Efficient Ring-LWE Encryption on 8-bit AVR Processors

Zhe Liu[1] Hwajeong Seo[2] Sujoy Sinha Roy[3]
Johann Großschädl[1] Howon Kim[2] Ingrid Verbauwhede[3]

[1]University of Luxembourg
[2]Pusan National University
[3]Katholieke Universiteit Leuven

2015/09/16

Outline

Outline

1. Short Overview
2. Ring-LWE Encryption Scheme
3. Our Implementation
    - Optimization Techniques for NTT Computation
    - Optimization of the Knuth-Yao Sampler
4. Implementation Results
    - Comparison with Related Work
5. Conclusion

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'14] (reducing the polynomial arithmetic)

Implementation Platform

- **8-bit XMEGA128 Microcontroller**
  - Wireless Sensor Networks; Internet of Things
  - Operating Frequency: 32 MHz
  - 128KB Flash, 8KB RAM, 32 registers
  - Core instruction: 8-bit mul/add (2/1 cycles)
  - AES/DES Crypto Engine (for PRNG)

Previous Works

Hardware Implementations

- Göttert et al. [CHES'12]: First hardware of Ring-LWE
- Pöppelmann et al. [Latincrypt'12] $\rightarrow$ Roy et al. [CHES'14]

Previous Works

Hardware Implementations

- Göttert et al. [CHES'12]: First hardware of Ring-LWE
- Pöppelmann et al. [Latincrypt'12] → Roy et al. [CHES'14]

Software Implementations on Embedded Processors

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

Previous Works

Hardware Implementations

- Göttert et al. [CHES'12]: First hardware of Ring-LWE
- Pöppelmann et al. [Latincrypt'12] → Roy et al. [CHES'14]

Software Implementations on Embedded Processors

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

Motivation & Contributions

- Motivation
  - Few 8-bit AVR implementation
  - "Cryptosystem of the Future" for "Internet of the Future"

Motivation & Contributions

- Motivation
  - Few 8-bit AVR implementation
  - "Cryptosystem of the Future" for "Internet of the Future"
- Contributions: Efficient implementation of Ring-LWE
  - Fast NTT computation: "MOV-and-ADD" + "SAMS2"
  - Reducing the RAM consumption for coefficient
  - Efficient techniques for Knuth-Yao sampler: "Byte-Scanning"

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

- **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

$$(\tilde{C}_1, \tilde{C}_2) = (\tilde{a} \cdot \tilde{e}_1 + \tilde{e}_2, \tilde{p} \cdot \tilde{e}_1 + \mathrm{NTT}(e_3 + M'))$$

