Efficient Implementation of Ring-LWE Encryption on High-end IoT Platform

Zhe Liu[1,2] Reza Azarderakhsh[3]
Howon Kim[4] **Hwajeong Seo**[5]

[1] College of Computer Science and Technology,
Nanjing University of Aeronautics and Astronautics, China

[2] Department of Combinatorics and Optimization, University of Waterloo, Canada

[3] Computer and Electrical Engineering and Computer Science (CEECS) and I-SENSE, USA

[4] Pusan National University, Republic of Korea

[5] Institute for Infocomm Research (I2R), Singapore

2016/12/1

Outline

Outline

Lattice-based Cryptography

- RSA and ECC: Integer Factorization and ECDLP
  - Hard problems can be solved by Shor's algorithm

Lattice-based Cryptography

- RSA and ECC: Integer Factorization and ECDLP
  - Hard problems can be solved by Shor's algorithm
- Lattice-based Cryptography: Hard for quantum computers
  - Ring-LWE Encryption schemes: proposed [EUROCRYPT'10]
    $\rightarrow$ optimized [CHES'15] (compact NTT and sampler)

Previous Works

Hardware Implementations

- Göttert et al. [CHES'12]: First hardware of Ring-LWE
- Pöppelmann et al. [Latincrypt'12] → Reparaz et al. [CHES'15]

Previous Works

Hardware Implementations

- Göttert et al. [CHES'12]: First hardware of Ring-LWE
- Pöppelmann et al. [Latincrypt'12] → Reparaz et al. [CHES'15]

Software Implementations on Embedded Processors

- 8-bit AVR processor:
  Boorghany et al. [ACM TEC'15]: NTRU, Ring-LWE
  → Pöppelmann et al. [Latincrypt'15]: BLISS
  → Liu et al. [CHES'15]: Ring-LWE

Previous Works

Hardware Implementations

- Göttert et al. [CHES'12]: First hardware of Ring-LWE
- Pöppelmann et al. [Latincrypt'12] → Reparaz et al. [CHES'15]

Software Implementations on Embedded Processors

- 8-bit AVR processor:
  Boorghany et al. [ACM TEC'15]: NTRU, Ring-LWE
  → Pöppelmann et al. [Latincrypt'15]: BLISS
  → Liu et al. [CHES'15]: Ring-LWE
- 32-bit ARM processor:
  Oder et al. [DATE'14]: BLISS
  → Boorghany et al. [ACM TEC'15]: NTRU, Ring-LWE
  → De Clercq et al. [DATE'15]: Ring-LWE

Previous Works

Hardware Implementations

- Göttert et al. [CHES'12]: First hardware of Ring-LWE
- Pöppelmann et al. [Latincrypt'12] → Reparaz et al. [CHES'15]

Software Implementations on Embedded Processors

- 8-bit AVR processor:
  Boorghany et al. [ACM TEC'15]: NTRU, Ring-LWE
  → Pöppelmann et al. [Latincrypt'15]: BLISS
  → Liu et al. [CHES'15]: Ring-LWE
- 32-bit ARM processor:
  Oder et al. [DATE'14]: BLISS
  → Boorghany et al. [ACM TEC'15]: NTRU, Ring-LWE
  → De Clercq et al. [DATE'15]: Ring-LWE
  → This work [RFIDSec'16]: Ring-LWE

Motivation & Contributions

- Motivation
  - Few 32-bit ARM–NEON implementation
  - Premier candidate for PQ-crypto (Lattice-based cryptography)

Motivation & Contributions

- Motivation
  - Few 32-bit ARM–NEON implementation
  - Premier candidate for PQ-crypto (Lattice-based cryptography)
- Contributions: Efficient implementation of Ring-LWE
  - 4-way NTT computations over NEON architecture
  - 32-bit wise SAMS2 (Shift-Add-Mul-Sub-Sub)
  - Gaussian distribution sampler with efficient PRNG

Outline

Ring-LWE Encryption Scheme

- Key generation stage: Gen($\tilde{a}$)
  - Two error polynomials $r_1, r_2 \in R_q$ from the discrete Gaussian distribution $\mathcal{X}_\sigma$ by the Knuth-Yao sampler twice:

$$\tilde{r_1} = \mathrm{NTT}(r_1),\ \tilde{r_2} = \mathrm{NTT}(r_2),\ \tilde{p} = \tilde{r_1} - \tilde{a} \cdot \tilde{r_2} \in R_q$$

  Public key $(\tilde{a}, \tilde{p})$, Private key $(\tilde{r_2})$ are obtained

Ring-LWE Encryption Scheme

- Encryption stage: Enc($\tilde{a}$, $\tilde{p}$, M)
  - Message $M \in \{0,1\}^n$ is encoded into a polynomial in the ring; Three error polynomials $e_1, e_2, e_3 \in R_q$ are sampled

$$(\tilde{C_1}, \tilde{C_2}) = (\tilde{a} \cdot \tilde{e_1} + \tilde{e_2}, \tilde{p} \cdot \tilde{e_1} + \mathrm{NTT}(e_3 + M'))$$

