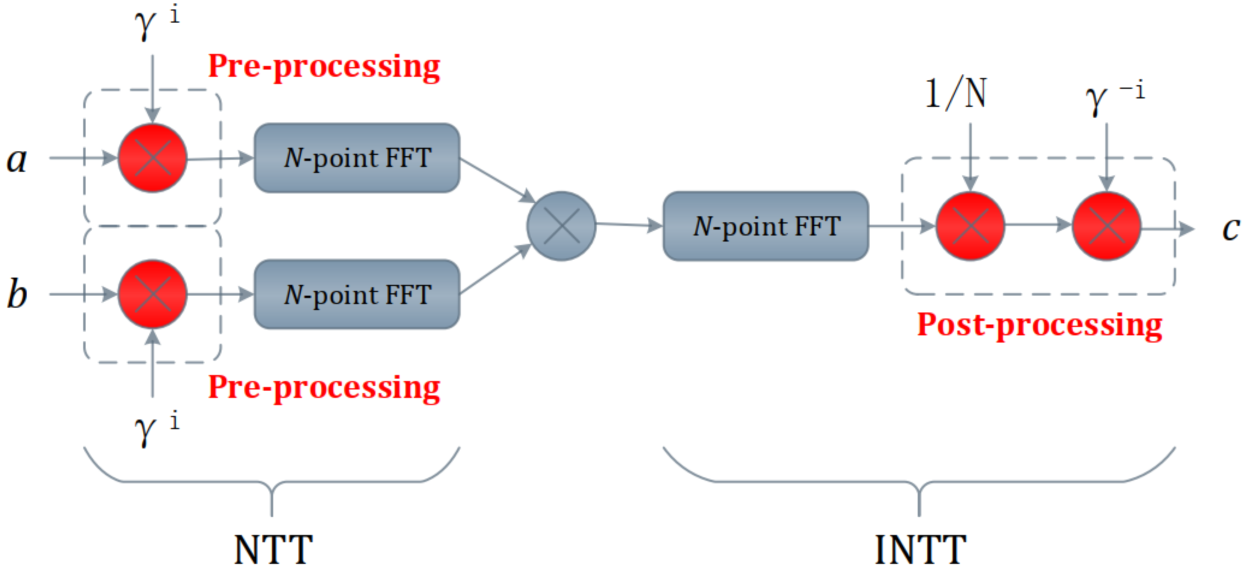
1 Introduction

□Multiplication over the ring $Z_q[x]/f(x)$

- f(x) is arbitrary
  - Convolution theory
  - q≡1 $(mod\ N)$

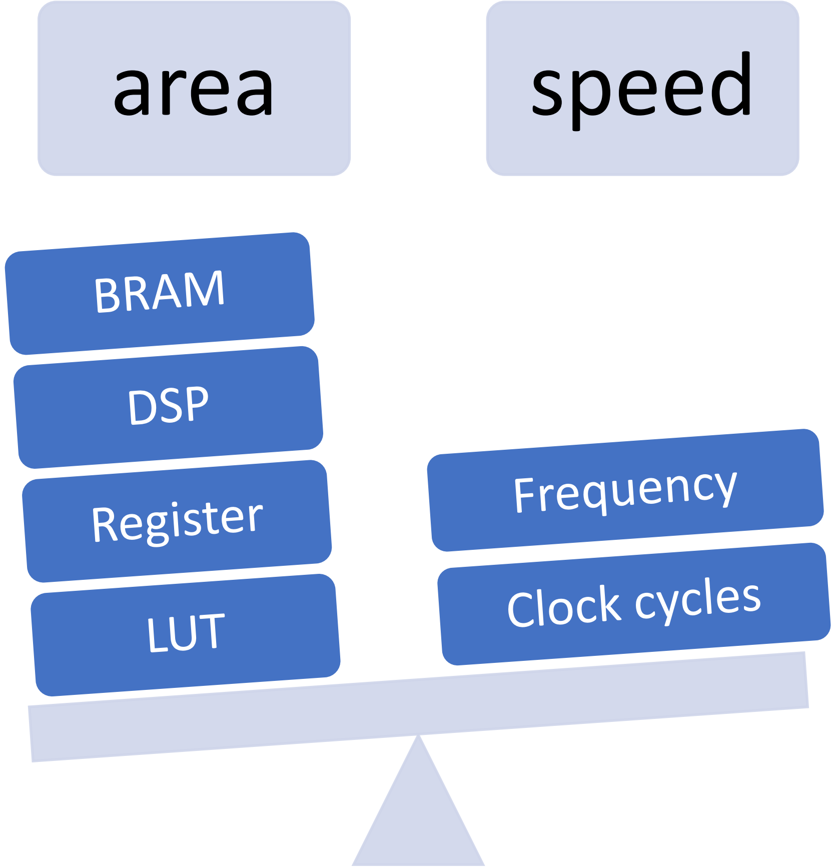


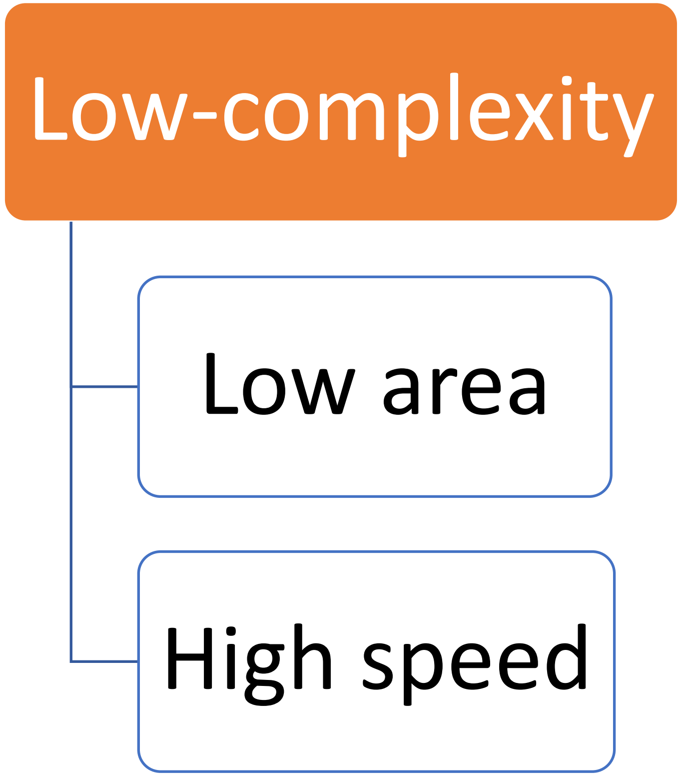
- f(x) = x^N+1
  - Negative Wrapped Convolution (NWC)
  - q≡1 $(mod\ 2N)$



1 Introduction

☐ Why do we need low-complexity ?