- **Decryption stage: 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 $a(x) = \mathrm{NTT}(a)$

```
a ← BitReverse(a)
for i from 2 by 2i to n do
    ω_i ← ω_n^{n/i}, ω ← 1
    for j from 0 by 1 to i/2 − 1 do
        for k from 0 by i to n − 1 do
            ① U ← a[k + j],        ② V ← ω · a[k + j + i/2]
            ③ a[k + j] ← U + V,    ④ a[k + j + i/2] ← U − V
        end for
        ω ← ω · ω_i
    end for
end for
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

$d \leftarrow 0$
for col from 0 by 1 to $MAXCOL$ **do**
  $d \leftarrow 2d + (r\&1)$; $r \leftarrow r \gg 1$
  for row from $MAXROW$ by -1 to 0 **do**
    $d \leftarrow d - P_{mat}[row][col]$
    if $d = -1$ **then**
      if $(r\&1) = 1$ **then**
        return $q - row$
      else
        return row
      end if
    end if
  end for
end for
return 0

Outline

Outline

1. Short Overview
2. Ring-LWE Encryption Scheme
3. Our Implementation
   - Optimization Techniques for NTT Computation
   - Optimization of the Knuth-Yao Sampler
4. Implementation Results
   - Comparison with Related Work
5. Conclusion

Optimization Techniques for NTT Computation

- **Parameter Selection** (n, q, σ) **for Ring-LWE**
  - 128-bit security level: $(256, 7681, 11.31/\sqrt{2\pi})$
  - 256-bit security level: $(512, 12289, 12.18/\sqrt{2\pi})$
  - Discrete Gaussian sampler: 12σ

Optimization Techniques for NTT Computation

- **Parameter Selection** (n, q, σ) **for Ring-LWE**
  - 128-bit security level: $(256, 7681, 11.31/\sqrt{2\pi})$
  - 256-bit security level: $(512, 12289, 12.18/\sqrt{2\pi})$
  - Discrete Gaussian sampler: 12σ
- LUT based Twiddle Factor: ω_n and $\omega \cdot \omega_i$ [LATINCRYPT'12]

Optimization Techniques for NTT Computation

- **Parameter Selection** (n, q, σ) **for Ring-LWE**
  - 128-bit security level: $(256, 7681, 11.31/\sqrt{2\pi})$
  - 256-bit security level: $(512, 12289, 12.18/\sqrt{2\pi})$
  - Discrete Gaussian sampler: 12σ
- LUT based Twiddle Factor: ω_n and $\omega \cdot \omega_i$ [LATINCRYPT'12]
- Negative wrapped convolution: Reduce coefficient [CHES'14]

Optimization Techniques for NTT Computation

- **Parameter Selection** (n, q, σ) **for Ring-LWE**
  - 128-bit security level: $(256, 7681, 11.31/\sqrt{2\pi})$
  - 256-bit security level: $(512, 12289, 12.18/\sqrt{2\pi})$
  - Discrete Gaussian sampler: 12σ
- 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]

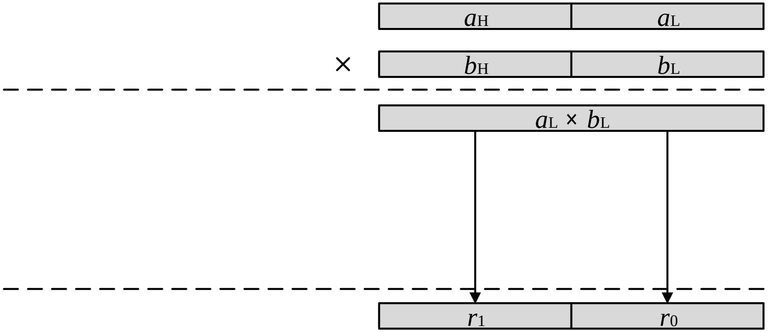
Optimization Techniques for NTT Computation

