Ring-LWE on 8-bit AVR Embedded Processor

Hwajeong Seo*, Hyeokdong Kwon, Yongbeen Kwon, Kyungho Kim, Seungju Choi, Hyunjun Kim, and Kyoungbae Jang

IT Department, Hansung University, Seoul, South Korea, {hwajeong84, korlethean, vexyoung, pgm.kkh, bookingstore3, khj930704, starj1023}@gmail.com

Abstract. Fast implementation of Ring-LWE is a challenge for the low-end embedded processors. One of the most expensive operation for Ring-LWE is Number Theoretic Transform (NTT). Many works have investigated the optimized implementation for the NTT operation. In this paper, we further optimized the NTT operation on the low-end 8-bit AVR microcontrollers. We focused on the optimized and secure polynomial multiplication to ensure countermeasures against timing attacks and high performance. In particular, we propose the combined Look-Up Table (LUT) based fast reduction techniques in regular fashion. With the optimization techniques, the proposed NTT implementation enhances the performance by 14.6% than previous best results. Finally, proposed NTT implementations are applied to the Ring-LWE key scheduling and encryption operations, which require the only 1,325,171 and 1,430,601 clock cycles for 256-bit security levels.

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

1 Introduction

The hard problem of traditional public key cryptography algorithms, such as RSA and Elliptic Curve Cryptography (ECC), are based on integer factorization and discrete logarithm problems, which are believed to be secure against classical attacks on traditional computers. For this reason, previous works focused on efficient implementations of RSA and ECC cryptography systems [18, 23, 6, 22, 14, 24, 10, 13, 25, 11, 9, 7]. However, such hard problems can be solved by using Shor's algorithm in polynomial time when a sufficient large quantum computer is ready [27]. In order to avoid the potential quantum attacks, the lattice-based cryptography is considered as one of the most promising candidates for post-quantum cryptography. The lattice-based cryptography is built based on worst-case computational assumptions in lattices that would remain hard even for quantum algorithms.

Furthermore, the future computing platforms, such as Internet of Things (IoT), are widely deployed and used. The low-end IoT devices with many sensors handle important sensor data. For this reason, secure cryptographic algorithms

* Corresponding Author

should be implemented on the low-end IoT devices. However, the low-end IoT devices are very resource-constrained, in terms of computing power, energy, and storage. This hard condition introduces a challenge to implement the cryptography algorithm on low-end devices. In this paper, we present the most optimal implementation of NTT computation for lattice based cryptography. The implementation also ensures the constant timing, which is secure against the timing attacks.

The introduction of Learning with Errors (LWE) problem and its ring variant (Ring-LWE) [19, 15] 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 [16]. De Clercq et al. implemented Ring-LWE encryption scheme on the identical ARM processors [5]. They utilized 32-bit registers to retain two $13 \sim 14$ coefficients at once. Boorghany et al. implemented a lattice-based cryptographic scheme on an 8-bit processor for the first time in [1, 3]. 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 optimal way. In LATINCRYPT'15, Pöppelmann et al. studied and compared implementations of Ring-LWE encryption and BLISS on an 8-bit Atmel ATxmega128 microcontroller [17]. 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 [12]. 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 [8]. 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. In ICISC'17, Seo et al. proposed secure and efficient Ring-LWE implementations by using LUT based modular reduction technique and random shuffling [26].

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

In particular, we focused on the optimization of Number Theoretic Transform (NTT) based polynomial multiplication. In NTT computation, a number of modular arithmetic operations are required and the optimization of modular arithmetic is highly related with performance. In order to improve the performance of modular reduction, we used the combined Look Up Table (LUT) tech-

niques for modular multiplication. This efficiently performs all reduction with memory accesses.

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 key scheduling and encryption implementations require 1,325 K and 1,430 K clock cycles for 256-bit security level, respectively.

The rest of this paper is organized as follows. In section 2, we introduce 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, the proposed method ensures the constant timing and reduces the execution time of NTT algorithm. In Section 4, we report performance of our implementation and compare with the state-of-the-art implementations of NTT, key scheduling, and 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 [21]. 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. [21]. 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

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 $a(x) = NTT(a) \in \mathbb{Z}_q[x]$

```
a = BitReverse(a)
for i from 2 by i = 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
            U = a[k + j]
            V = ω · a[k + j + i/2]
            a[k + j] = U + V
            a[k + j + i/2] = U − V
        ω = ω · ω_i
return a
```

$e_1, e_2, e_3 \in R_q$ 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 $\mathbf{Dec}(\tilde{C_1}, \tilde{C_2}, \tilde{r_2})$: 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 $log_2(n-1)$, 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 [12]. 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 [8]. 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. In ICISC'17, the optimized Look-Up Table (LUT) based fast reduction technique is proposed [26]. 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.