Ring-LWE Encryption Scheme

- Decryption stage: $\mathrm{Dec}(\tilde{C}_1,\ \tilde{C}_2,\ \tilde{r}_2)$
  - Inverse NTT has to be performed to recover M':

$$M' = \mathrm{INTT}(\tilde{r}_2 \cdot \tilde{C}_1 + \tilde{C}_2)$$

and a decoder is to recover the original message M from M'

Ring-LWE Encryption Scheme

- Number Theoretic Transform
  - Polynomial multiplication $a(x) = \sum_{i=0}^{n-1} a_i x^i \in \mathbb{Z}_q$ in the n-th roots of unity ω_n^i

Algorithm 1: Iterative Number Theoretic Transform

Require: Polynomial $a(x)$, n-th root of unity ω
Ensure: Polynomial $a(x) = \mathrm{NTT}(a)$

1: $a \leftarrow \mathrm{BitReverse}(a)$
2: **for** i from 2 by $2i$ to n **do**
3: $\quad \omega_i \leftarrow \omega_n^{n/i}$, $\omega \leftarrow 1$
4: $\quad$ **for** j from 0 by 1 to $i/2 - 1$ **do**
5: $\quad\quad$ **for** k from 0 by i to $n - 1$ **do**
6: $\quad\quad\quad$ ① $U \leftarrow a[k+j]$, ② $V \leftarrow \omega \cdot a[k+j+i/2]$
7: $\quad\quad\quad$ ③ $a[k+j] \leftarrow U + V$, ④ $a[k+j+i/2] \leftarrow U - V$
8: $\quad\quad$ **end for**
9: $\quad\quad$ $\omega \leftarrow \omega \cdot \omega_i$
10: $\quad$ **end for**
11: **end for**
12: **return** a

Ring-LWE Encryption Scheme

- Gaussian Sampler
  - Random walk by Discrete Distribution Generating tree

Algorithm 2: Low-level implementation of Knuth-Yao sampling

Require: Probability matrix P_{mat}, random number r, modulus q
Ensure: Sample value s

```
1:  d ← 0
2:  for col from 0 by 1 to MAXCOL do
3:      d ← 2d + (r&1); r ← r ≫ 1
4:      for row from MAXROW by −1 to 0 do
5:          d ← d − P_mat[row][col]
6:          if d = −1 then
7:              if (r&1) = 1 then
8:                  return q − row
9:              else
10:                 return row
11:             end if
12:         end if
13:     end for
14: end for
15: return 0
```

Outline

Implementation Platform

- 32-bit ARM–NEON Microcontroller (ARMv7)
  - High-end Internet of Things platform
    - e.g. mini computers, tablets, smartphones
  - Advanced SIMD extension (NEON)
    - 64-bit doubleword `D` register, 128-bit quadword `Q` register
    - 8/16/32/64-bit wise vector instructions

Implementation Techniques for NTT Computation

- Parameter selection (n, q, σ) for Ring-LWE
  - 128-bit security level: $(256, 7681, 11.31/\sqrt{2\pi})$
  - Discrete Gaussian sampler: 12σ

