Secure Number Theoretic Transform and Speed Record for Ring-LWE Encryption on Embedded Processors*

Hwajeong Seo[1], Zhe Liu[2], Taehwan Park[3],
Hyeokchan Kwon[4], Sokjoon Lee[4], and Howon Kim[3]

[1] Department of IT, Hansung University,
116 Samseongyoro-16gil Seongbuk-gu Seoul 136-792, Republic of Korea
hwajeong@hansung.ac.kr
[2] APSIA, Interdisciplinary Centre for Security,
Reliability and Trust (SnT), University of Luxembourg, Luxembourg.
sduliuzhe@gmail.com
[3] Pusan National University,
School of Computer Science and Engineering,
San-30, Jangjeon-Dong, Geumjeong-Gu, Busan 609–735, Republic of Korea
{pth5804,howonkim}@pusan.ac.kr
[4] System Security Research Group,
Electronics and Telecommunciations Research Institute,
Daejeon, 34129, Korea
{hckwon,junny}@etri.re.kr

Abstract. Compact implementations of the ring variant of the Learning with Errors (Ring-LWE) on the embedded processors have been actively studied due to potential quantum threats. Various Ring-LWE implementation works mainly focused on optimization techniques to reduce the execution timing and memory consumptions for high availability. For this reason, they failed to provide secure implementations against general side channel attacks, such as timing attack. In this paper, we present secure and fastest Ring-LWE encryption implementation on low-end 8-bit AVR processors. We targeted the most expensive operation, i.e. Number Theoretic Transform (NTT) based polynomial multiplication, to provide countermeasures against timing attacks and best performance among similar implementations till now. Our contributions for optimizations are concluded as follows: (1) we propose the Look-Up Table (LUT) based fast reduction techniques for speeding up the modular coefficient multiplication in regular fashion, (2) we use the modular addition and subtraction operations, which are performed in constant timing. With these optimization techniques, the proposed NTT implementation enhances the

* This research of Hwajeong Seo was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2017-2014-0-00743) supervised by the IITP(Institute for Information & communications Technology Promotion), This work of Hyeokchan Kwon and Sokjoon Lee was supported by the ICT R&D program of MSIP/IITP. [B0717-16-0097, Development of V2X Service Integrated Security Technology for Autonomous Driving Vehicle].

performance by 18.3∼22% than previous works. Finally, our Ring-LWE encryption implementations require only 680,796 and 1,754,064 clock cycles for 128-bit and 256-bit security levels, respectively.

Keywords: ring learning with errors, software implementation, public key encryption, 8-bit AVR, number theoretic transform, discrete Gaussian sampling, timing attack

1 Introduction

Classic public key cryptography algorithms such as RSA and Elliptic Curve Cryptography (ECC) are built based on integer factorization and discrete logarithm problems, which are believed to be secure against classical computer environments with properly chosen parameters. For this reason, a number of works focused on compact implementations of RSA and ECC [17, 22, 5, 21, 13, 23, 9, 12, 24, 10, 8, 6]. However, such hard problems can be solved using Shor's algorithm on a sufficient large quantum computer in polynomial time [25]. To defeat potential attacks and threats, lattice-based cryptography is considered as one of the most promising candidates for post-quantum cryptography. Lattice-based cryptography is built based on worst-case computational assumptions in lattices that would remain hard even for quantum computers. Furthermore, the emerging Internet of Things (IoT) technology introduces new computing environments including all kinds of sensors, actuators, meters, consumer electronics, medical monitors, household appliances and vehicles. Since these devices are very resource-constrained in terms of computing power, power supply and memory resources, implementing public-key cryptographic algorithms on low-end 8-bit processors poses a big challenge. Therefore, it is necessary to further study the post-quantum cryptosystems on the low-end IoT devices.

The introduction of Learning with Errors (LWE) problem and its ring variant (Ring-LWE) [18, 14] provide efficient ways to build lattice-based public key cryptosystems. The following software implementations of Ring-LWE based public-key encryption or digital signature schemes improved performance and memory requirements: Oder et al. presented an efficient implementation of Bimodal Lattice Signature Scheme (BLISS) on a 32-bit ARM Cortex-M4F microcontroller [15]. De Clercq et al. implemented Ring-LWE encryption scheme on the identical ARM processors [3]. They utilized 32-bit registers to retain two $13 \sim 14$ coefficients. Boorghany et al. implemented a lattice-based cryptographic scheme on an 8-bit processor for the first time in [1, 2]. The authors evaluated four lattice-based authentication protocols on both 8-bit AVR and 32-bit ARM processors. In particular, Fast Fourier Transform (FFT) transform and Gaussian sampler function are implemented. In LATINCRYPT'15, Pöppelmann et al. studied and compared implementations of Ring-LWE encryption and BLISS on an 8-bit Atmel ATxmega128 microcontroller [16]. In CHES'15, Liu et al. optimized implementations of Ring-LWE encryption by presenting efficient modular multiplication, NTT computation and refined memory access schemes to achieve high performance and low memory consumption [11]. They presented two implementations of Ring-LWE encryption scheme for both medium-term and long-term security levels on an 8-bit AVR processor. Liu et al. presented the first secure Ring-LWE encryption and BLISS signature implementations against timing attack [7]. NTT and sampling computations are implemented in constant time to prevent timing attack. Particularly, modular reduction is performed in Montgomery reduction to reduce computation complexity. Recently, in [4, 9], high efficient implementations on ARM-NEON and MSP430 processors are also covered.