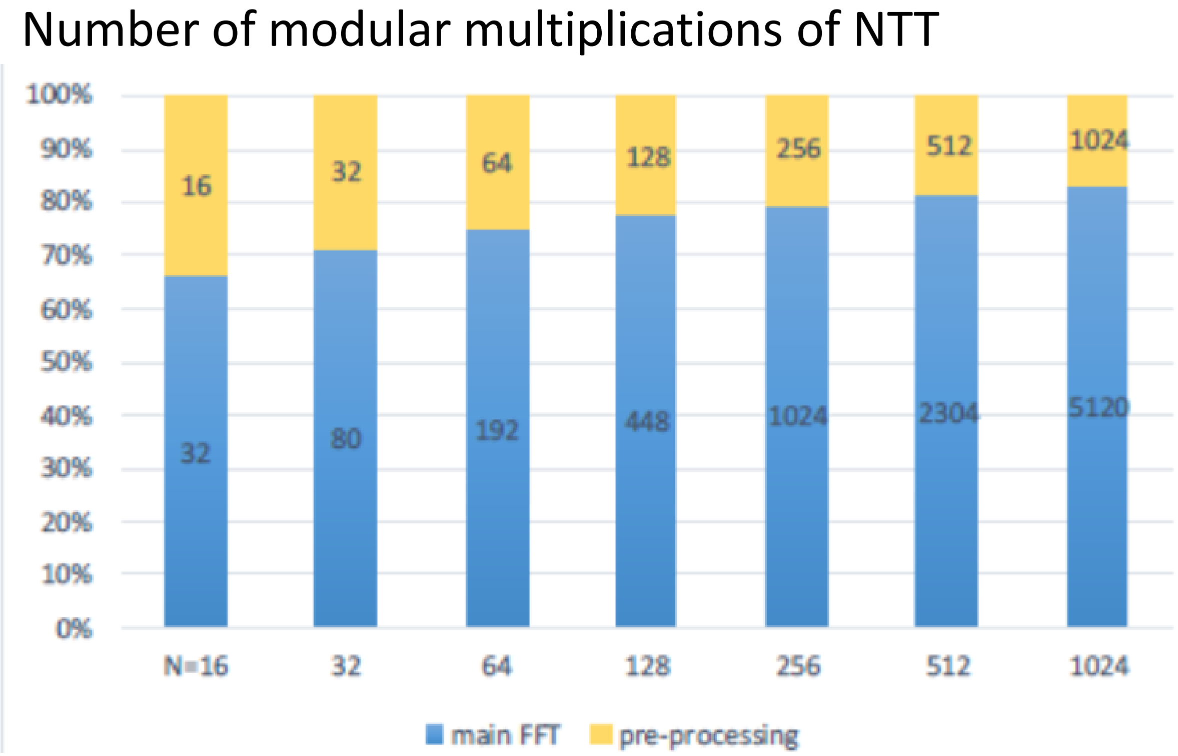
2.1 Low-Complexity NTT

- **Cost of the pre-processing is considerable**

$$(N/2)\log N + N$$

FFT

pre-processing


Number of modular multiplications of NTT

- **Low-Complexity NTT**
  - ➢A low-complexity NTT with twiddle factors computed on-the-fly [1].
  - ➢Merge the pre-processing into the DIT FFT with twiddle factors pre-computed.

[1] S. Roy, et al., Compact ring-lwe cryptoprocessor. CHES 2014

2.1 Low-Complexity NTT

◻ Derivation of the low-complexity NTT

- Inspired by the strategy of the Cooley-Turkey FFT
- Follow the divide-and-conquer method of FFT that divides in time domain (DIT)

- First, the pre-processing and the FFT are written together as a summation of N items

$$\hat{a}_i = \sum_{j=0}^{N-1} a_j \gamma_{2N}^{j} \omega_N^{ij} \bmod q$$

- Second, the summation is split into two groups according to parity of the index of *a*

$$\hat{a}_i = \sum_{j=0}^{N/2-1} a_{2j} \omega_N^{2ij} \gamma_{2N}^{2j} + \sum_{j=0}^{N/2-1} a_{2j+1} \omega_N^{i(2j+1)} \gamma_{2N}^{2j+1} \bmod q$$

2.1 Low-Complexity NTT

☐ Derivation of the low-complexity NTT

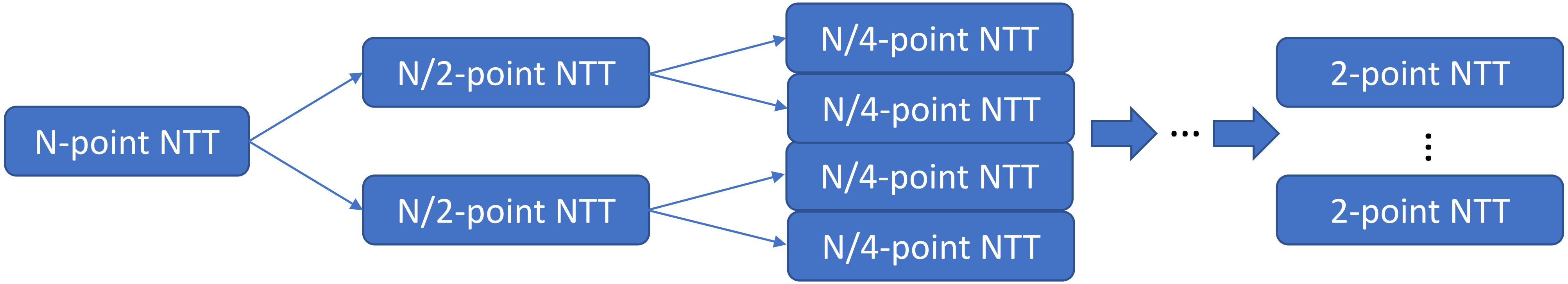
- Third, the equation is grouped into two parts according to the size of index i.

$$\hat{a}_i^{(0)} = \sum_{j=0}^{N/2-1} a_{2j}\,\omega_{N/2}^{ij}\gamma_N^{j} \bmod q,\ \hat{a}_i^{(1)} = \sum_{j=0}^{N/2-1} a_{2j+1}\,\omega_{N/2}^{ij}\gamma_N^{j} \bmod q.$$

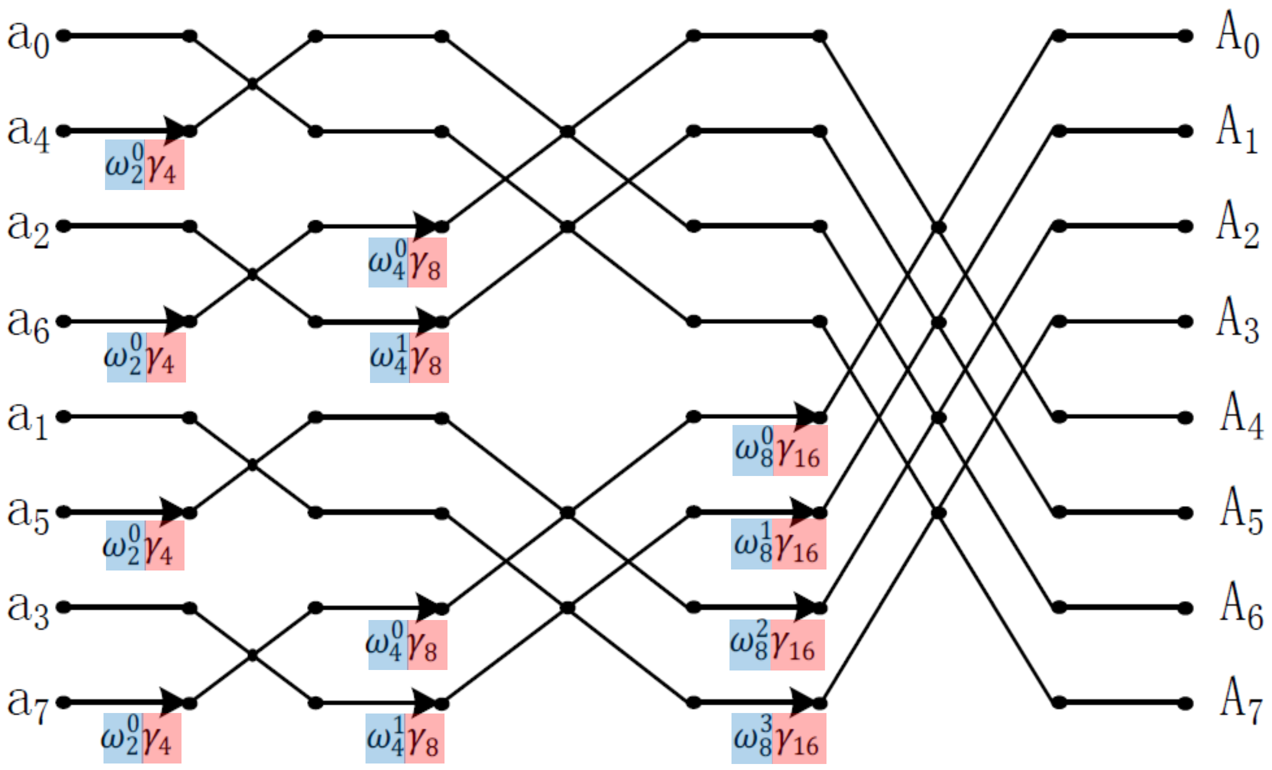
$$\begin{aligned} \hat{a}_i &= \hat{a}_i^{(0)} + \omega_N^{i}\gamma_{2N}\hat{a}_i^{(1)} \bmod q \\ \hat{a}_{i+N/2} &= \hat{a}_i^{(0)} - \omega_N^{i}\gamma_{2N}\hat{a}_i^{(1)} \bmod q \end{aligned}$$