3 Proposed Methods

NTT computation uses the majority of the execution time on modular multiplication operation since it is performed in the innermost k-loop of NTT computation. The target 16-bit wise multiplication can be efficiently performed by using previous works [12]. In this paper, we focused on the optimization of fast reduction operations for NTT computation. We chose the prime modulus $q = 12289$ (i.e. `0x3001` in hexadecimal representation) for the target parameters, which are used in previous works [12, 8, 26].

The modular reduction can be implemented using the bit-shift and add technique (i.e. SAMS2) covered in previous works [12, 8], This approach can be accelerated by using the optimized Look-Up Table (LUT) based fast reduction technique for performing the mod 12289 operations [26]. 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.

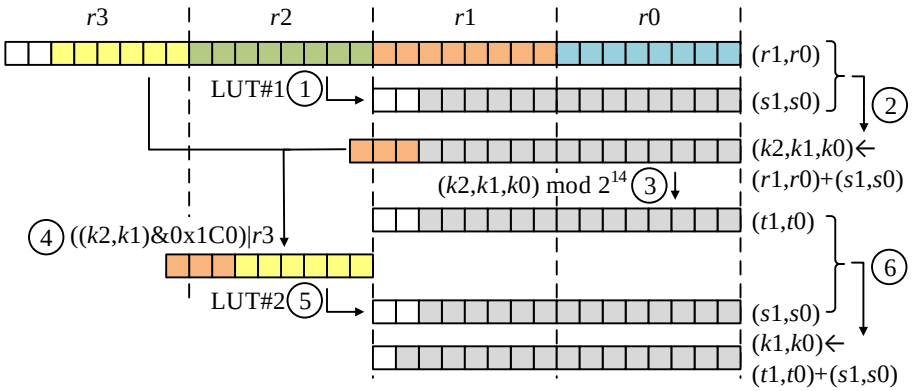

Fig. 1: Look-Up Table based Fast Reduction for $q = 12289$, ①: LUT access; ②: addition; ③: modulo; ④: concatenation; ⑤: LUT access; ⑥: addition.

In this paper, we further optimized the LUT based approach by using combined LUT techniques. For the case of prime modulus $q = 12289$, the variables are always kept in range of $(0, 2^{15} - 1)$ in incomplete representations and the intermediate results (IR) of multiplication are kept in $(0, 2^{30} - 1)$. We set two pre-computed LUTs with (mod 12289) 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. Afterward, the variable is reduced to 14-bit wise results through (mod 12289) operation ($\approx ((IR \ div \ 2^{16}) \ mod \ 2^8) \ mod \ 12289$). The reduced results are added to the intermediate results (i.e. r_1, r_0). The addition of 14-bit results and 16-bit results generates 17-bit intermediate results. With this intermediate results, the other LUT received the combined two input variables (one from 15-th bit to 17th bit and the other one from 25-th bit to 30-th bit)[1]. The LUT ensures that the variable is reduced to the 14-bit results ($\approx (IR \ div \ 2^{14}) \ mod \ 12289$).

After the second LUT based reduction operations, the 14-bit wise results are added to the remaining 14-bit wise intermediate results (1-st bit $\sim$ 14-th bits), which output 15-bit intermediate results. In previous works, they utilize the tiny fast reduction in the last step. The proposed approach only requires LUT based reduction without tiny fast reduction. This ensures the optimized performance than previous works.

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

[1] Two LUTs only require 1.5KB ($2^8 \times 2 + 2^9 \times 2$) and the LUTs are stored in the FLASH memory. Considering that 8-bit AVR platforms support to write data into the FLASH memory and its size is ranging from 128$\sim$384 KB. The storage for LUTs is negligible on the target processors.

Algorithm 2 LUT based modular reduction in source code (mod 12289)