Outline

Implementation Techniques for NTT Computation

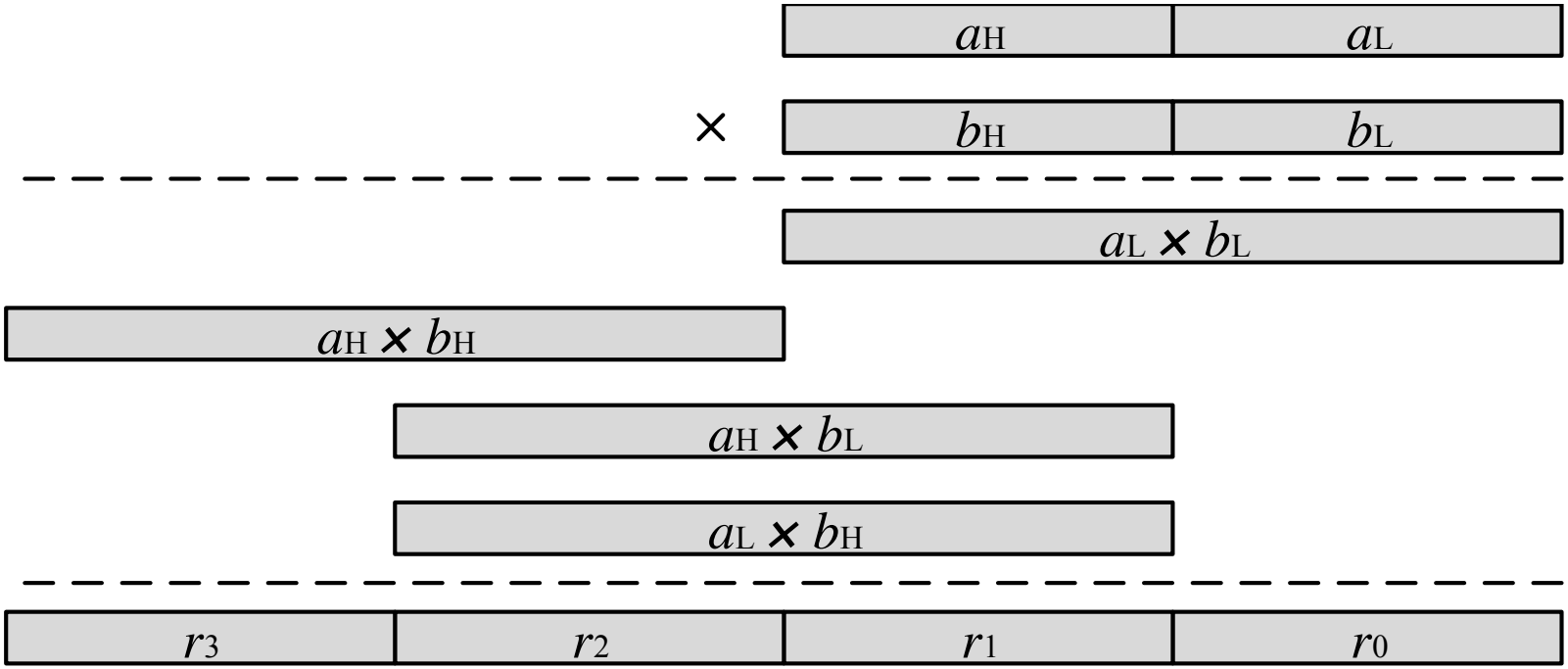
- Implementation techniques in detail
  - LUT based Twiddle Factor: ω_n and $\omega \cdot \omega_i$ [LATINCRYPT'12]
  - Negative wrapped convolution: Reduce coefficient [CHES'14]
  - Changing of the j and k-loops in the NTT [HOST'13]
  - Merging of the scaling operation by n^{-1} in INTT [CHES'14]

Implementation Techniques for NTT Computation

- Implementation techniques in detail
  - LUT based Twiddle Factor: ω_n and $\omega \cdot \omega_i$ [LATINCRYPT'12]
  - Negative wrapped convolution: Reduce coefficient [CHES'14]
  - Changing of the j and k-loops in the NTT [HOST'13]
  - Merging of the scaling operation by n^{-1} in INTT [CHES'14]
  - MOV-and-ADD mul [CHES'15] $\rightarrow$ Parallel 4-way mul
  - SAMS2 [CHES'15] $\rightarrow$ Parallel SAMS2

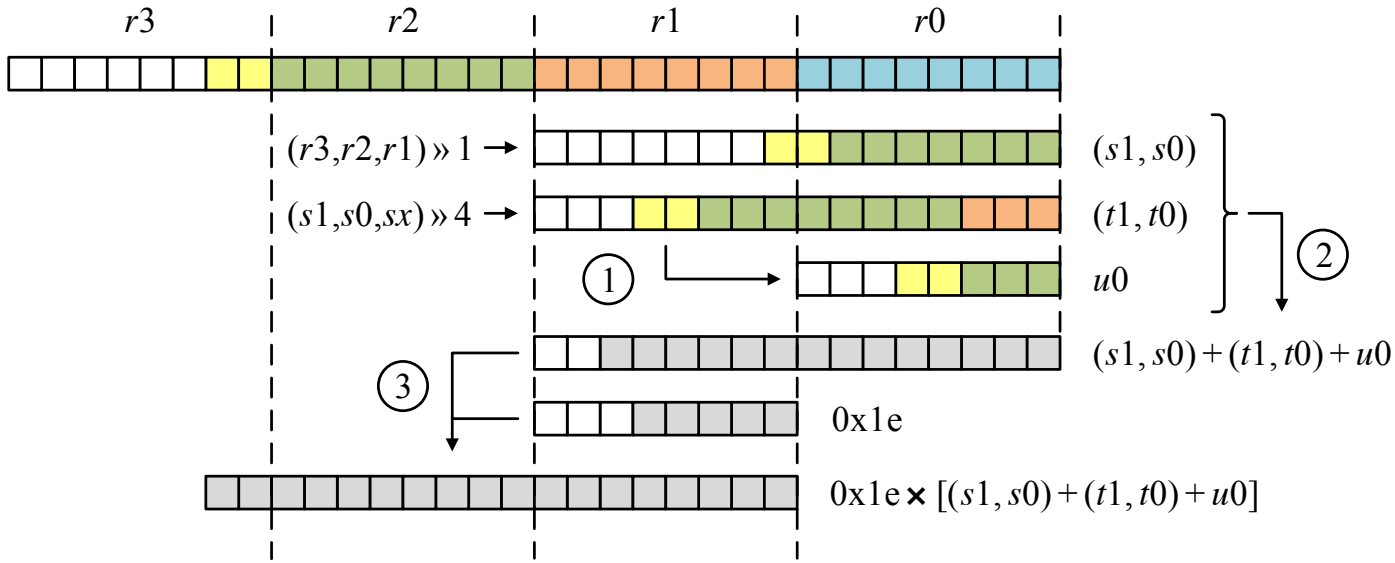
Previous Techniques for NTT Computation

MOV-and-ADD coefficient multiplication
(Total): 4 mul, 2 movw, 6 add [CHES'15]