$\hat{a}_i^{(0)}$ and $\hat{a}_i^{(1)}$ are N/2-point NTTs of a_{2j} and a_{2j+1}

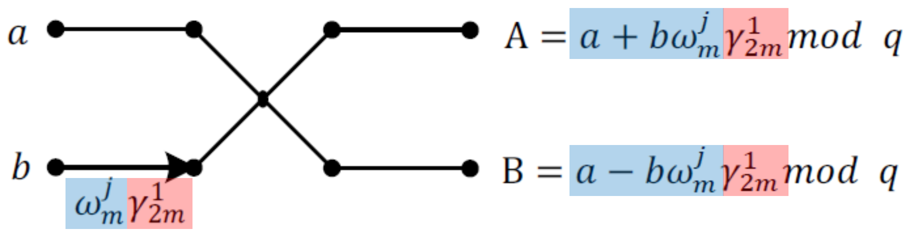
- In this way, N-point NTT can be resolved with two N/2-point NTTs



2.1 Low-Complexity NTT



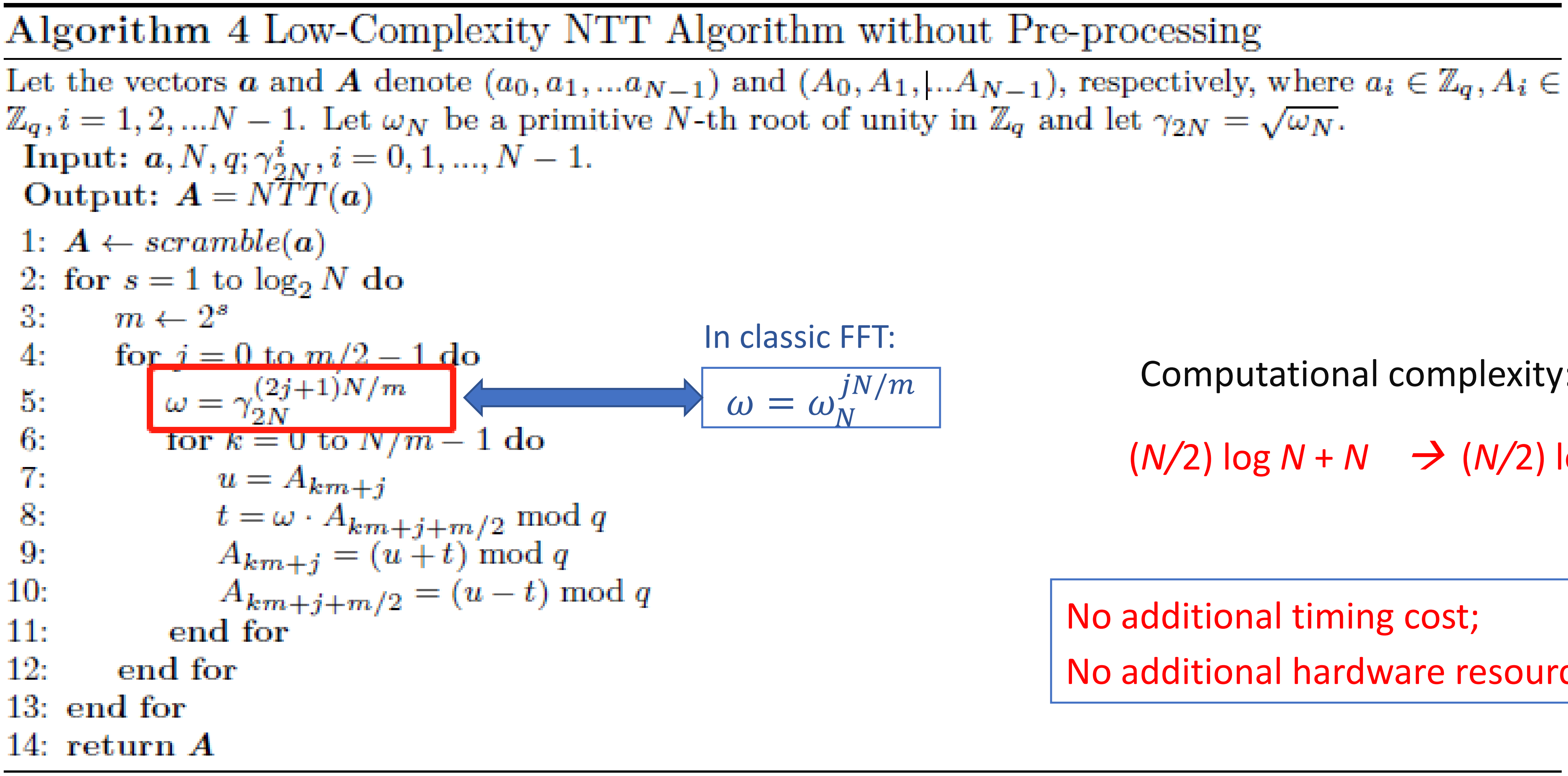
Dataflow of a 8-point low-complexity NTT



Butterfly of low-complexity NTT

$$\begin{aligned}\omega_m^j\gamma_{2m} &\equiv \gamma_{2m}^{2j+1} \\ &\equiv \gamma_{2N}^{(2j+1)N/m} \pmod{q}\end{aligned}$$

2.1 Low-Complexity NTT



Algorithm 4 Low-Complexity NTT Algorithm without Pre-processing

Let the vectors $\boldsymbol{a}$ and $\boldsymbol{A}$ denote $(a_0, a_1, \ldots a_{N-1})$ and $(A_0, A_1, \ldots A_{N-1})$, respectively, where $a_i \in \mathbb{Z}_q, A_i \in \mathbb{Z}_q, i = 1, 2, \ldots N-1$. Let ω_N be a primitive N-th root of unity in $\mathbb{Z}_q$ and let $\gamma_{2N} = \sqrt{\omega_N}$.

Input: $\boldsymbol{a}, N, q; \gamma_{2N}^i, i = 0, 1, \ldots, N-1$.

Output: $\boldsymbol{A} = NTT(\boldsymbol{a})$

```
$\boldsymbol{A} \leftarrow scramble(\boldsymbol{a})$
for $s = 1$ to $\log_2 N$ do
    $m \leftarrow 2^s$
    for $j = 0$ to $m/2 - 1$ do
        $\omega = \gamma_{2N}^{(2j+1)N/m}$
        for $k = 0$ to $N/m - 1$ do
            $u = A_{km+j}$
            $t = \omega \cdot A_{km+j+m/2} \bmod q$
            $A_{km+j} = (u + t) \bmod q$
            $A_{km+j+m/2} = (u - t) \bmod q$
        end for
    end for
end for
return $\boldsymbol{A}$
```

In classic FFT:

$$\omega = \omega_N^{jN/m}$$

Computational complexity:

$(N/2)\log N + N$ → $(N/2)\log N$

No additional timing cost;