1.1 Contributions

This paper continues the line of research on the secure and compact implementations of the Ring-LWE encryption scheme on low-end 8-bit AVR processor. Core contributions are the techniques to prevent information leakage and optimizations to improve real-world performance of Ring-LWE encryption scheme.

In particular, we focused on the optimization of Number Theoretic Transform (NTT) based polynomial multiplication, which is the most expensive computation in the Ring-LWE. In NTT computation, a number of modular arithmetic operations are required and optimization of modular reduction is highly related with performance. To accelerate performance, we use Look Up Table (LUT) based fast reduction techniques for modular coefficient multiplication. Modular addition and subtraction operations are also implemented in constant time and incomplete representation. To optimize the performance in assembly level, NTT routines fully utilize general purpose registers in the target processors.

Based on the above NTT optimization techniques, we present secure and compact implementations of Ring-LWE encryption scheme on an low-end 8-bit AVR processor. All operations are designed to prevent the timing attack. The implementation only requires 681K and $1,754$K clock cycles for 128-bit and 256-bit security level encryption respectively.

The rest of this paper is organized as follows. In section 2, we recall background of Ring-LWE encryption scheme, NTT algorithm, and previous implementation techniques for NTT algorithm. In Section 3, we present optimization techniques for NTT on low-end 8-bit AVR processors. In particular, we propose techniques to prevent information leakage through timing and reduce execution time of NTT algorithm. In Section 4, we report performance of our implementation and compare with the state-of-the-art NTT and Ring-LWE encryption on the low-end 8-bit AVR platforms. Finally, we conclude the paper in Section 5.

2 Background

2.1 Ring-LWE encryption scheme

In 2010, Lyubashevshy et al. proposed an encryption scheme based on a more practical algebraic variant of LWE problem defined over polynomial rings $R_q = \mathbb{Z}_q[\mathbf{x}]/\langle f \rangle$ with an irreducible polynomial $f(x)$ and a modulus q. In Ring-LWE problem, elements a, s and t are polynomials in the ring R_q. Ring-LWE encryption scheme proposed by Lyubashevshy et al. was later optimized in [20]. Roy et al.'s variant aims at reducing the cost of polynomial arithmetic. In particular, the polynomial arithmetic during a decryption operation requires only one Number Theoretic Transform (NTT) operation. Beside this computational optimization, the scheme performs sampling from the discrete Gaussian distribution using a Knuth-Yao sampler. In next subsection, we will first present mathematical concepts of NTT and Knuth-Yao sampling operations, then we will describe the steps used in the Roy et al.'s version of the encryption scheme.

Now, we describe steps applied in the encryption scheme proposed by Roy et al. [20]. We denote the NTT of a polynomial a by $\tilde{a}$.

- Key generation stage **Gen**$(\tilde{a})$: Two error polynomials $r_1, r_2 \in R_q$ are sampled from the discrete Gaussian distribution $\mathcal{X}_\sigma$ by applying the Knuth-Yao sampler twice.
$$\tilde{r_1} = NTT(r_1), \tilde{r_2} = NTT(r_2)$$
and then an operation $\tilde{p} = \tilde{r_1} - \tilde{a} \cdot \tilde{r_2} \in R_q$ is performed. Public key is polynomial pair $(\tilde{a}, \tilde{p})$ and private key is polynomial $\tilde{r_2}$.
- Encryption stage **Enc**$(\tilde{a}, \tilde{p}, M)$: The input message $M \in \{0,1\}^n$ is a binary vector of n bits. This message is first encoded into a polynomial in the ring R_q by multiplying the bits of message by $q/2$. Three error polynomials $e_1, e_2, e_3 \in R_q$ are sampled from $\mathcal{X}_\sigma$. The ciphertext is computed as a set of two polynomials $(\tilde{C_1}, \tilde{C_2})$:
$$(\tilde{C_1}, \tilde{C_2}) = (\tilde{a} \cdot \tilde{e_1} + \tilde{e_2}, \tilde{p} \cdot \tilde{e_1} + NTT(e_3 + M'))$$
- Decryption stage **Dec**$(\tilde{C_1}, \tilde{C_2}, \tilde{r_2})$: One inverse NTT is performed to recover M':
$$M' = INTT(\tilde{r_2} \cdot \tilde{C_1} + \tilde{C_2})$$
and then a decoder is used to recover the original message M from M'.