- **Parameter Selection** (n, q, σ) **for Ring-LWE**
  - 128-bit security level: $(256, 7681, 11.31/\sqrt{2\pi})$
  - 256-bit security level: $(512, 12289, 12.18/\sqrt{2\pi})$
  - Discrete Gaussian sampler: 12σ

- 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]

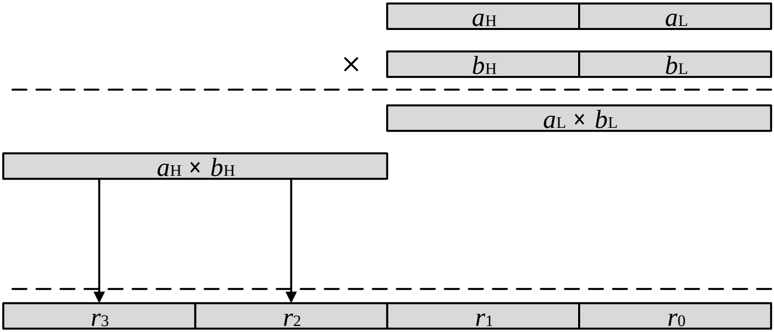
Optimization Techniques for NTT Computation

MOV-and-ADD Coefficient Multiplication (Step1): 1 mul, 1 movw



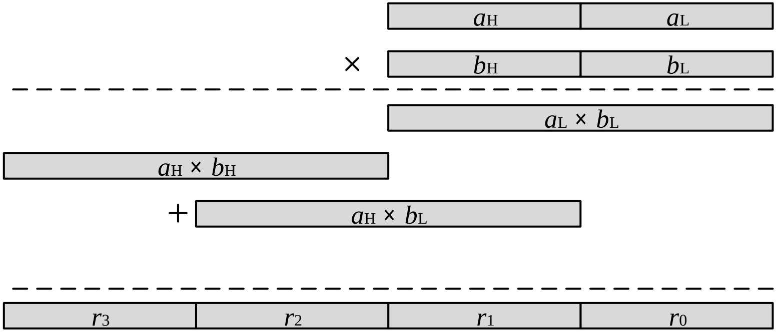
Optimization Techniques for NTT Computation

MOV-and-ADD Coefficient Multiplication (Step2): 1 mul, 1 movw



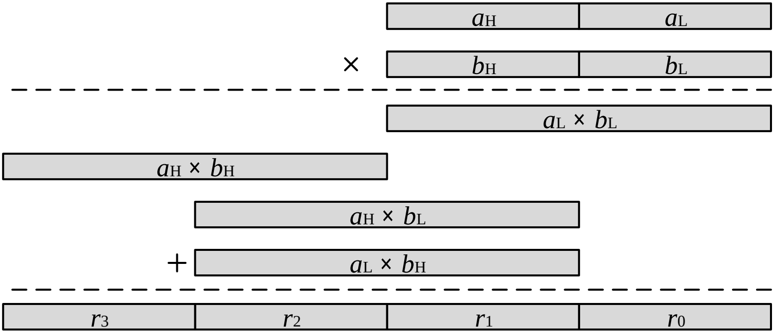
Optimization Techniques for NTT Computation

MOV-and-ADD Coefficient Multiplication (Step3): 1 mul, 3 add



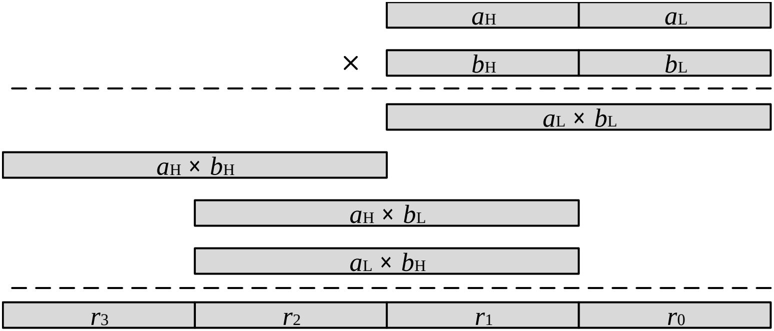
Optimization Techniques for NTT Computation

MOV-and-ADD Coefficient Multiplication (Step4): 1 mul, 3 add



Optimization Techniques for NTT Computation

MOV-and-ADD Coefficient Multiplication
(Total): 4 mul, 2 movw, 6 add instructions (16 cycles)



Optimization Techniques for NTT Computation

Approximation based reduction [ACM TEC'15]

Optimization Techniques for NTT Computation

Approximation based reduction [ACM TEC'15]

- Position of 1's in $(2^w \times 1/q) \rightarrow p_1, \ldots, p_l$

Optimization Techniques for NTT Computation

Approximation based reduction [ACM TEC'15]

- Position of 1's in $(2^w \times 1/q) \rightarrow p_1, ..., p_l$
- $\lfloor z/q \rfloor \cong \sum_{i=1}^{l} (z \gg (w - p_i))$

Optimization Techniques for NTT Computation

Approximation based reduction [ACM TEC'15]

- Position of 1's in $(2^w \times 1/q) \rightarrow p_1, \ldots, p_l$
- $\lfloor z/q \rfloor \cong \sum_{i=1}^{l} (z \gg (w - p_i))$
- $z \bmod q \cong z - q \times \lfloor z/q \rfloor$

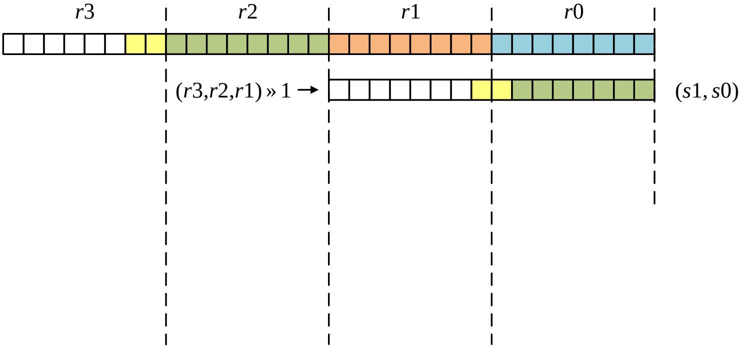
Optimization Techniques for NTT Computation

Approximation based reduction [ACM TEC'15]

- Position of 1's in $(2^w \times 1/q) \rightarrow p_1, \ldots, p_l$
- $\lfloor z/q \rfloor \cong \sum_{i=1}^{l} (z \gg (w - p_i))$
- $z \bmod q \cong z - q \times \lfloor z/q \rfloor$
- $\lfloor z/7681 \rfloor \cong (z \gg 13) + (z \gg 17) + (z \gg 21)$

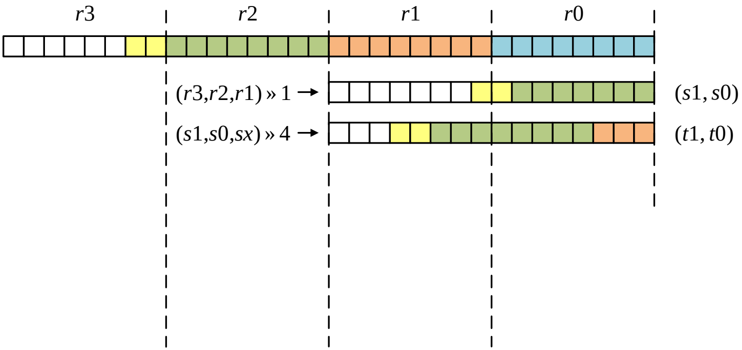
Optimization Techniques for NTT Computation

SAMS2 (Step1-1): shifting ($z \gg 17$)



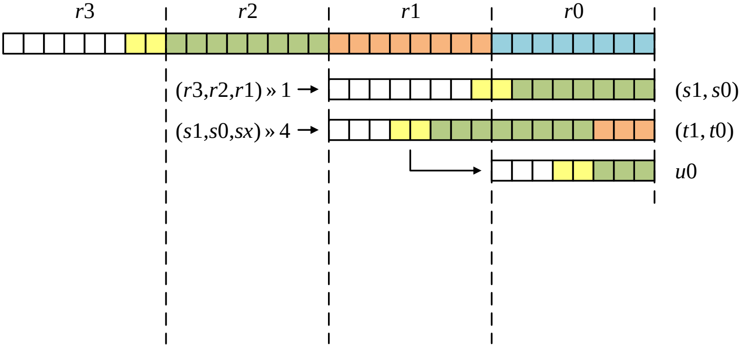
Optimization Techniques for NTT Computation

SAMS2 (Step1-2): shifting ($z \gg 13$)



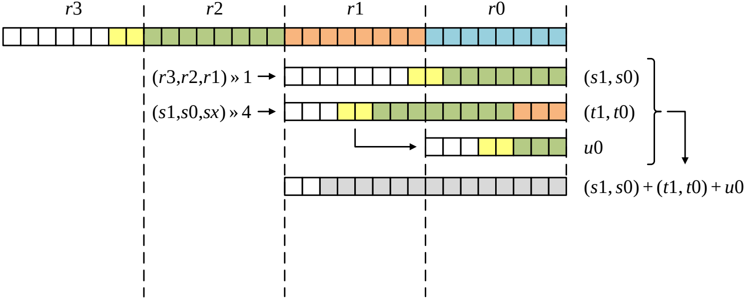
Optimization Techniques for NTT Computation