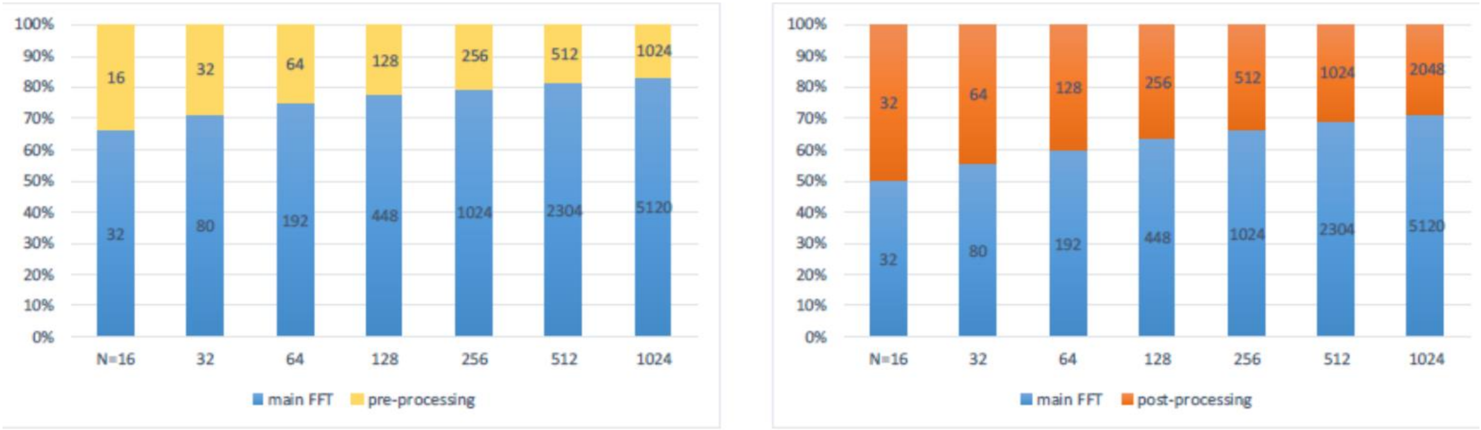
No additional hardware resources cost

2.2 Low-Complexity INTT

□ Cost of the post-processing is greater than pre-processing

$$\underbrace{(N/2)\log N}_{\text{FFT}} + \underbrace{2N}_{\text{post-processing}}$$

Number of modular multiplications of NTT and INTT



(a) NTT (b) INTT

□ Low-Complexity INTT

- ➢[1] merges the scaling of λ_{2N}^{-i} into the FFT.
- ➢Further merge the scaling of N^{-1} into the FFT

[1] T. Pöppelmann, et al., High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. LATINCRYPT 2015

2.2 Low-Complexity INTT

☐ Derivation of the low-complexity INTT

- ➢Inspired by the strategy of the Gentleman-Sande FFT
- ➢Follow the divide-and-conquer method of FFT that divides in frequency domain (DIF)

- ➢First, the post-processing and the FFT are written together as a summation of N items

$$a_i = N^{-1}\gamma_{2N}^{-i}\sum_{j=0}^{N-1}\hat{a}_j\omega_N^{-ij} \bmod q$$

- ➢Second, the summation is split into two groups according to the size of index of $\hat{a}$

$$a_i = N^{-1}\gamma_{2N}^{-i}\left(\sum_{j=0}^{N/2-1}\hat{a}_j\omega_N^{-ij} + \sum_{j=N/2}^{N-1}\hat{a}_j\omega_N^{-ij}\right) \bmod q$$

2.2 Low-Complexity INTT

☐ Derivation of the low-complexity INTT

➢Third, the equation is grouped into two parts according to the parity of i.

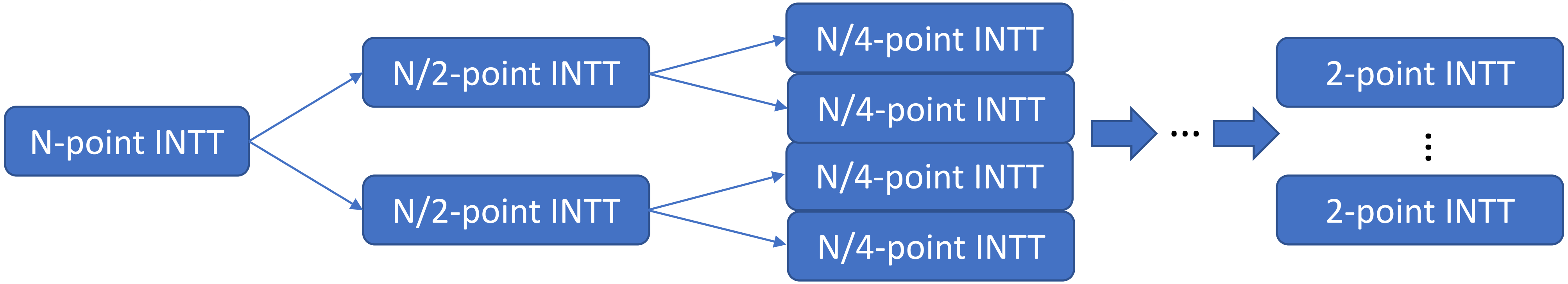
$$\hat{b}_j^{(0)} = \frac{\hat{a}_j + \hat{a}_{(j+N/2)}}{2} \bmod q, \hat{b}_j^{(1)} = \left[\frac{\hat{a}_j - \hat{a}_{(j+N/2)}}{2}\right] \omega_N^{-j}\gamma_{2N}^{-1} \bmod q$$

$$a_{2i} = (\frac{N}{2})^{-1}\gamma_N^{-i}\sum_{j=0}^{N/2-1}\hat{b}_j^{(0)}\omega_{N/2}^{-ij} \bmod q$$

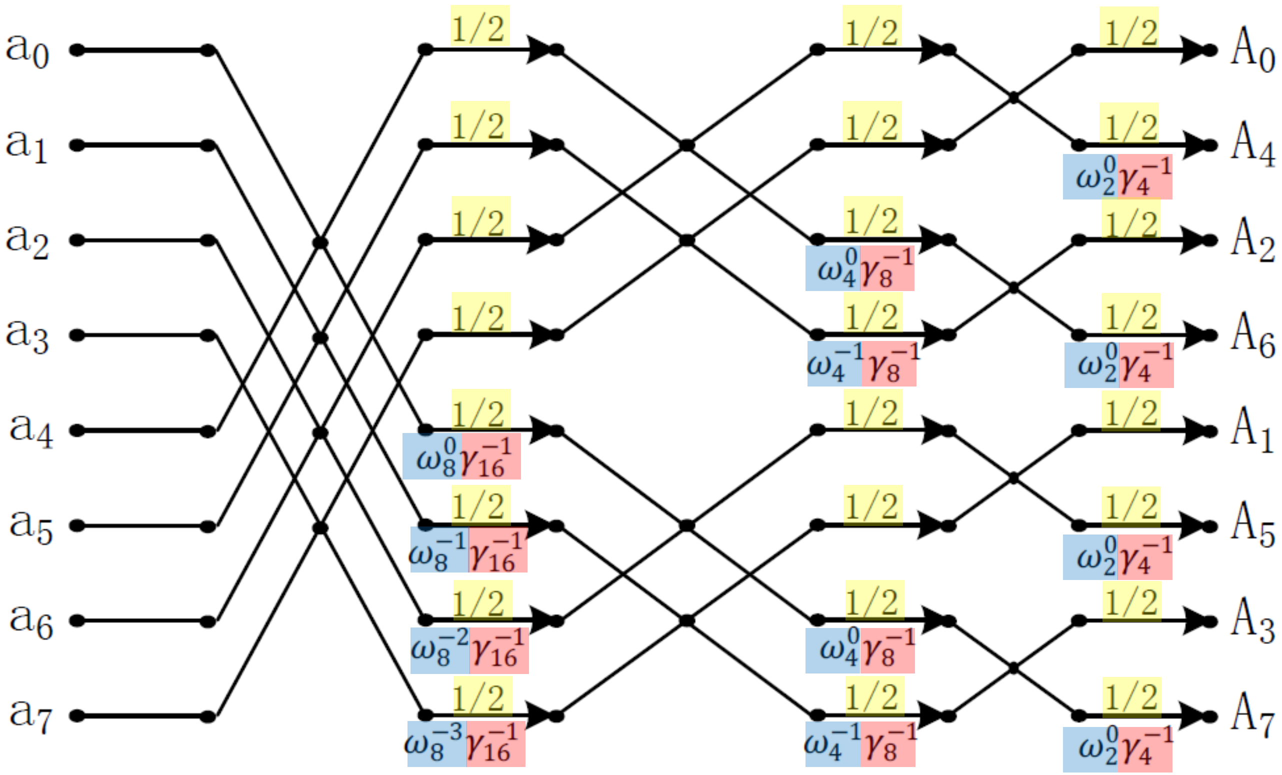
$$a_{2i+1} = (\frac{N}{2})^{-1}\gamma_N^{-i}\sum_{j=0}^{N/2-1}\hat{b}_j^{(1)}\omega_{N/2}^{-ij} \bmod q$$

a_{2i} and a_{2i+1} correspond to N/2-point INTT of $\hat{b}_i^{(0)}$ and $\hat{b}_i^{(1)}$