2.2 Number Theoretic Transform

We use the Number Theoretic Transform (NTT) to perform polynomial multiplication. NTT can be seen as a discrete variant of Fast Fourier Transform (FFT) but performs in a finite ring $\mathbb{Z}_q$. Instead of using the complex roots of unity, NTT evaluates a 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 for $i = 0, \ldots, n-1$, where ω_n denotes a primitive n-th root of unity. Algorithm 1 shows the iterative version of NTT algorithm.

The iterative NTT algorithm consists of three nested loops. The outermost loop (i-loop) starts from $i = 2$ and increases by doubling i, and the loop stops when $i = n$, thus it has only $log_2 n$ iterations. In each iteration, the value of twiddle factor ω_i are computed by executing a power operation $\omega_i = \omega_n^{n/i}$, and the value of ω is initialized by 1. Compared to i-loop, the j-loop executes more iterations, the number of iteration can be seen as a sum of a geometric progression for 2^i where i starts from 0 and has a maximum value of $log_2(n-1)$,

Algorithm 1 Iterative Number Theoretic Transform

```
Require: A polynomial $a(x) \in \mathbb{Z}_q[x]$ of degree $n-1$ and $n$-th primitive $\omega \in \mathbb{Z}_q$ of unity
Ensure: Polynomial $a(x) = NTT(a) \in \mathbb{Z}_q[x]$
 1: $a = BitReverse(a)$
 2: for $i$ from 2 by $i = 2i$ to $n$ do
 3:     $\omega_i = \omega_n^{n/i}$, $\omega = 1$
 4:     for $j$ from 0 by 1 to $i/2 - 1$ do
 5:         for $k$ from 0 by $i$ to $n-1$ do
 6:             $U = a[k+j]$
 7:             $V = \omega \cdot a[k+j+i/2]$
 8:             $a[k+j] = U + V$
 9:             $a[k+j+i/2] = U - V$
10:         $\omega = \omega \cdot \omega_i$
11: return $a$
```

thus, the j-loop has $n-1$ iterations. In each iteration of j-loop, the twiddle factor ω is updated by performing a coefficient modular multiplication. Apparently, the innermost loop (k-loop) occupies most part of the execution time of NTT algorithm since it is executed roughly $\frac{n}{2}log_2 n$ times. In each iteration of the innermost loop, two coefficients $a[i+j]$ and $a[i+j+i/2]$ are loaded from memory into registers, and then $a[i+j+i/2]$ are multiplied by the twiddle factor ω, after that, the value of $a[k+j]$ and $a[k+j+i/2]$ are updated and stored in the memory.

2.3 Previous Implementations of NTT

In LATINCRYPT'15, Pöppelmann et al. optimized the NTT operation by merging inverse NTT and multiplication by powers of ψ^{-1}. Furthermore, bit-reversal step is removed by the manipulation of the standard iterative algorithms. In CHES'15, Liu et al. suggested the high-speed NTT operations with efficient coefficient modular multiplication [11]. They presented the Move-and-Add (MA) method to perform the 16-bit wise coefficient multiplication and the Shift-Add-Multiply-Subtract-Subtract (SAMS2) techniques to replace the expensive reduction operations with the `MUL` instructions by cheaper shift and addition instructions. In TECS'17, Liu et al. improved the modular reduction by using Montgomery reduction [7]. This improves the previous SAMS2 techniques when the case requires a number of shift and addition operations on low-end devices. The new technique ensures the constant time computation together with high performance.

3 Proposed Methods

NTT computation takes up the majority of the execution time on modular multiplication operation since it is performed in the innermost k-loop. The 16-bit wise multiplication requires only 4 8-bit wise multiplication operations and this is already well covered in previous works [11]. Thus, the optimization of fast reduction operation is a perquisite for high-speed implementation of NTT algorithm. We chose the prime modulus $q = 7681$ (i.e. `0x1e01` in hexadecimal representation) and $q = 12289$ (i.e. `0x3001` in hexadecimal representation) for the target parameters, which are used in previous works [11, 7].