SAMS2 (Step1-3): shifting ($z \gg 21$)



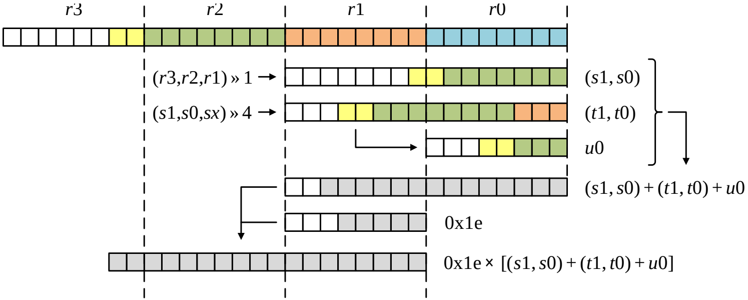
Optimization Techniques for NTT Computation

SAMS2 (Step2): addition $(z \gg 13) + (z \gg 17) + (z \gg 21)$



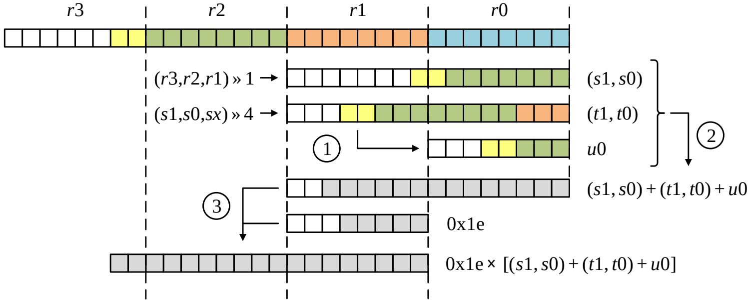
Optimization Techniques for NTT Computation

SAMS2 (Step3): multiplication



Optimization Techniques for NTT Computation

SAMS2 method, ①: shifting; ②: addition; ③: multiplication



Optimization Techniques for NTT Computation

- Incomplete modular arithmetic

Optimization Techniques for NTT Computation

- Incomplete modular arithmetic
  - Complete: $s = a + b \bmod q$

Optimization Techniques for NTT Computation

- Incomplete modular arithmetic
  - Complete: $s = a + b \bmod q$
  - Incomplete: $s = a + b \bmod 2^m$ where $m = \lceil \log_2 q \rceil$

Optimization Techniques for NTT Computation

- Incomplete modular arithmetic
  - Complete: $s = a + b \bmod q$
  - Incomplete: $s = a + b \bmod 2^m$ where $m = \lceil \log_2 q \rceil$
- Taking $q = 7681$ as an example

Optimization Techniques for NTT Computation

- Incomplete modular arithmetic
  - Complete: $s = a + b \bmod q$
  - Incomplete: $s = a + b \bmod 2^m$ where $m = \lceil \log_2 q \rceil$
- Taking $q = 7681$ as an example
  - Perform a normal coefficient addition

Optimization Techniques for NTT Computation

- Incomplete modular arithmetic
  - Complete: $s = a + b \bmod q$
  - Incomplete: $s = a + b \bmod 2^m$ where $m = \lceil \log_2 q \rceil$
- Taking $q = 7681$ as an example
  - Perform a normal coefficient addition
  - Compare the results with 2^{13}

Optimization Techniques for NTT Computation

- Incomplete modular arithmetic
  - Complete: $s = a + b \bmod q$
  - Incomplete: $s = a + b \bmod 2^m$ where $m = \lceil \log_2 q \rceil$
- Taking $q = 7681$ as an example
  - Perform a normal coefficient addition
  - Compare the results with 2^{13}
  - Conduct a subtraction of q where $r > 2^{13}$

Optimization Techniques for NTT Computation

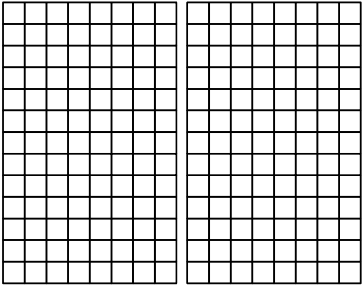
- Incomplete modular arithmetic
  - Complete: $s = a + b \bmod q$
  - Incomplete: $s = a + b \bmod 2^m$ where $m = \lceil \log_2 q \rceil$
- Taking $q = 7681$ as an example
  - Perform a normal coefficient addition
  - Compare the results with 2^{13}
  - Conduct a subtraction of q where $r > 2^{13}$
  - The operands are kept in $[0, 2^{13} - 1]$

Optimization Techniques for NTT Computation

- Incomplete modular arithmetic
  - Complete: $s = a + b \bmod q$