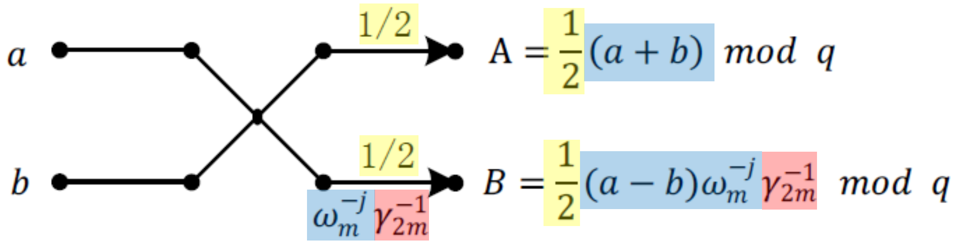
➢In this way, N-point INTT can be resolved with two N/2-point INTTs



2.2 Low-Complexity INTT



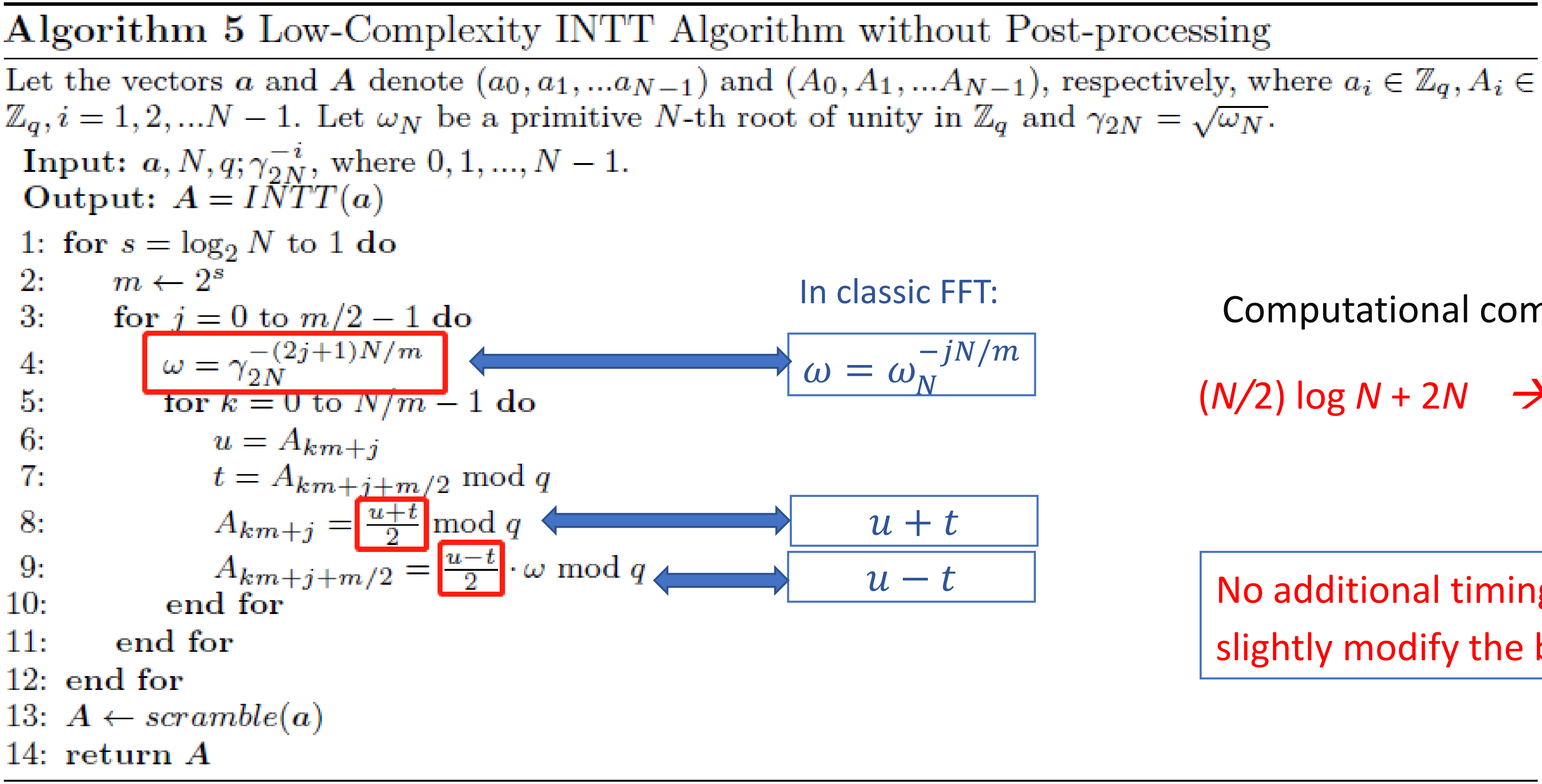
Dataflow of a 8-point low-complexity INTT



Butterfly of low-complexity INTT

$$\begin{aligned}\omega_m^{-j}\gamma_{2m}^{-1} &\equiv \gamma_{2m}^{-(2j+1)} \\ &\equiv \gamma_{2N}^{-(2j+1)N/m} \quad \pmod q\end{aligned}$$

2.2 Low-Complexity INTT



Algorithm 5 Low-Complexity INTT Algorithm without Post-processing

Let the vectors $\boldsymbol{a}$ and $\boldsymbol{A}$ denote $(a_0, a_1, ...a_{N-1})$ and $(A_0, A_1, ...A_{N-1})$, respectively, where $a_i \in \mathbb{Z}_q, A_i \in \mathbb{Z}_q, i = 1, 2, ...N-1$. Let ω_N be a primitive N-th root of unity in $\mathbb{Z}_q$ and $\gamma_{2N} = \sqrt{\omega_N}$.

Input: $\boldsymbol{a}, N, q; \gamma_{2N}^{-i}$, where $0, 1, ..., N-1$.

Output: $\boldsymbol{A} = INTT(\boldsymbol{a})$

```
for $s = \log_2 N$ to 1 do
    $m \leftarrow 2^s$
    for $j = 0$ to $m/2 - 1$ do
        $\boxed{\omega = \gamma_{2N}^{-(2j+1)N/m}}$
        for $k = 0$ to $N/m - 1$ do
            $u = A_{km+j}$
            $t = A_{km+j+m/2} \bmod q$
            $A_{km+j} = \boxed{\frac{u+t}{2}} \bmod q$
            $A_{km+j+m/2} = \boxed{\frac{u-t}{2}} \cdot \omega \bmod q$
        end for
    end for
end for
$\boldsymbol{A} \leftarrow scramble(\boldsymbol{a})$
return $\boldsymbol{A}$
```

In classic FFT:

$\omega = \omega_N^{-jN/m}$

$u + t$

$u - t$

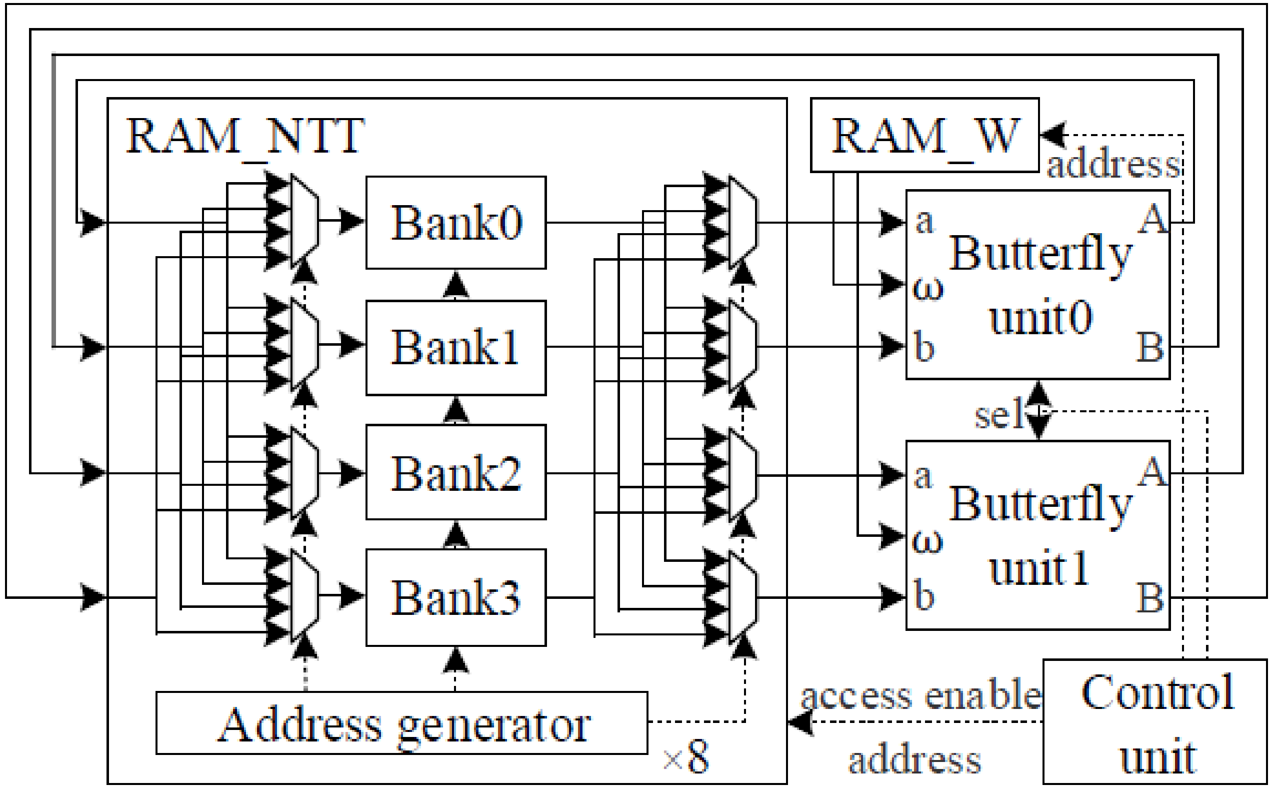
Computational complexity:

$(N/2) \log N + 2N$ → $(N/2) \log N$

No additional timing cost;
slightly modify the butterfly unit

3 The Hardware Architecture

☐ The architecture of NTT/INTT



☐ Multi-bank memory

- ➢Address generator [1] :

$$BankAddr = \sum_{i=0}^{\lceil \frac{1}{2}\log_2 N \rceil - 1} addr[2i+1:2i] \bmod 4$$

$$NewAddr = addr >> 2,$$

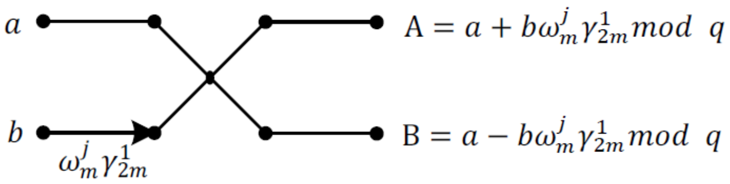
- ➢Log N: Even √ Odd ×
- ➢The execution order of the last s-loop is rearranged as :

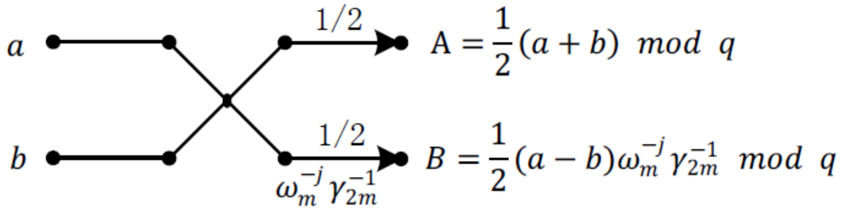
```
for $j = 0$ to $N/4 - 1$
    $A_j \leftarrow A_j + \gamma_{2N}^{2j+1} A_{j+N/2}$
    $A_{j+N/2} \leftarrow A_j - \gamma_{2N}^{2j+1} A_{j+N/2}$
    $A_{j+N/4} \leftarrow A_{j+N/4} + \gamma_{2N}^{2j+N/2+1} A_{j+3N/4}$
    $A_{j+3N/4} \leftarrow A_{j+N/4} - \gamma_{2N}^{2j+N/2+1} A_{j+3N/4}$
```

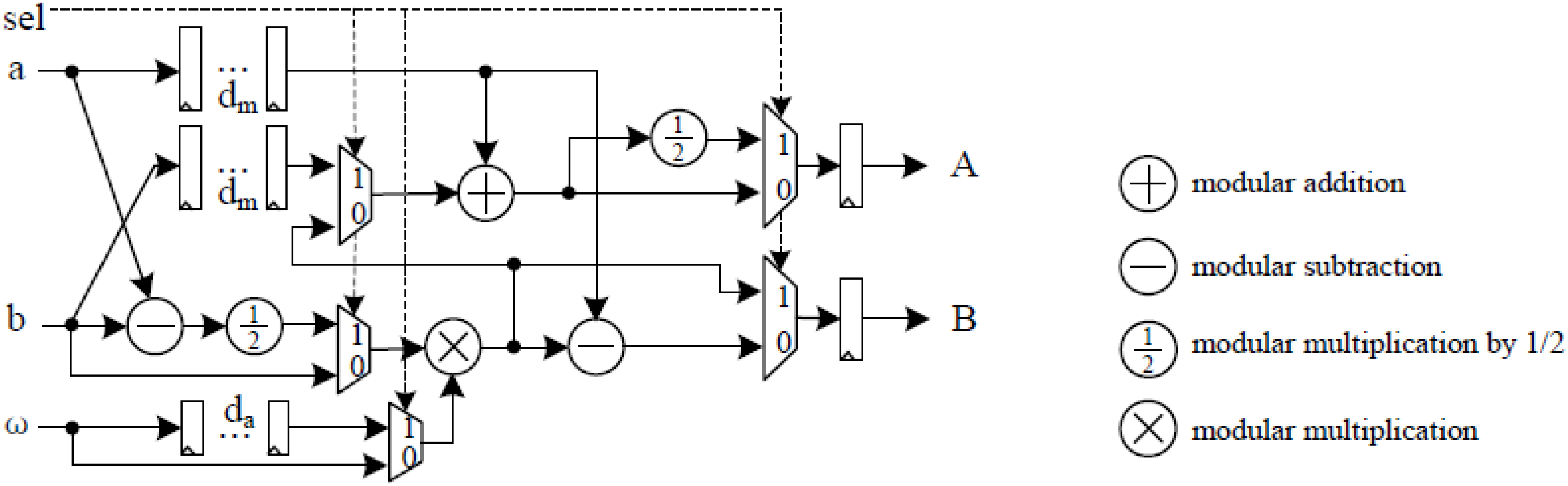
[1] W. Wang, et al., VLSI design of a large number multiplier for fully homomorphic encryption. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 22(9):1879–1887, Sept 2014.

3 The Hardware Architecture

□ Compact Butterfly Unit







$$\frac{x}{2} \equiv (2\lfloor\frac{x}{2}\rfloor + 1)\frac{q+1}{2} \equiv \lfloor\frac{x}{2}\rfloor(q+1) + \frac{q+1}{2} \equiv \lfloor\frac{x}{2}\rfloor + \frac{q+1}{2} \pmod q$$