Previous Techniques for NTT Computation

Approximation based reduction: $z \bmod q \cong z - q \times \lfloor z/q \rfloor$
SAMS2 [CHES'15], ①: shifting; ②: addition; ③: multiplication;



Ring-LWE Encryption Scheme

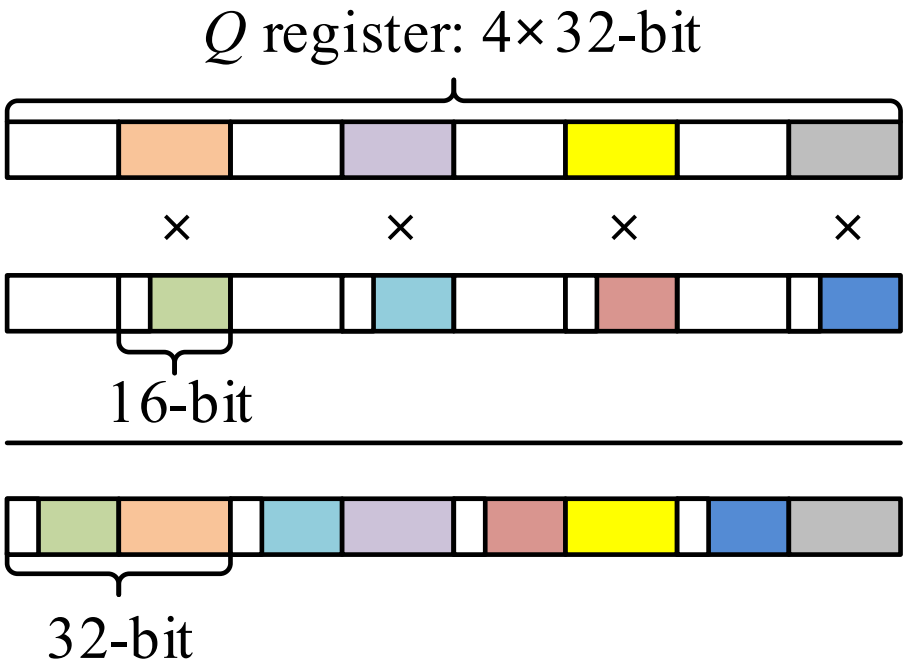
Algorithm 3: Parallel Iterative Number Theoretic Transform

Require: Polynomial $a(x)$, n-th root of unity ω
Ensure: Polynomial $a(x) = \mathrm{NTT}(a)$

1: $a \leftarrow \mathrm{BitReverse}(a)$
2: **for** i from 2 by $2i$ to n **do**
3: $\quad \omega_i \leftarrow \omega_n^{n/i}$, $\omega \leftarrow 1$
4: $\quad$ **if** $i = 2$ or $i = 4$ **then**
5: $\quad\quad \mathrm{BasicNTT}$
6: $\quad$ **else**
7: $\quad\quad \omega_{array}[0] = \omega$
8: $\quad\quad$ **for** p from 1 by 1 to $i/2 - 1$ **do**
9: $\quad\quad\quad \omega = \omega \cdot \omega_i$, $\omega_{array}[p] = \omega$
10: $\quad\quad$ **end for**
11: $\quad\quad$ **for** j from 0 by i to $n - 1$ **do**
12: $\quad\quad\quad p = 0$
13: $\quad\quad\quad$ **for** k from 0 by 4 to $i/2 - 1$ **do**
14: $\quad\quad\quad\quad U_{array} = a[k + j : k + j + 3]$, $V_{array} = \omega_{array}[p : p + 3] \cdot a[k + j + i/2 : k + j + 3 + i/2]$
15: $\quad\quad\quad\quad p = p + 4$, $a[k + j : k + j + 3] = U_{array} + V_{array}$, $a[k + j + i/2 : k + j + 3 + i/2] = U_{array} - V_{array}$
16: $\quad\quad\quad$ **end for**
17: $\quad\quad$ **end for**
18: $\quad$ **end if**
19: **end for**
20: **return** a