  - Incomplete: $s = a + b \bmod 2^m$ where $m = \lceil \log_2 q \rceil$
- Taking $q = 7681$ as an example
  - Perform a normal coefficient addition
  - Compare the results with 2^{13}
  - Conduct a subtraction of q where $r > 2^{13}$
  - The operands are kept in $[0, 2^{13} - 1]$
  - In the last iteration, the result back into the range $[0, q - 1]$

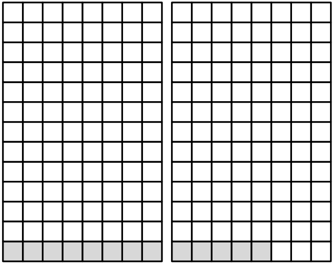
Optimization Techniques for NTT Computation

Reducing the RAM for coefficients (Step 1): Initialized registers



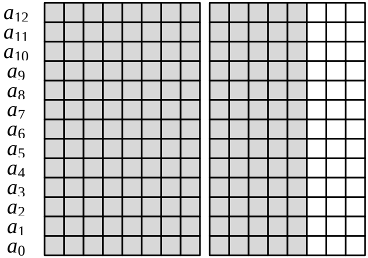
Optimization Techniques for NTT Computation

(Step 2): 13-bit coefficient (a_0) is stored

a_0 

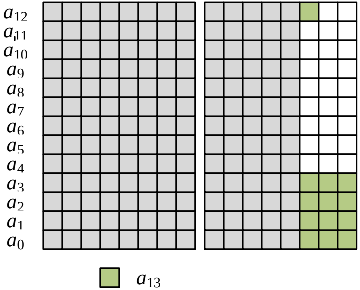
Optimization Techniques for NTT Computation

(Step 3): Other coefficients ($a_{1\sim12}$) are stored



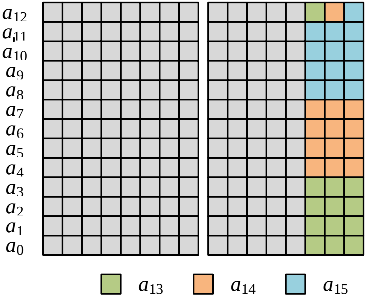
Optimization Techniques for NTT Computation

(Step 4): Coefficient (a_{13}) is stored



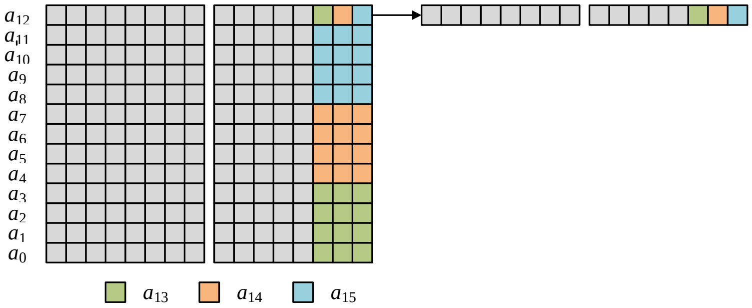
Optimization Techniques for NTT Computation

(Step 5): Remaining coefficients ($a_{14\sim15}$) are stored



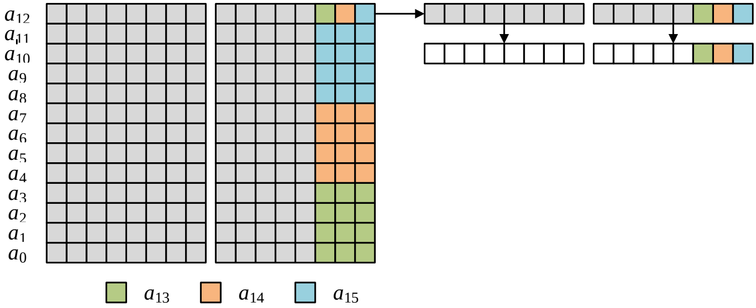
Optimization Techniques for NTT Computation

Updating coefficient (a_{12})



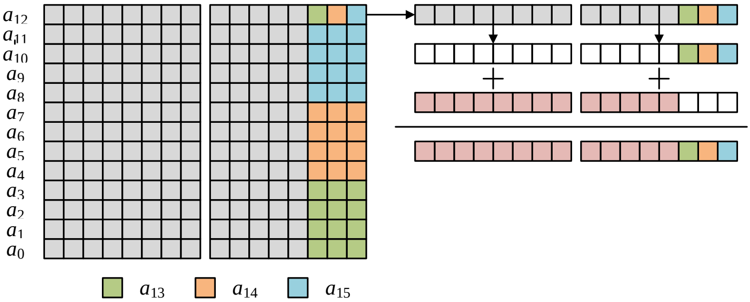
Optimization Techniques for NTT Computation

(Step 1): Clear the lower 13-bit



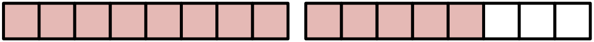
Optimization Techniques for NTT Computation

(Step 2): Add with target register



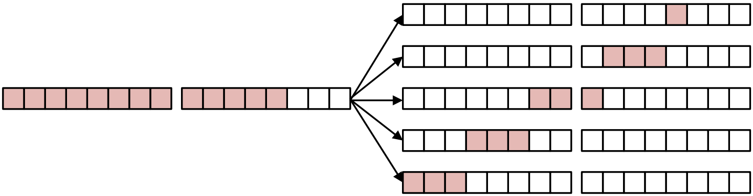
Optimization Techniques for NTT Computation

Updating coefficient (a_{13})



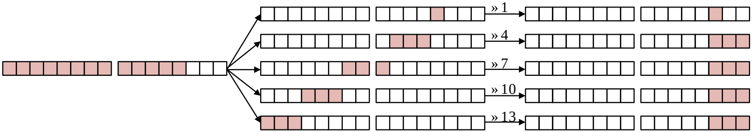
Optimization Techniques for NTT Computation

(Step 1): Divide the coefficient into 5 limbs



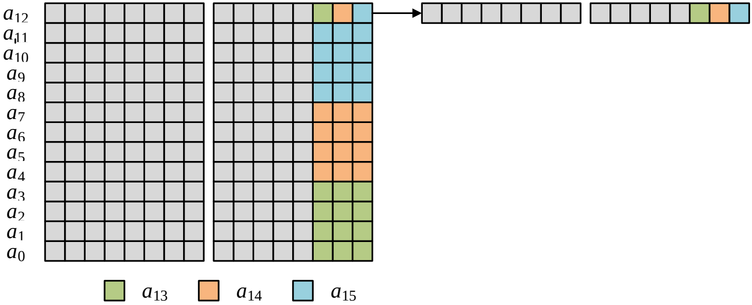
Optimization Techniques for NTT Computation

(Step 2): Shift the coefficient



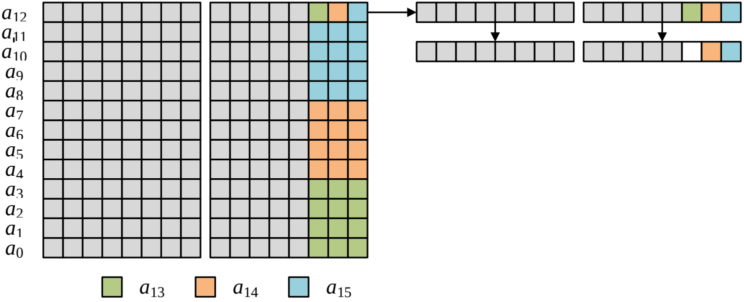
Optimization Techniques for NTT Computation

(Step 3): Select the memory



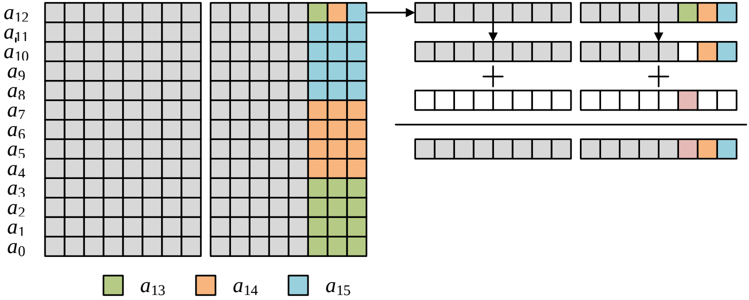
Optimization Techniques for NTT Computation

(Step 4): Clear the 14th bit



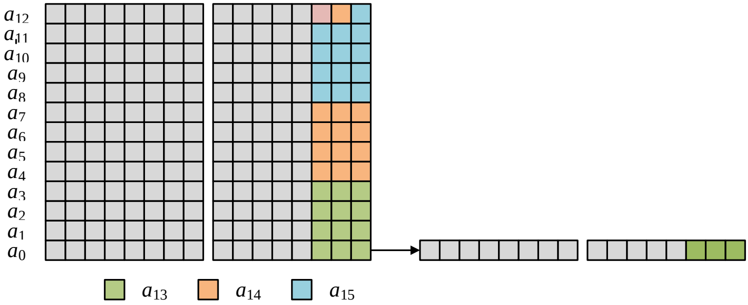
Optimization Techniques for NTT Computation

(Step 5): Add with target register



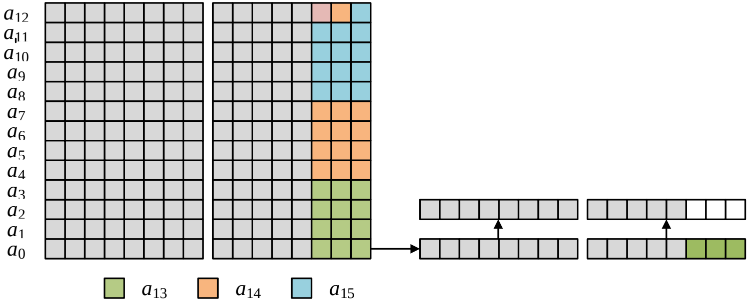
Optimization Techniques for NTT Computation

(Step 6): Select the memory



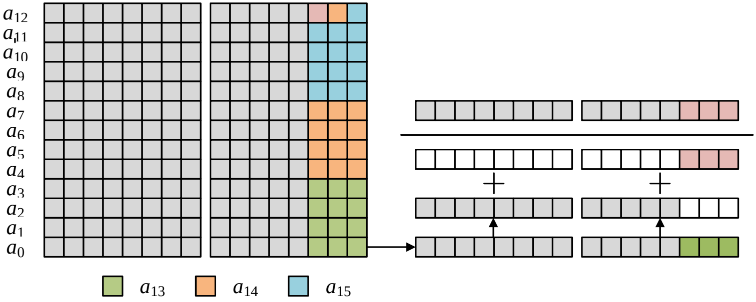
Optimization Techniques for NTT Computation

(Step 7): Clear the higher 3-bit



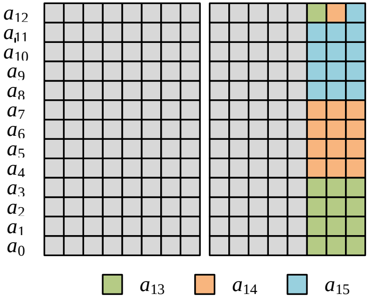
Optimization Techniques for NTT Computation