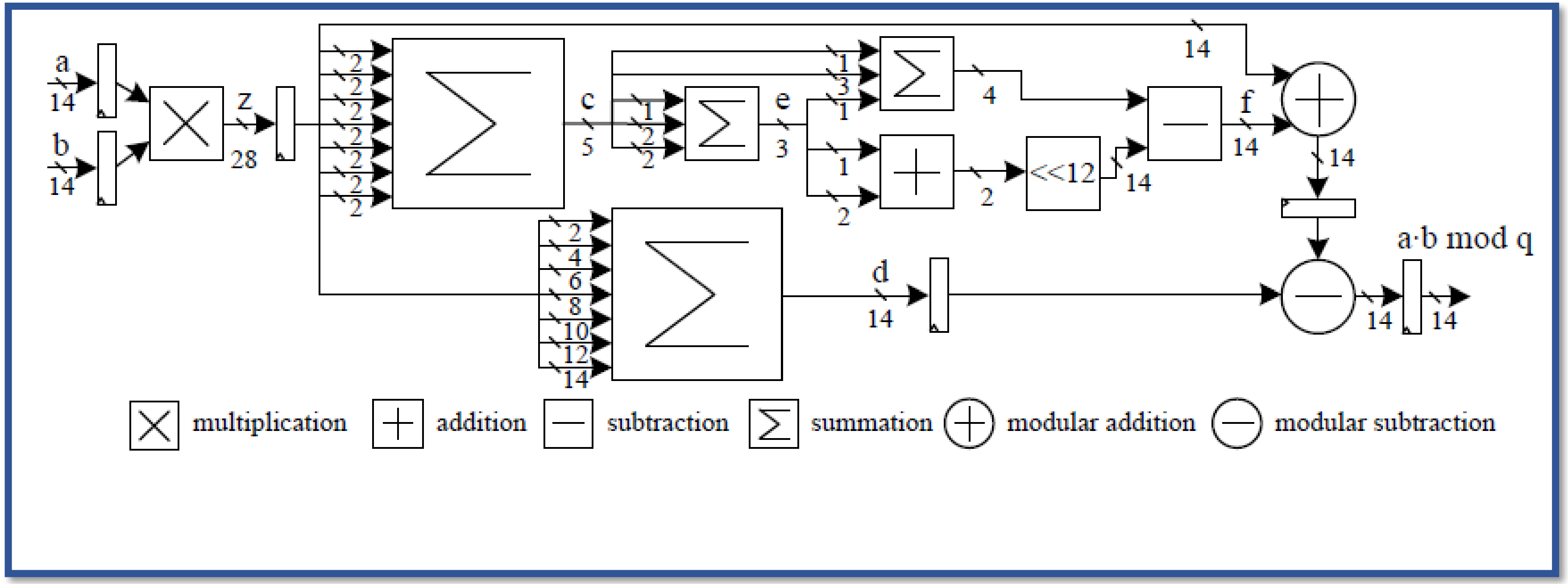
3 The Hardware Architecture

☐ Low-Complexity Modular Multiplication

$$\boxed{2^{14} \equiv 2^{12} - 1 \bmod 12289}$$

$$\begin{aligned} z &\equiv 2^{14} z[27:14] + z[13:0] \\ &\equiv 2^{12} z[27:14] - z[27:14] + z[13:0] \\ &\equiv 2^{14} z[27:16] + 2^{12} z[15:14] - z[27:14] + z[13:0] \end{aligned}$$

…



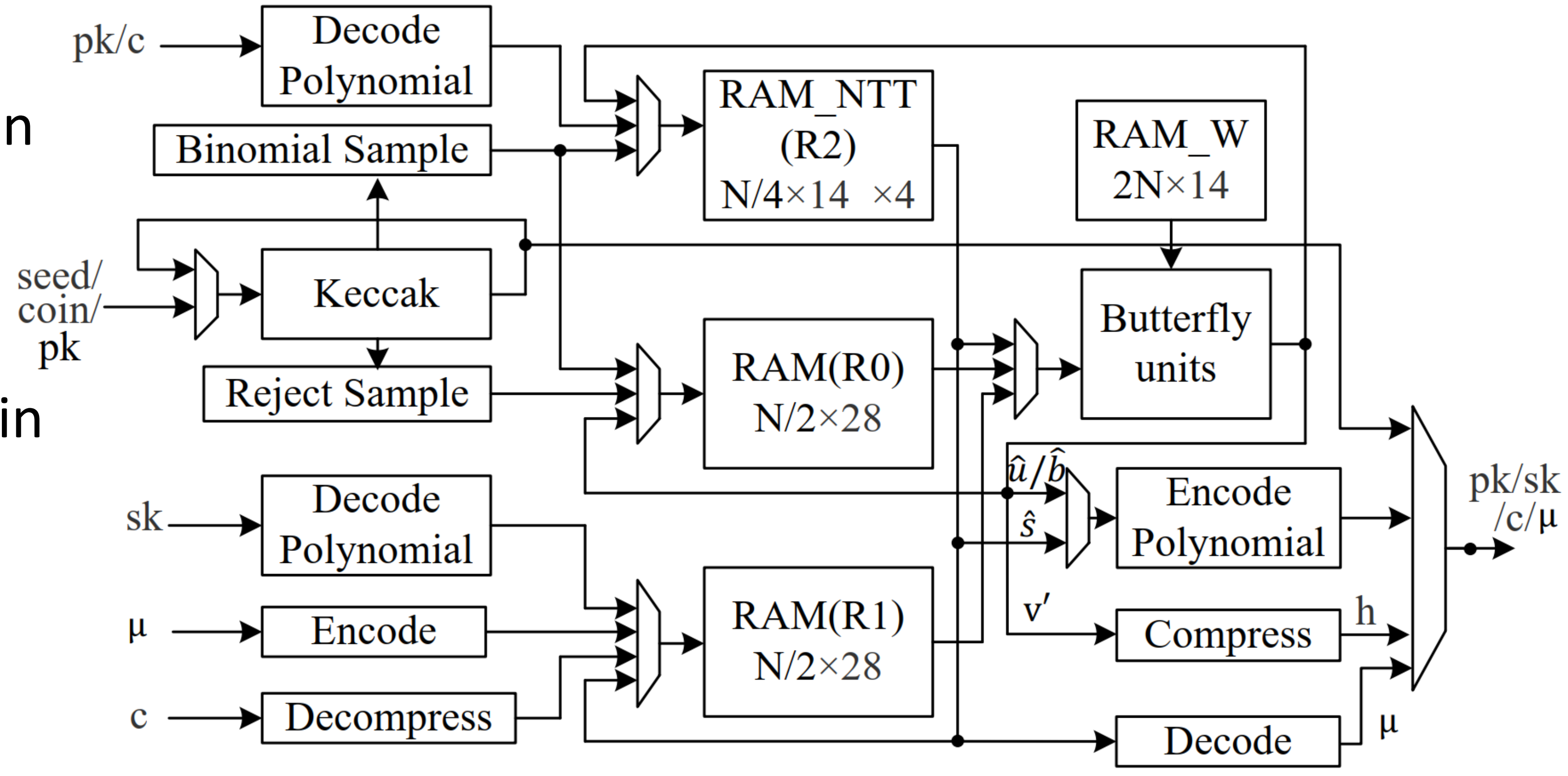
No additional multiplication;

Time-constant

3 The Hardware Architecture

☐ The architecture of NewHope-NIST

- Support: key generation, encryption and decryption
- Doubled bandwidth matching
- RAM (R0, R1): two data in an address

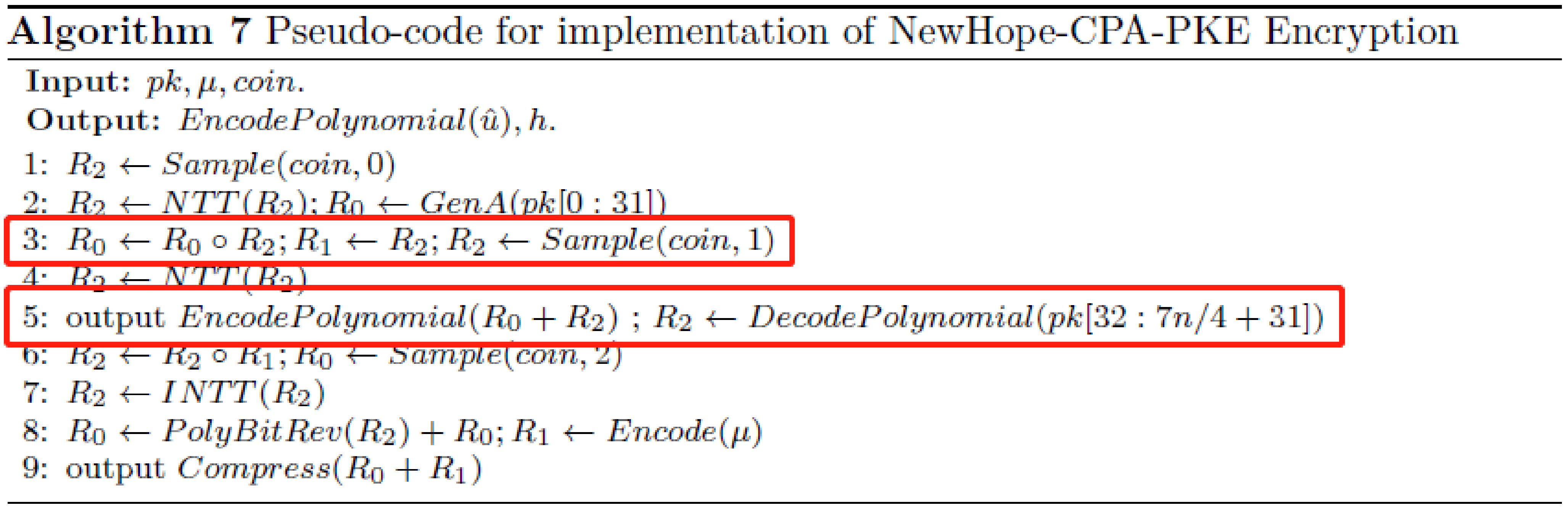


3 The Hardware Architecture

☐ Timing hiding

- ➢Resource conflict
- ➢data dependency

A RAM may be read and write by operations in the same line.