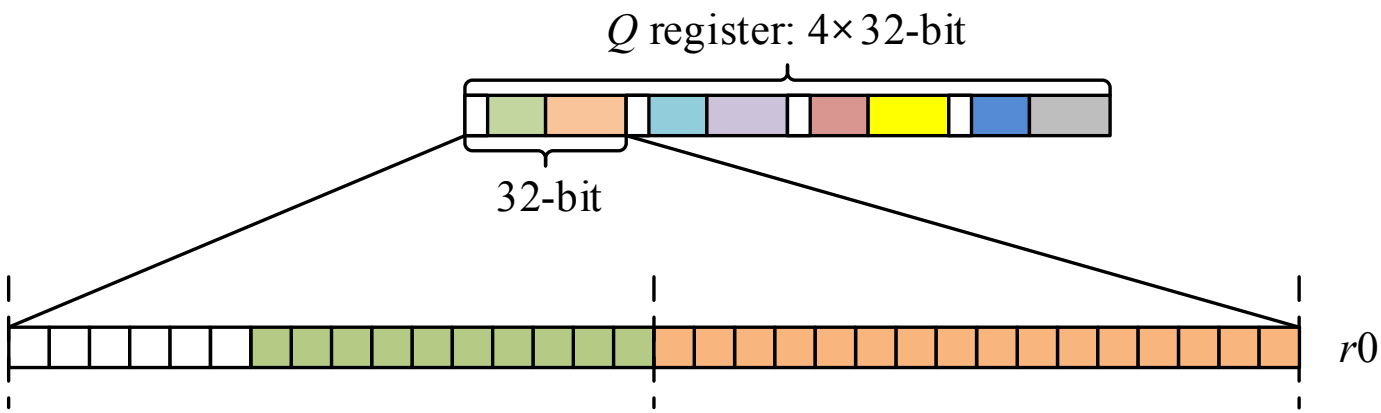
Optimization Techniques for NTT Computation

Parallel 4-way mul: four different $32 \leftarrow 32 \times 32$ mul



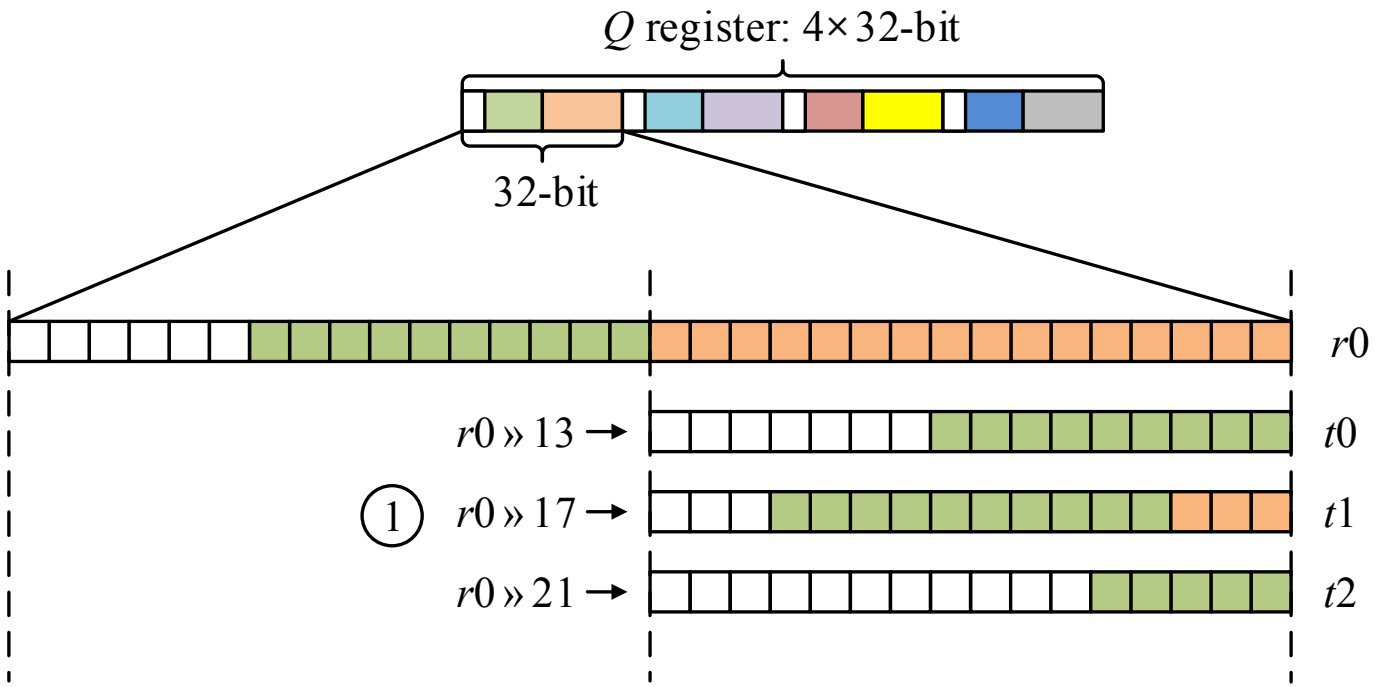
Optimization Techniques for NTT Computation