Unlike previous SAMS2 method by [11, 7], we propose an optimized Look-Up Table (LUT) based fast reduction technique for performing the $\bmod 7681$ and $\bmod 12289$ operations. The main idea is to first reduce the result by using the 8-bit wise pre-computed reduced results, and then perform the tiny fast reduction steps on short coefficients. The results are kept in the incomplete representation in order to optimize the number of subtraction in the reduction step. For the case of prime modulus $q = 7681$, the variables are always kept in range of $(0, 2^{14} - 1)$ in incomplete representations and the intermediate results (IR) of multiplication are kept in $(0, 2^{28} - 1)$. We set two pre-computed LUTs with (mod 7681) operation. One input variable are ranging from 17-th bit to 24-th bit, which are the values located in $(x \times 2^{16})$ where x is ranging from 0 to 2^8. Afterwards, the variable is reduced to 13-bit wise results through (mod 7681) operation ($\approx ((IR\ div\ 2^{16})\ mod\ 2^8)\ mod\ 7681$). The other input variable is from 25-th bit to 28-th bit, which are values $(x \times 2^{24})$ where x is ranging from 0 to 2^{45}. The LUT ensures that the variable is reduced to the 13-bit results ($\approx (IR\ div\ 2^{24})\ mod\ 7681$). After two times of LUT based reduction operations, the two 13-bit wise outputs are added to the remaining 16-bit wise intermediate results (1-st $\sim$ 16-th bits), which output 17-bit intermediate results. Afterwards, the tiny fast reduction is performed on the intermediate results. Observing that $2^{13} \equiv 2^9 - 1 \bmod 7681$, the fast reduction can be performed with 16-bit wise addition (2^9) and 8-bit wise subtraction operations (-1).

The detailed method is described in Figure 1. We keep the product in four registers $(r3, r2, r1, r0)$, which has been marked by different colors. Each of register $(r3, r2, r1, r0)$ is 8-bit long. The colorful parts mean that this bit has been occupied while the white part means the current bit is empty. The reduction with 7681 using LUT approach can be performed as follows:

1. LUT access. We first perform the LUT access with variable $(r2)$ to get the 13-bit wise reduced results ($s1$ and $s0$). Then, the variable $(r3)$ is also reduced to the results ($t1$ and $t0$). Both results are 13-bit wise long and stored in 2 8-bit registers.
2. Addition. We then perform the addition of $(r1, r0)+(s1, s0)+(t1, t0)$. Apparently, the sum result is less than 18-bit, which can be kept in three registers $(k2, k1, k0)$.
3. Shifting. We right shift $(k2, k1, k0)$ by 13-bit to get the result $(u0)$. Afterwards, the value $(u0)$ is left shifted by 9-bit to get the $(d1, d0)$.

[5] Two LUTs only require 1KB $(2^8 \times 2 + 2^8 \times 2)$ and the LUTs are stored in the ROM. Considering that AVR platforms support ROM size in 128, 256, and 384 KB, the ROM consumption of LUT is negligible.

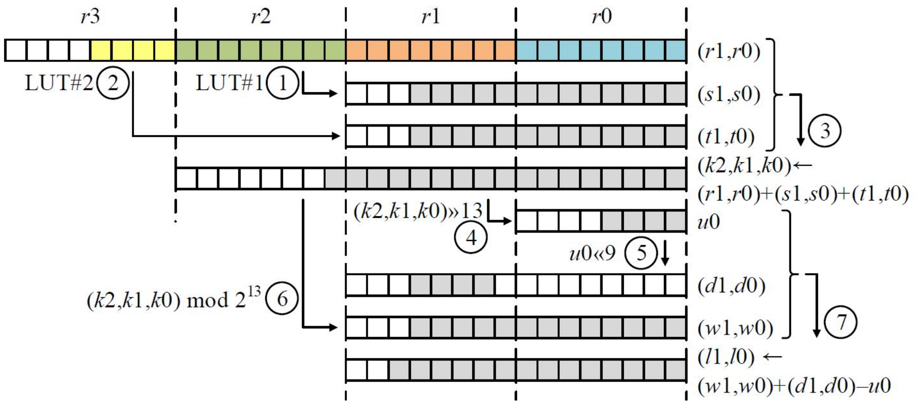

Fig. 1. Look-Up Table based Fast Reduction for $q = 7681$, ①②: LUT access; ③: addition; ④⑤: shifting; ⑥: modulo; ⑦: addition and subtraction.

4. Modulo. Thereafter, the intermediate results $(k2, k1, k0)$ below 13-bit are extracted and we obtain the $(w1, w0)$.
5. Addition and Subtraction. Finally, we perform the addition and subtraction operations of $(w1, w0) + (d1, d0) - u0$.

In Algorithm 2, the LUT based modular reduction in source code level is described. In Step 1∼13, MOV-and-ADD multiplication is used to perform the 16-bit wise multiplication. The 32-bit intermediate results are stored in 4 8-bit registers (`R18, R19, R20, R21`). In Step 14∼15, the address of LUT_1 is loaded to 2 registers (`R30, R31`). Then, the 17∼24-th bits (`R20`) is added to the address. When the address pointer is ready, the LUT access is performed. From Step 22 to 29, the 25∼28-th bits (`R21`) are used to access the LUT_2. Afterwards the results are reduced. In Step 30∼31, two 13-bit LUT results are added. Afterwards, the summation is added to the intermediate results. From Step 35 to 45, tiny fast reduction is performed on 17-bit intermediate results with 16-bit wise addition and 8-bit wise subtraction operations.