Algorithm 7 Pseudo-code for implementation of NewHope-CPA-PKE Encryption

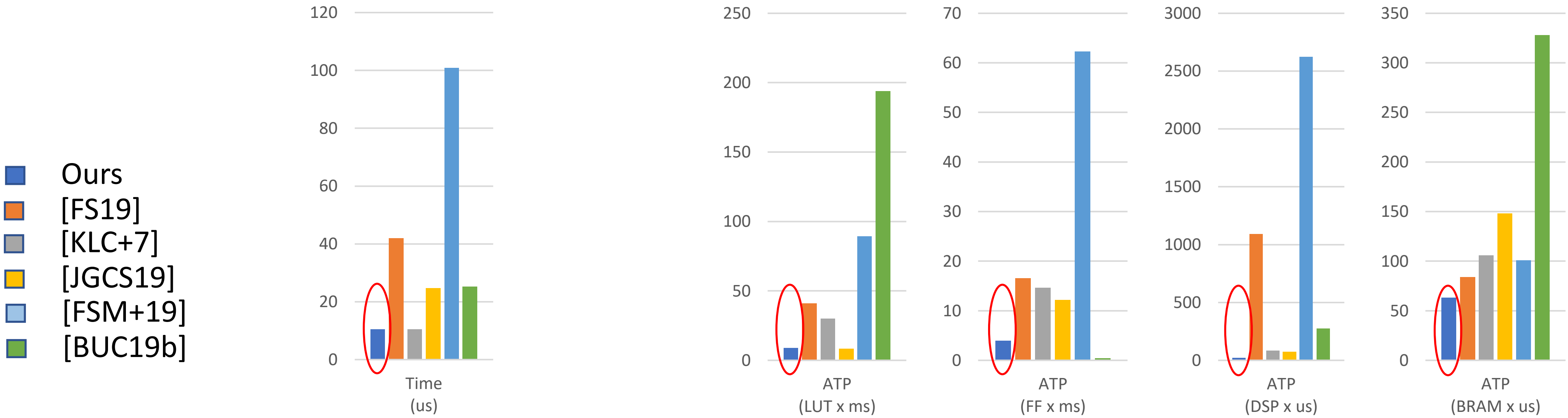
```
Input: $pk, \mu, coin$.
Output: $EncodePolynomial(\hat{u}), h$.
$R_2 \leftarrow Sample(coin, 0)$
$R_2 \leftarrow NTT(R_2); R_0 \leftarrow GenA(pk[0:31])$
$R_0 \leftarrow R_0 \circ R_2; R_1 \leftarrow R_2; R_2 \leftarrow Sample(coin, 1)$
$R_2 \leftarrow NTT(R_2)$
output $EncodePolynomial(R_0 + R_2)$ ; $R_2 \leftarrow DecodePolynomial(pk[32 : 7n/4 + 31])$
$R_2 \leftarrow R_2 \circ R_1; R_0 \leftarrow Sample(coin, 2)$
$R_2 \leftarrow INTT(R_2)$
$R_0 \leftarrow PolyBitRev(R_2) + R_0; R_1 \leftarrow Encode(\mu)$
output $Compress(R_0 + R_1)$
```

4 Implementation Results

☐ Implementation platform

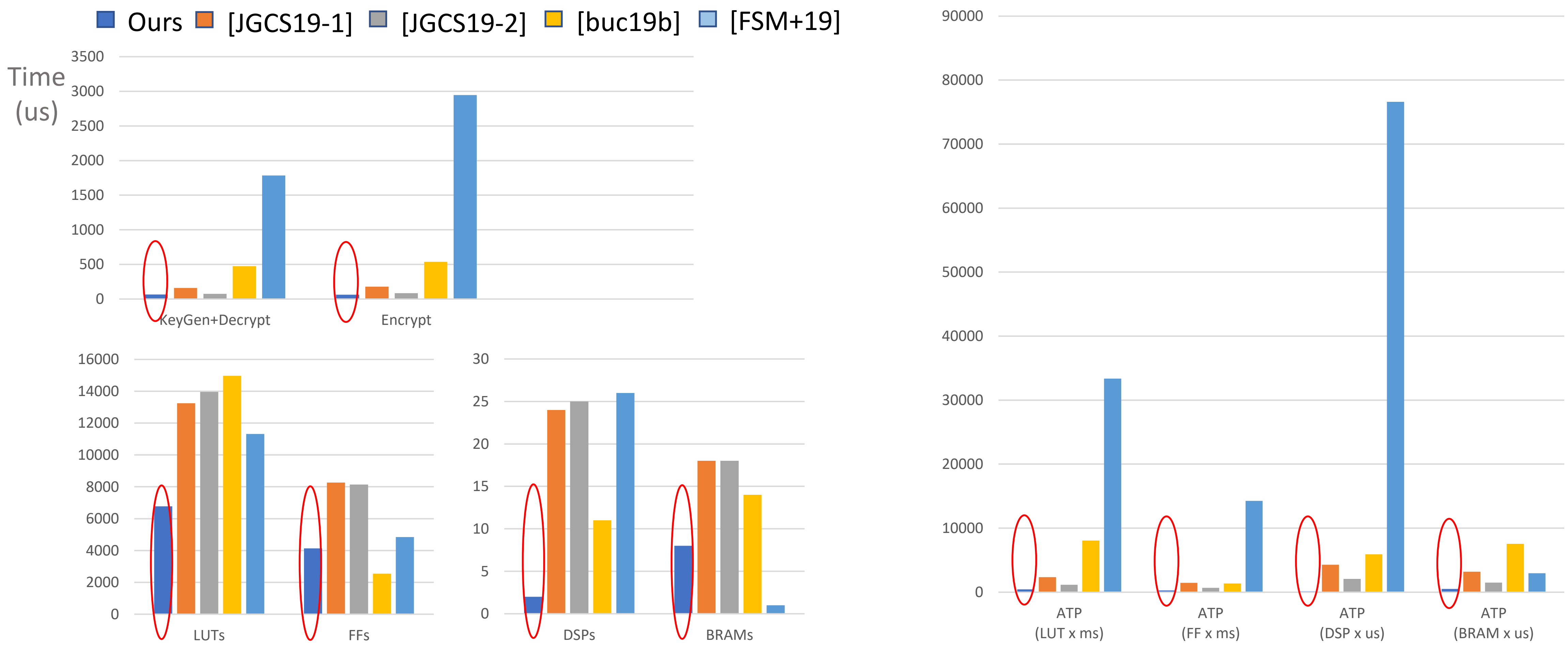
- ➢Xilinx Artix-7 FPGA
- ➢Vivado 2019.1.1

☐Implementation Results of NTT/INTT



4 Implementation Results

☐ Implementation Results of NewHope-NIST



Conclusion

☐ **Low-complexity NTT/INTT**

- ➢NTT: no pre-processing
- ➢INTT: no post-processing

☐ **A highly efficient architecture of NewHope-NIST**

- ➢A clear advantage in both speed and ATP

☐ **Low-complexity NTT/INTT can benefit other NTT-inside algorithms**

Thanks！