Parallel SAMS2: 13-bit wise mul $\rightarrow$ 26-bit result



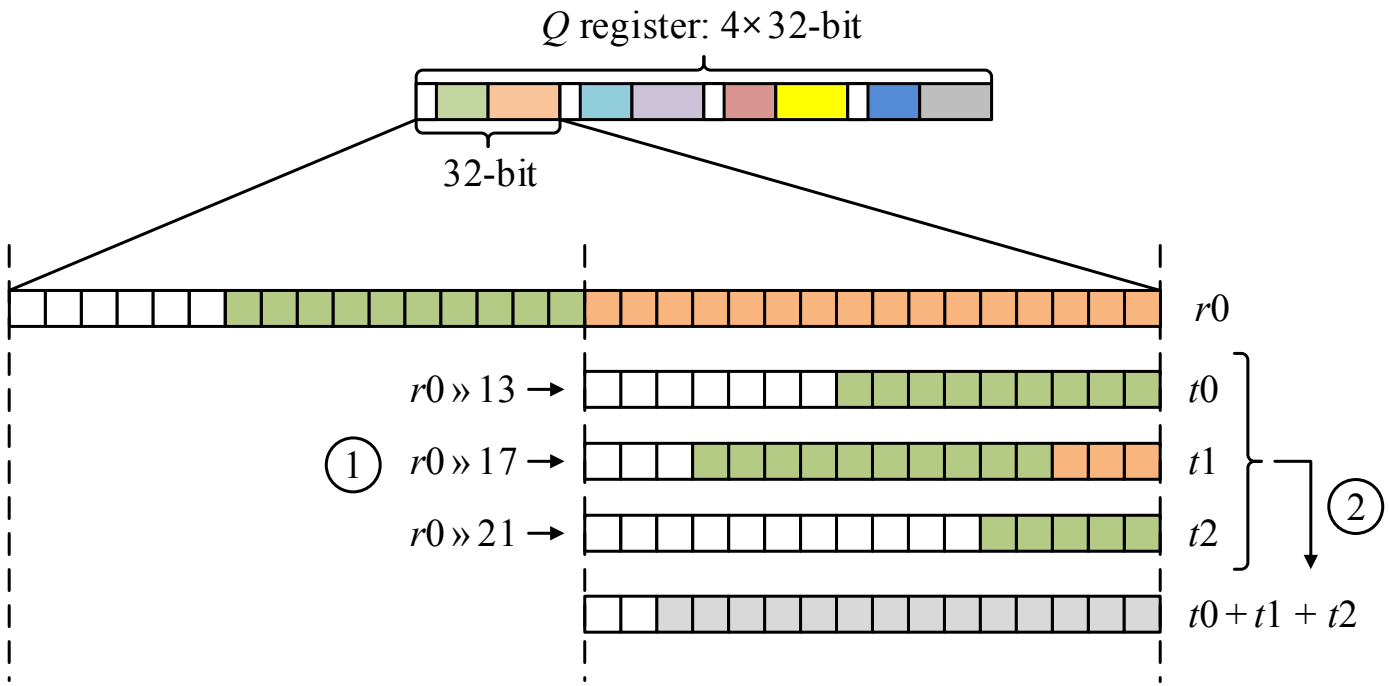
Optimization Techniques for NTT Computation

Parallel SAMS2: ①: shifting (13/17/21-bit to right);



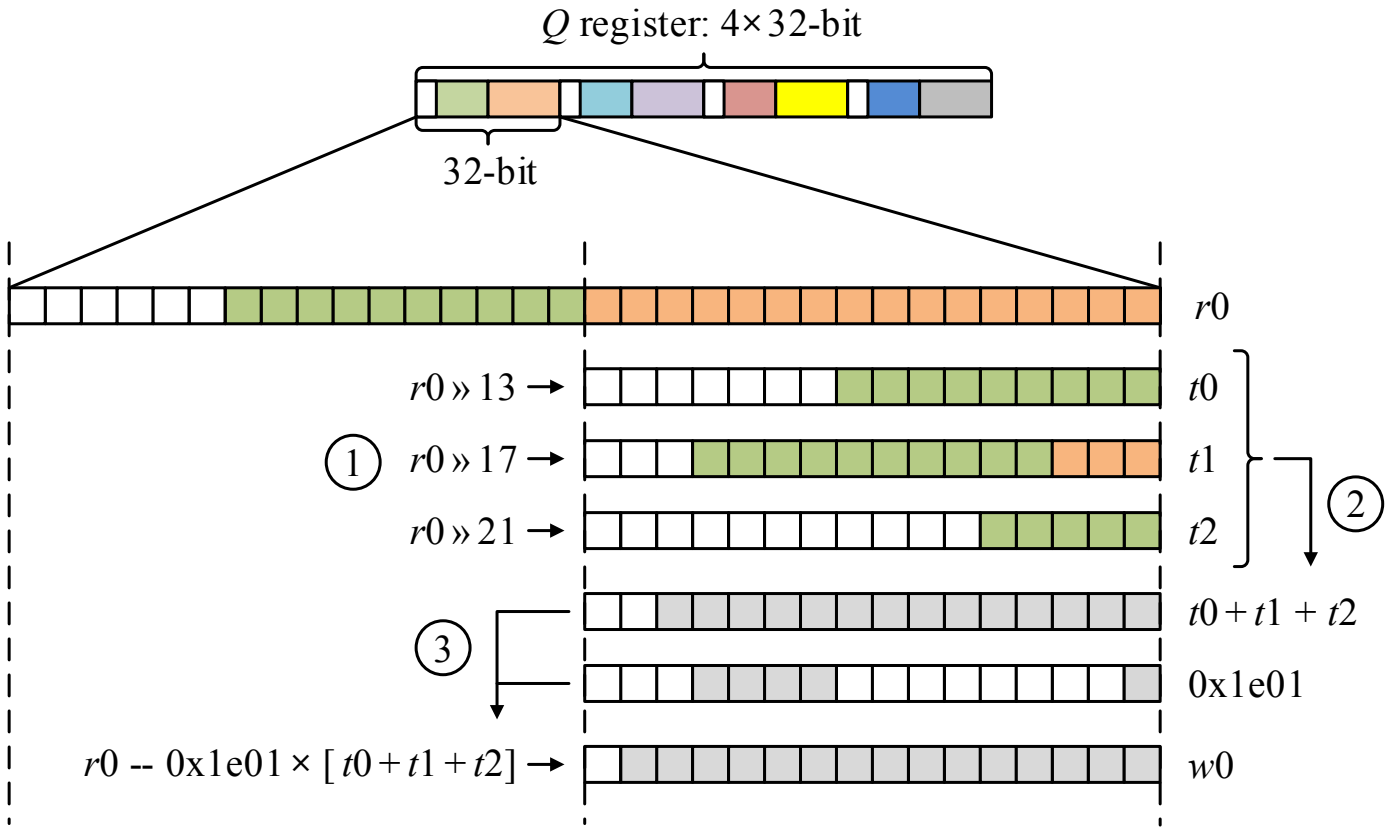
Optimization Techniques for NTT Computation