(Step 8): Add with target register



Optimization Techniques for NTT Computation

Optimized Storages: 16 13-bit elements in 26 bytes



Outline

1. Short Overview
2. Ring-LWE Encryption Scheme
3. Our Implementation
   - Optimization Techniques for NTT Computation
   - Optimization of the Knuth-Yao Sampler
4. Implementation Results
   - Comparison with Related Work
5. Conclusion

Optimization of the Knuth-Yao Sampler

Algorithm 3: Bit Scanning

1: **for** *row* from *MAXROW* by -1 to 0 **do**
2: $d \leftarrow d - P_{mat}[row][col]$ {Bit wise computations}
3: . . . **omit** . . .
4: **end for**

Algorithm 4: Byte Scanning

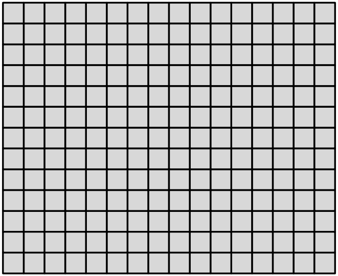
1: **for** *row* **from** *MAXROW* **by** -8 **to** 0 **do**
2: **if** $(P_{mat}[row][col] \parallel \ldots \parallel P_{mat}[row-7][col]) > 0$ **then**
3: $sum = \sum_{i=row}^{row-7}(P_{mat}[i][col])$
4: $d \leftarrow d - sum$ {Byte wise computations}
5: . . . **omit** . . .
6: **end if**
7: **end for**

Byte scanning saves 7 branch operations at the expense of 1 sub

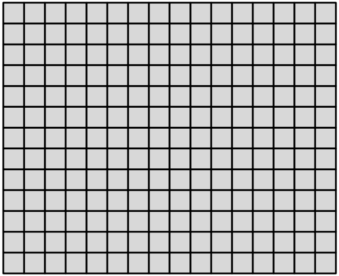
Optimization of the Knuth-Yao Sampler

Comparison between BitScanning and ByteScanning

Bit Scanning

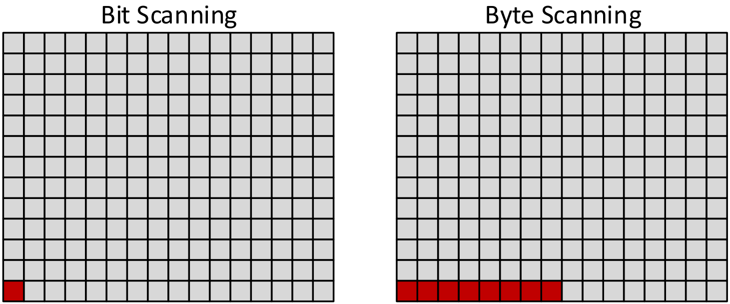


Byte Scanning



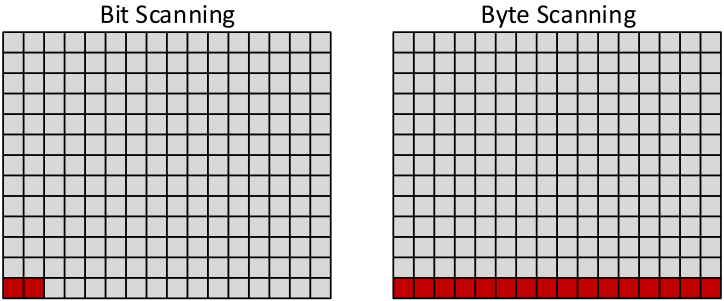
Optimization of the Knuth-Yao Sampler

(Step 1):BitScanning (1-bit), ByteScanning (1-byte)



Optimization of the Knuth-Yao Sampler

(Step 2):BitScanning (2-bit), ByteScanning (2-byte)



Optimization of the Knuth-Yao Sampler

- Pseudo Random Number Generation

Optimization of the Knuth-Yao Sampler

- Pseudo Random Number Generation
  - AES block cipher with counter mode

Optimization of the Knuth-Yao Sampler

- Pseudo Random Number Generation
  - AES block cipher with counter mode
  - ATxmega128A1 supports AES engine (375 cycles)

Optimization of the Knuth-Yao Sampler

- Pseudo Random Number Generation
  - AES block cipher with counter mode
  - ATxmega128A1 supports AES engine (375 cycles)
  - SW requires 1.9K cycles and 2KB ROM

Optimization of the Knuth-Yao Sampler

- Pseudo Random Number Generation
  - AES block cipher with counter mode
  - ATxmega128A1 supports AES engine (375 cycles)
  - SW requires 1.9K cycles and 2KB ROM
- Parallel Computations

Optimization of the Knuth-Yao Sampler

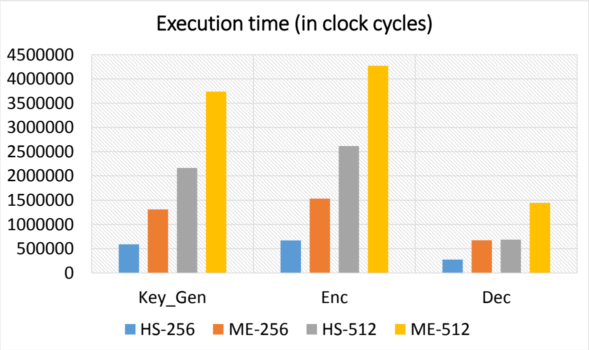
- Pseudo Random Number Generation
  - AES block cipher with counter mode
  - ATxmega128A1 supports AES engine (375 cycles)
  - SW requires 1.9K cycles and 2KB ROM