Since the LUT approach is generic approach for any primes, proposed LUT based approach is also available in the case of mod 12289. Two differences are LUT value and final step (tiny fast reduction). We need to construct the (mod 12289)'s LUT. For the final step, we perform the tiny fast reduction with modulus equation ($2^{14} \equiv 2^{12} - 1 \bmod 12289$). The detailed descriptions are drawn in Figure 2. We execute two LUT and one tiny final reduction. After the tiny fast reduction, it outputs 16-bit results and this can incur the overflow in following operations. We perform the fast reduction once again to fit the results within 15-bit. By leaving the most significant bit in the register, addition and subtraction operations do not need to check whether the intermediate results generate the overflow/underflow or not.

Algorithm 2 LUT based modular reduction in source code (mod 7681)

```
Input: operands R22, R23, R24, R25
Output: results {R24, R25}
 1: CLR R26            {MOV-and-ADD}
 2: MUL R24, R22
 3: MOVW R18, R0
 4: MUL R25, R23
 5: MOVW R20, R0

 6: MUL R24, R23
 7: ADD R19, R0
 8: ADC R20, R1
 9: ADC R21, R26

10: MUL R25, R22
11: ADD R19, R0
12: ADC R20, R1
13: ADC R21, R26

14: LDI R30, lo8(LUT_1)  {LUT access}
15: LDI R31, hi8(LUT_1)
16: ADD R30, R20
17: ADC R31, R26
18: ADD R30, R20
19: ADC R31, R26

20: LPM R22, Z+
21: LPM R23, Z+

22: LDI R30, lo8(LUT_2)  {LUT access}
23: LDI R31, hi8(LUT_2)
24: ADD R30, R21
25: ADC R31, R26
26: ADD R30, R21
27: ADC R31, R26

28: LPM R24, Z+
29: LPM R25, Z+

30: ADD R24, R22
31: ADC R25, R23

32: ADD R24, R18
33: ADC R25, R19
34: ADC R26, R26

35: MOV R20, R25{tiny fast reduction}
36: ANDI R25, 0X1F

37: LSR R26
38: ROR R20
39: SWAP R20
40: ANDI R20, 0X0F

41: SUB R24, R20
42: SBC R25, R26

43: LSL R20
44: ADD R25, R20
45: CLR R1
```

Constant Modular Addition and Subtraction To prevent timing attacks, modular addition and subtraction operations should be implemented in constant time. We used the incomplete representation and unsigned type for variable format. The results are always kept in 2 bytes and positive values. The detailed descriptions are available in Algorithm 3. First addition or subtraction operation is performed. In particular, subtraction operation is performed with addition of variable (q_2) to avoid underflow condition. From Step 6 to 9, the tiny fast reduction operation is performed. However, the result we get in Step 9 may still be larger than modulus ($q = 7681$), thus, we do the correction by subtracting the modulus (q). If the underflow condition occurs, we perform the addition with modulus (q) with the mask variable (P). Finally, the results (R) are always kept within 0x2000 in the incomplete representation.

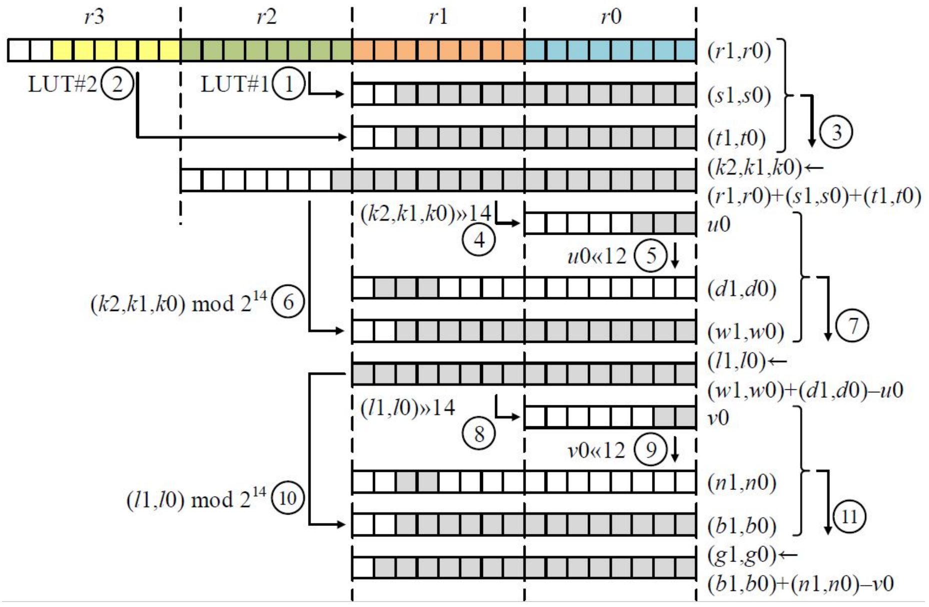

Fig. 2. Look-Up Table based Fast Reduction for $q = 12289$, ①②: LUT access; ③: addition; ④⑤: shifting; ⑥: modulo; ⑦: addition and subtraction; ⑧⑨: shifting; ⑩: modulo; ⑪: addition and subtraction.