Parallel SAMS2: ②: addition (shifted results);



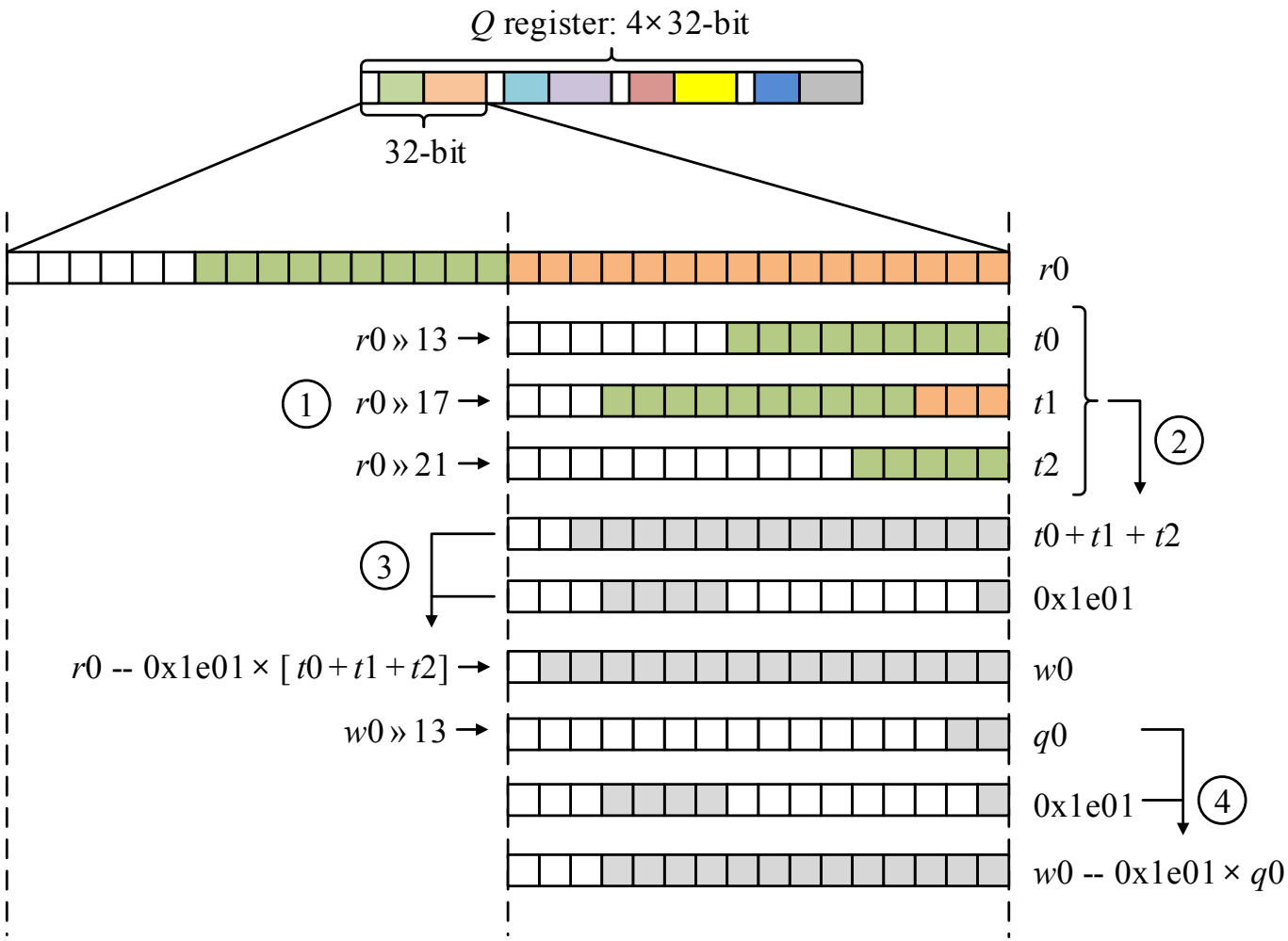
Optimization Techniques for NTT Computation