- Parallel Computations
  - AES engine and processor are executed simultaneously

Optimization of the Knuth-Yao Sampler

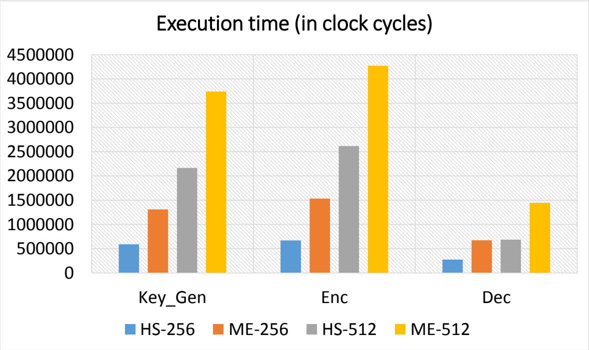
- Pseudo Random Number Generation
  - AES block cipher with counter mode
  - ATxmega128A1 supports AES engine (375 cycles)
  - SW requires 1.9K cycles and 2KB ROM
- Parallel Computations
  - AES engine and processor are executed simultaneously
  - PRNG and KY sampling in same time

Outline

Performance Evaluation

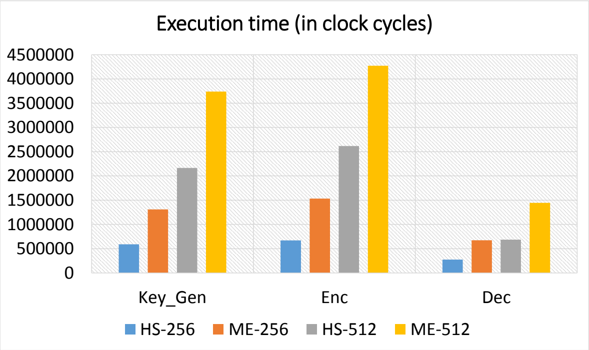


Performance Evaluation



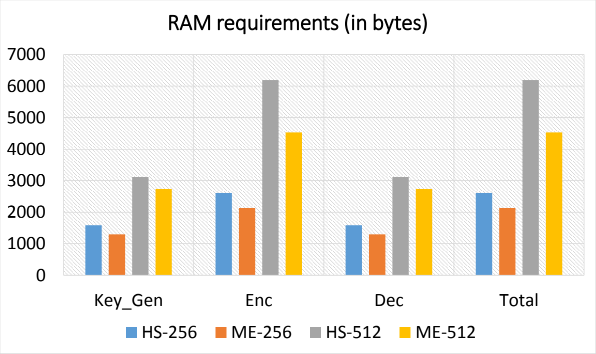
- High speed (HS) is 2.3x faster than memory efficient (ME)

Performance Evaluation

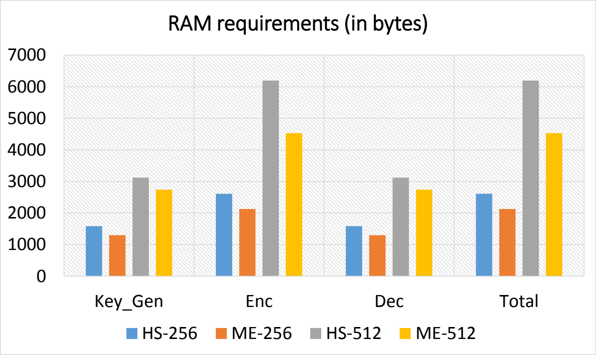


- High speed (HS) is 2.3x faster than memory efficient (ME)
- ME version requires sophisticated memory alignments

Memory Evaluation

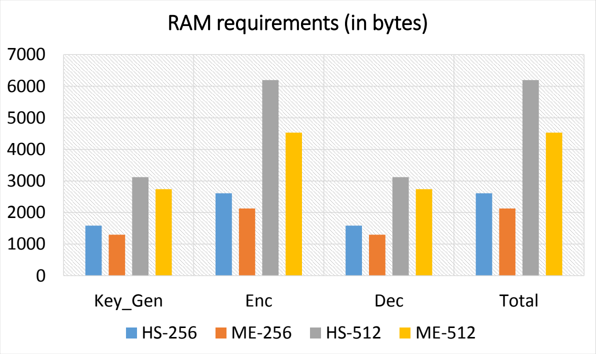


Memory Evaluation



- Compared to HS, ME version reduces the RAM by 21 %

Memory Evaluation



- Compared to HS, ME version reduces the RAM by 21 %
- HS and ME consume the 8K RAM by 77 % and 56 %

Outline

Comparison with Related Work

Implementation	Key-Gen	Enc	Dec
Boorghany (256)	2,770K	3,042K	1,368K
Pöppelmann (256)	n/a	1,314K	381K
This work (HS-256)	589K	671K	275K
Pöppelmann (512)	n/a	3,279K	1,019K
This work (HS-512)	2,165K	2,617K	686K

Comparison with Related Work

Implementation	Key-Gen	Enc	Dec
Boorghany (256)	2,770K	3,042K	1,368K
Pöppelmann (256)	n/a	1,314K	381K
This work (HS-256)	589K	671K	275K
Pöppelmann (512)	n/a	3,279K	1,019K
This work (HS-512)	2,165K	2,617K	686K

- Boorghany et al.: 4.5x faster (ENC, 256)

Comparison with Related Work

Implementation	Key-Gen	Enc	Dec
Boorghany (256)	2,770K	3,042K	1,368K
Pöppelmann (256)	n/a	1,314K	381K
This work (HS-256)	589K	671K	275K
Pöppelmann (512)	n/a	3,279K	1,019K
This work (HS-512)	2,165K	2,617K	686K

- Boorghany et al.: 4.5x faster (ENC, 256)
- Pöppelmann et al.: 2x and 1.25x faster (ENC, 256/512)

Comparison with Related Work

Implementation	Scheme	Enc	Dec
Liu et al.	RSA-1024	n/a	75,680K
Düll et al. (HS)	ECC-255	27,800K	13,900K
Düll et al. (ME)	ECC-255	28,293K	14,146K
Aranha et al.	ECC-233	11,796K	5,898K
This work (HS)	LWE-256	671K	275K
This work (ME)	LWE-256	1,532K	673K

Comparison with Related Work

Implementation	Scheme	Enc	Dec
Liu et al.	RSA-1024	n/a	75,680K
Düll et al. (HS)	ECC-255	27,800K	13,900K
Düll et al. (ME)	ECC-255	28,293K	14,146K
Aranha et al.	ECC-233	11,796K	5,898K