Algorithm 3 Constant modular addition/subtraction for $q = 7681$ (`0x1E01`)

```
Require: Two 2-word operands $A$ and $B$ in $[0, 2^{14} - 1]$.
Ensure: The incomplete result $R = A(+,-)B$ mod 0x2000 $[0, 2^{14} - 1]$ or 0x1E01.
 1: if addition then
 2:     $R \leftarrow A + B$                                   {addition operation}
 3: else if subtraction then
 4:     $q_2 \leftarrow q \ll 2$                               {subtraction operation}
 5:     $R \leftarrow q_2 + A - B$                              {underflow prevention}
 6: $R_1 \leftarrow R \gg 13$                                   {tiny fast reduction}
 7: $R_2 \leftarrow (R \gg 4)\&0x1E00$
 8: $R \leftarrow R\&0x1FFF$
 9: $R \leftarrow R - R_1 + R_2$
10: if complete then
11:     $\{Borrow, R\} \leftarrow R - 0x1E01$                  {last correction}
12:     $P \leftarrow 0x0000 - Borrow$
13:     $R \leftarrow R + (0x1E01\&P)$
14: return $R$
```

Algorithm 4 Constant modular addition/subtraction for $q = 12289$ (`0x3001`)

```
Require: Two 2-word operands $A$ and $B$ in $[0, 2^{15} - 1]$.
Ensure: The incomplete or complete result $R = A(+,-)B$ mod 0x4000 $\in [0, 2^{15} - 1]$
    or 0x3001.
 1: if addition then
 2:     $R \leftarrow A + B$                          {addition operation}
 3: else if subtraction then
 4:     $q_2 \leftarrow q \ll 2$                      {subtraction operation}
 5:     $R \leftarrow q_2 + A - B$                    {underflow prevention}
 6: $R_1 \leftarrow R \gg 14$                         {tiny fast reduction}
 7: $R_2 \leftarrow (R \gg 2) \& 0x7000$
 8: $R \leftarrow R \& 0x3FFF$
 9: $R \leftarrow R - R_1 + R_2$
10: if complete then
11:     $\{Borrow, R\} \leftarrow R - 0x3001$         {last correction}
12:     $P \leftarrow 0x0000 - Borrow$
13:     $R \leftarrow R + (0x3001 \& P)$
14: return $R$
```

For the case of 12289, we can adopt the constant modular addition and subtraction techniques in Algorithm 3. Only the parameters are different. The detailed descriptions are given in Algorithm 4. Firstly, the addition and subtraction operations are performed. Afterwards, the fast reduction is performed. The obtained results (R) are always kept within `0x4000` in the incomplete representation.

4 Performance Evaluation

This section presents performance results of our implementation. We first give the experimental platform in section 4.1. Afterwards, we show a comparison with the previous modular multiplication and NTT implementations in section 4.2. Finally, we show a comparison with the previous Ring-LWE implementation in section 4.3.

4.1 Experimental platform

Our implementation uses ATxmega128A1 processor on an Xplain board as target platform. This processor has a maximum frequency of 32 MHz, 128 KB flash program memory, and 8 KB SRAM. It supports an AES crypto-accelerator and can be used in a wide range of applications, such as industrial, hand-held battery applications as well as some medical devices. The implementation is written using a mixed ANSI C and Assembly languages. In particular, the main structure and interface are written in C while the core operations such as modular arithmetic is implemented in Assembly. For the LUT based approach, the constant LUT variables are stored in flash program memory, which requires 0.5KB

Table 1. Execution time of modular multiplication and NTT (in clock cycles), where 128-bit security represents ($n : 256, q : 7681$) and 256-bit security represents ($n : 512, q : 12289$) on 8-bit AVR processors, e.g., ATxmega64, ATxmega128.

Implementation	128-bit security			256-bit security		
	`MOD MUL`	`NTT`	`Const`	`Mod MUL`	`NTT`	`Const`
Boorghany et al. [2]	N/A	1,216,000	–	N/A	2,207,787	–
Boorghany et al. [1]	N/A	754,668	–	N/A	N/A	–
Pöppelmann et al. [16]	N/A	334,646	–	N/A	855,595	–
Liu et al. [11]	N/A	193,731	–	N/A	441,572	–
Liu et al. [7]	73	194,145	$\surd$	70	516,971	$\surd$
This work	57	158,607	$\surd$	66	403,224	$\surd$

for saving the parameters and 3 clock cycles for each byte access. We complied our implementation with speed optimization option '-O3' on Atmel Studio 6.2. In order to obtain accurate timing, we execute each operation for at least 1000 times and report average cycle count for each operation.