Parallel SAMS2: ③: multiplication (w/ modulus) & subtraction;



Optimization Techniques for NTT Computation

Parallel SAMS2: ④: multiplication (w/ modulus) & subtraction;



Optimization Techniques for NTT Computation

Algorithm 4: Assembly codes of vectorized NTT for innermost loop

Require: Four 32-bit coefficients $B[0:3]$(q3), ω(q1), modulo(q0).
Ensure: Four 32-bit results C(q3).

```
1: vmul.i32 q3, q3, q1      {Four 32-bit parallel multiplications}
2: vshr.u32 q4, q3, #13                          {SAMS2 ①:shifting}
3: vshr.u32 q5, q3, #17                          {SAMS2 ①:shifting}
4: vshr.u32 q6, q3, #21                          {SAMS2 ①:shifting}
5: vadd.i32 q4, q4, q5                           {SAMS2 ②:addition}
6: vadd.i32 q4, q4, q6                           {SAMS2 ②:addition}
7: vmls.i32 q3, q4, d0[0] {SAMS2 ③:multiplication & subtraction}
8: vshr.u32 q4, q3, #13                          {SAMS2 ④:shifting}
9: vmls.i32 q3, q4, d0[0] {SAMS2 ④:multiplication & subtraction}
```

Outline

Implementation Techniques for KY-sampler

- Implementation techniques in detail
  - LUT for probability matrix [DATE'15]
  - Byte-wise scanning [CHES'15]
  - Skip the consecutive leading zeros [DATE'15]

Implementation Techniques for KY-sampler

- Implementation techniques in detail
  - LUT for probability matrix [DATE'15]
  - Byte-wise scanning [CHES'15]
  - Skip the consecutive leading zeros [DATE'15]
  - PRNG–AES [CHES'15] $\rightarrow$ PRNG–LEA

Implementation Techniques for KY-sampler

- LEA block cipher
  - 128-bit block with 32-bit word
  - Key size: 128-bit, 192-bit, 256-bit in 24, 28, 32 rounds
  - No S-box and ARX architecture

Table: First and second winners of FELICS (Block Size/Key Size)

Rank	First Triathlon	Second Triathlon
1	LEA (128/128)	HIGHT (64/128)
2	SPECK (64/96)	Chaskey (128/128)
3	Chaskey (128/128)	SPECK (64/128)

- Implementation in detail
  - 8 128-bit encryption in parallel way (SIMD instruction)
  - Counter mode based PRNG for high performance

Outline

Comparison with Related Work

Implementations	NTT/FFT	Sampling	Gen	Enc	Dec
Implementations on 8-bit AVR processors, e.g., ATxmega64, ATxmega128:					
Pöppelmann et al.	334K	N/A	N/A	1,315K	381K
Liu et al.	193K	26K	589K	671K	275K
Implementations on 32-bit ARM processors: e.g., Cortex-M4:					
DeClercq et al.	31K	7K	117K	121K	43K
Implementations on 32-bit ARM-NEON processors, e.g., Cortex-A9:					
This work	**25K**	**18K**	**123K**	**145K**	**32K**

- DeClercq et al.: 1.24x faster (NTT)

Comparison with Related Work

Table: Comparison of Ring-LWE encryption schemes with RSA and ECC on ARM NEON processors (Enc and Dec in clock cycles)

Implementation	Scheme	Enc	Dec
Seo et al.	RSA-2048	535,020	20,977,660
Bernstein et al.	ECC-255	1,157,952	578,976
This work	LWE-256	**145,200**	**32,800**

- RSA: 639x faster (DEC, 2048)
- ECC: 18x faster (DEC, 255)

Outline

Conclusion

- Compact Ring-LWE encryption for 32-bit ARM–NEON platform
  - 4-way NTT computations over NEON architecture
  - 32-bit wise SAMS2 (Shift-Add-Mul-Sub-Sub)
  - Gaussian distribution sampler with efficient PRNG
- Faster than RSA (639x) and ECC (18x)

Conclusion

- Compact Ring-LWE encryption for 32-bit ARM–NEON platform
  - 4-way NTT computations over NEON architecture
  - 32-bit wise SAMS2 (Shift-Add-Mul-Sub-Sub)
  - Gaussian distribution sampler with efficient PRNG
- Faster than RSA (639x) and ECC (18x)
- Future works
  - Performance enhancements (PRNG for Sampler)
  - Countermeasure against side channel attack

Thank you for your attention