This work (HS)	LWE-256	671K	275K
This work (ME)	LWE-256	1,532K	673K

- RSA: 278x faster (DEC, 1024)

Comparison with Related Work

Implementation	Scheme	Enc	Dec
Liu et al.	RSA-1024	n/a	75,680K
Düll et al. (HS)	ECC-255	27,800K	13,900K
Düll et al. (ME)	ECC-255	28,293K	14,146K
Aranha et al.	ECC-233	11,796K	5,898K
This work (HS)	LWE-256	671K	275K
This work (ME)	LWE-256	1,532K	673K

- RSA: 278x faster (DEC, 1024)
- ECC: 41x faster (ENC, 255)

Outline

1. Short Overview
2. Ring-LWE Encryption Scheme
3. Our Implementation
   - Optimization Techniques for NTT Computation
   - Optimization of the Knuth-Yao Sampler
4. Implementation Results
   - Comparison with Related Work
5. Conclusion

Conclusion

- Compact Ring-LWE encryption for 8-bit platform

Conclusion

- Compact Ring-LWE encryption for 8-bit platform
  - Fast NTT computation: "MOV-and-ADD" + "SAMS2"

Conclusion

- Compact Ring-LWE encryption for 8-bit platform
  - Fast NTT computation: "MOV-and-ADD" + "SAMS2"
  - Reducing the RAM consumption for coefficient

Conclusion

- Compact Ring-LWE encryption for 8-bit platform
  - Fast NTT computation: "MOV-and-ADD" + "SAMS2"
  - Reducing the RAM consumption for coefficient
  - Efficient techniques for Knuth-Yao sampler: "Byte-Scanning"

Conclusion

- Compact Ring-LWE encryption for 8-bit platform
  - Fast NTT computation: "MOV-and-ADD" + "SAMS2"
  - Reducing the RAM consumption for coefficient
  - Efficient techniques for Knuth-Yao sampler: "Byte-Scanning"
- Faster than RSA (278x) and ECC (41x)

Conclusion

- Compact Ring-LWE encryption for 8-bit platform
  - Fast NTT computation: "MOV-and-ADD" + "SAMS2"
  - Reducing the RAM consumption for coefficient
  - Efficient techniques for Knuth-Yao sampler: "Byte-Scanning"
- Faster than RSA (278x) and ECC (41x)

More information:

Conclusion

- Compact Ring-LWE encryption for 8-bit platform
  - Fast NTT computation: "MOV-and-ADD" + "SAMS2"
  - Reducing the RAM consumption for coefficient
  - Efficient techniques for Knuth-Yao sampler: "Byte-Scanning"
- Faster than RSA (278x) and ECC (41x)

More information:

- Software is available (contacting the authors)

Conclusion

- Compact Ring-LWE encryption for 8-bit platform
  - Fast NTT computation: "MOV-and-ADD" + "SAMS2"
  - Reducing the RAM consumption for coefficient
  - Efficient techniques for Knuth-Yao sampler: "Byte-Scanning"
- Faster than RSA (278x) and ECC (41x)

More information:

- Software is available (contacting the authors)

Thank you for your attention