4.2 Comparison of modular multiplication and NTT

Table 1 summarizes execution time of modular multiplication and NTT for both of medium-term and long-term security levels. First, various works including [2, 1, 16, 11] are not constant-time solutions, which means the attackers can perform timing attack to extract the secret information. Recent work by Liu et al. introduced the secure approach with tiny Montgomery reduction [7]. They perform the Montgomery reduction to reduce the 28/30-bit variables to 14/15-bit results. However, the complexity of n-word Montgomery reduction is generally $n^2 + n$, which is still high overheads on the low-end devices. Unlike previous approaches, we used LUT based approach to achieve high performance and secure implementation.

As shown in the Table 1, the proposed modular multiplication with 7681 and 12289 only requires 57 and 66 clock cycles, which are 16 and 4 clock cycles smaller than previous approaches, respectively [7]. The proposed NTT operation also shows higher performance than previous works. NTT operation only requires $158,607$ clock cycles for 128-bit security implementation and $403,224$ cycles for 256-bit security implementation. Results of NTT for medium and long-term security are 18.3% and 22.0% faster than previous works, respectively.

4.3 Comparison of Ring-LWE

With optimized NTT implementation, we evaluated the Ring-LWE encryption scheme with parameter sets (n, q, σ) with $(256, 7681, 11.31/\sqrt{2\pi})$ and (512,12289, $12.18/\sqrt{2\pi}$) for security levels of 128-bit and 256-bit. The tailcut of discrete Gaussian sampler is limited to 12σ to achieve a high precision statistical difference from the theoretical distribution, which is less than 2^{-90}. These parameter

Table 2. Performance comparison of software implementation of 128-bit and 256-bit security level lattice-based cryptosystems on 8-bit AVR processors, e.g., ATxmega64, ATxmega128.

Implementation	`NTT/FFT`	`Sampling`	`Enc`	`Secure`
Implementations of 128-bit security level				
Boorghany et al. [2]	1,216,000	N/A	5,024,000	–
Boorghany et al. [1]	754,668	N/A	3,042,675	–
Pöppelmann et al. [16]	334,646	N/A	1,314,977	–
Liu et al. [11]	193,731	26,763	671,628	–
Liu et al. [7]	194,145	53,023	796,872	$\surd$
This work	158,607	35,409	680,796	$\surd$
Implementations of 256-bit security level				
Boorghany et al. [1]	2,207,787	617,600	N/A	–
Pöppelmann et al. [16]	855,595	N/A	3,279,142	–
Liu et al. [11]	441,572	255,218	2,617,459	–
Liu et al. [7]	516,971	105,153	1,975,806	$\surd$
This work	403,224	69,062	1,754,064	$\surd$

sets were also used in most of the previous software implementations, e.g., [1–3, 11, 7].

Discrete Gaussian sampling is an integral part of Ring-LWE algorithm. However, previous implementations are not secure against timing and simple power analysis, since the Knuth-Yao sampler uses a bit/byte scanning operation in which the sample generated is related to the number of probability-bits/bytes scanned during a sampling operation and its timing provides secret information to an adversary about the value of the sample. In [19], Roy et al. suggested a random shuffling method to protect the Gaussian distributed polynomial against such attacks. The random permutation is performed after generating all samples. The random shuffle operation swaps all samples randomly, which removes any timing information from samplings. In the implementation, we adopt the previous Knuth-Yao sampler with byte-scanning [19, 11]. Afterwards, all generated samples are randomly mixed with the random numbers.

Table 2 compares software implementations of 128-bit and 256-bit security lattice-based cryptosystems on the 8-bit AVR processors. We compare the previous work [1, 2, 16, 11, 7] with ours. Proposed 128-bit security implementation requires 159K, 35K, and 681K cycles for NTT, sampling and encryption, respectively. Compared to the recent work [7], the NTT operation is significantly improved because we used compact modular multiplication routine. For the secure sampling, we adopted lightweight random shuffling technique, which shows better performance than previous works. The proposed implementations are constant timing, which ensures a secure computation against simple power analysis and timing attacks. The similar performance enhancement is observed in 256-bit case.

5 Conclusion

This paper presents optimization techniques for efficient and secure implementation of NTT and its application Ring-LWE encryption on the low-end 8-bit AVR platform. For the secure KY sampler, we use the random shuffling technique to prevent the side channel attack. A combination of both NTT and KY sampler implementation achieved new speed records for secure 128-bit and 256-bit Ring-LWE encryption implementation on low-end 8-bit AVR platforms.

Our future works are applying the proposed techniques to the other low-end IoT devices, such as 8-bit PIC and 16-bit MSP processors. Similarly, these platforms also support very limited Arithmetic Logic Unit (ALU) and memory consumptions. Second, we will further investigate side channel attacks on the implementation of Ring-LWE. Unlike traditional RSA and ECC, only few works explored potential threats on the implementation of Ring-LWE.

References

1. S. B. S. Ahmad Boorghany and R. Jalili. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. Cryptology ePrint Archive, Report 2014/514, 2014. `https://eprint.iacr.org/2014/514.pdf`.
2. A. Boorghany and R. Jalili. Implementation and Comparison of Lattice-based Identification Protocols on Smart Cards and Microcontrollers. Cryptology ePrint Archive, Report 2014/078, 2014.
3. R. De Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient Software Implementation of Ring-LWE Encryption. *18th Design, Automation & Test in Europe Conference & Exhibition–DATE*, 2015.
4. Z. Liu, R. Azarderakhsh, H. Kim, and H. Seo. Efficient software implementation of Ring-LWE encryption on IoT processors. *IEEE Transactions on Computers*, 2017.
5. Z. Liu, X. Huang, Z. Hu, M. K. Khan, H. Seo, and L. Zhou. On emerging family of elliptic curves to secure internet of things: ECC comes of age. *IEEE Transactions on Dependable and Secure Computing*, 14(3):237–248, 2017.
6. Z. Liu, P. Longa, G. Pereira, O. Reparaz, and H. Seo. Fourq on embedded devices with strong countermeasures against side-channel attacks. Technical report, Cryptology ePrint Archive, Report 2017/434, 2017. 28, 29.
7. Z. Liu, T. Pöppelmann, T. Oder, H. Seo, S. S. Roy, T. Güneysu, J. Großschädl, H. Kim, and I. Verbauwhede. High-performance ideal lattice-based cryptography on 8-bit AVR microcontrollers. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(4):117, 2017.
8. Z. Liu, H. Seo, J. Großschädl, and H. Kim. Efficient implementation of NIST-compliant elliptic curve cryptography for sensor nodes. In *International Conference on Information and Communications Security*, pages 302–317. Springer, 2013.
9. Z. Liu, H. Seo, J. Großschädl, and H. Kim. Efficient implementation of NIST-compliant elliptic curve cryptography for 8-bit AVR-based sensor nodes. *IEEE Transactions on Information Forensics and Security*, 11(7):1385–1397, 2016.
10. Z. Liu, H. Seo, Z. Hu, X. Hunag, and J. Großschädl. Efficient implementation of ECDH key exchange for MSP430-based wireless sensor networks. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 145–153. ACM, 2015.

11. Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, and I. Verbauwhede. Efficient Ring-LWE encryption on 8-bit AVR processors. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 663–682. Springer, 2015.
12. Z. Liu, H. Seo, and Q. Xu. Performance evaluation of twisted edwards-form elliptic curve cryptography for wireless sensor nodes. *Security and Communication Networks*, 8(18):3301–3310, 2015.
13. Z. Liu, J. Weng, Z. Hu, and H. Seo. Efficient elliptic curve cryptography for embedded devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(2):53, 2016.
14. V. Lyubashevsky, C. Peikert, and O. Regev. On Ideal Lattices and Learning with Errors Over Rings. Cryptology ePrint Archive, Report 2012/230, 2012.
15. T. Oder, T. Pöppelmann, and T. Güneysu. Beyond ECDSA and RSA: Lattice-based Digital Signatures on Constrained Devices. *51st Annual Design Automation Conference–DAC*, 2014.
16. T. Pöppelmann, T. Oder, and T. Güneysu. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In *International Conference on Cryptology and Information Security in Latin America*, pages 346–365. Springer, 2015.
17. L. Qiu, Z. Liu, G. C. Pereira, and H. Seo. Implementing RSA for sensor nodes in smart cities. *Personal and Ubiquitous Computing*, 21(5):807–813, 2017.
18. O. Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, 2005.
19. S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Compact and side channel secure discrete gaussian sampling. 2014.
20. S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact Ring-LWE Cryptoprocessor. In *Cryptographic Hardware and Embedded Systems C CHES 2014*, volume 8731, pages 371–391. 2014.
21. H. Seo. Faster (feat. ECC PMULL) over F2571. In *A Systems Approach to Cyber Security: Proceedings of the 2nd Singapore Cyber-Security R&D Conference (SG-CRC 2017)*, volume 15, page 97. IOS Press, 2017.
22. H. Seo and H. Kim. MoTE-ECC based encryption on MSP430. *Journal of information and communication convergence engineering*, 15(3):160–164, 2017.
23. H. Seo, Z. Liu, J. Großschädl, and H. Kim. Efficient arithmetic on ARM-NEON and its application for high-speed RSA implementation. *Security and Communication Networks*, 9(18):5401–5411, 2016.
24. H. Seo, Z. Liu, Y. Nogami, T. Park, J. Choi, L. Zhou, and H. Kim. Faster ECC over $\mathbb{F}_{2^{521}-1}$(feat. NEON). In *International Conference on Information Security and Cryptology*, pages 169–181. Springer, 2015.
25. P. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134, Nov 1994.