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: MOV R30, R20
15: LDI R31, hi8(LUT1_L) {LUT access}
16: LPM R22, Z
17: LDI R31, hi8(LUT1_H)
18: LPM R23, Z
19: ADD R18, R22
20: ADC R19, R23
21: ADC R20, R20    {Register re-use}
22: MOV R30, R19
23: ANDI R19, 0X3F
24: ANDI R20, 0X01
25: ANDI R30, 0XC0
26: ADD R30, R21
27: LDI R31, hi8(LUT2_L) {LUT access}
28: ADD R31, R20
29: LPM R24, Z
30: LDI R31, hi8(LUT2_H)
31: ADD R31, R20
32: LPM R25, Z
33: ADD R24, R18
34: ADC R25, R19
35: CLR R1
```

occupied while the white part means the current bit is empty. The reduction with 12289 using LUT approach can be performed as follows:

1. LUT access. We first perform the LUT access with variable ($r2$) to get the 14-bit wise reduced results ($s1$ and $s0$).
2. Addition. The reduced results ($s1, s0$) is added to the intermediate results ($r1, r0$). This addition generates 17-bit intermediate results.
3. Modulo. The intermediate results ($k2, k1, k0$) below 14-bit are extracted and we obtain the ($t1, t0$).
4. Concatenation. The intermediate result ($r3$) and the other 3-bit intermediate result (i.e. $(k2, k1)\&0x1C0$) are concatenated and generate 9-bit wise value.
5. LUT access. We perform the LUT access with the concatenated value to get the 14-bit wise reduced results ($s1$ and $s0$).
6. Addition. Finally, we perform the addition operation of $(t1, t0) + (s1, s0)$ to get final 15-bit wise results.

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). Afterward, we perform the LUT based reduction operation. The LUT input and output are 8-bit and 16-bit, respectively.

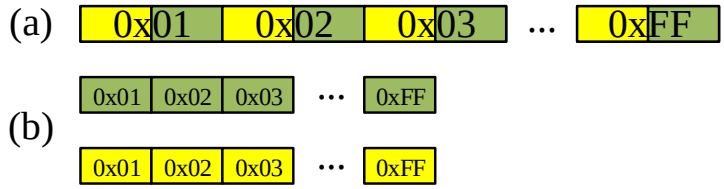

Fig. 2: Comparison of LUT construction, (a) previous method, (b) proposed separated memory access. Yellow and green blocks represent higher and lower parts for LUT, respectively.

In order to accelerate the memory access, we used two different optimized technique. First method is aligned memory access. We set the higher 8-bit address with one operand and only update the lower byte with different values. The detailed descriptions are as follows:

- 8-bit aligned (init): `MOV R30, R24` → `LDI R31, hi8(LUT)` → `LPM R22, Z`
- 8-bit aligned (second): `MOV R30, R24` → `LPM R22, Z`

  where `Z`, `R24`, `R25`, `R1`, `R30`, `R31`, and `R26` are `Z` pointer, first input value, second input value, zero value, lower part of memory address, higher part of memory address and result, respectively.

Second method is separated memory access. The LUT outputs 16-bit wise results, which requires doubled offsets. In this setting, the aligned memory access is not feasible. We separated the LUT into two parts. First one is for lower 8-bit and second one is for higher 8-bit. The detailed method is described in Figure 2. Unlike previous LUT construction, the separated LUTs are constructed. Under this LUT setting, the aligned memory access is efficiently performed. The technique is also applied to the following LUT accesses.

In Step 14∼15, the 17∼24-th bits (`R20`) is loaded to the lower 8-bit address (`R30`). Then, the higher 8-bit address of LUT1_L is loaded to the register (`R31`). In Step 16, FLASH memory access is performed with `LPM` instruction. From Step 17 and 18, higher part of LUT1 is accessed. Since we used the aligned memory access, only higher address is modified.

In Step 19∼21, the reduced results and intermediate results are added. In particular, we re-used the register (`R20`). The addition with carry can store the carry bit generated from Step 20.

Thereafter, in Step 22∼25, two intermediate results are concatenated. In Step 26∼32, LUT2 access is performed in aligned memory access method. Finally, the reduced results and intermediate results are added together. This ensures the 15-bit wise intermediate results.

The proposed modular reduction method is generic approach for any primes for Ring-LWE. For this reason, we can easily extend the proposed method to other primes without difficulty. Definitely, the proposed method is working for

lattice based PQC candidates, such as NewHope and CRYPSTALS-KYBER [4, 2]

Discrete Gaussian Sampling Discrete Gaussian sampling is an integral part of Ring-LWE algorithm. For fast computation, we adopted the Knuth-Yao sampler with byte-scanning [20, 12]. However, original sampling is not secure against timing attack and simple power analysis. In order to ensure the secure implementation, we adopted the random shuffling after random sampling [20]. The random shuffling removes the relation between random samples and timing information.

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 main structure and interface are written in C language while the core operations such as modular arithmetic is implemented in Assembly language. For the LUT based approach, the constant LUT variables are stored in flash program memory, which requires 1.5KB 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 long-term security levels. First, various works including [3, 1, 17, 12] are not constant-time solutions, which means the attackers can perform timing attack to extract the secret information. Previous work by Liu et al. introduced the secure approach with tiny Montgomery reduction [8]. 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. In ICISC'17, Seo et al. suggested LUT based approach to achieve high performance and constant timing [26]. However, we find that there is still room to improve the performance.

As shown in the Table 1, the proposed modular multiplication with 12289 only requires 47 clock cycles, which are 19 clock cycles smaller than previous approaches [26]. Definitely, the proposed NTT operation also shows higher performance than previous works. NTT operation only requires $344,288$ cycles for 256-bit security implementation. Results of NTT for long-term security is 14.6% faster than previous works.

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

Implementation	`Mod MUL`	`NTT`	`Const`
Boorghany et al. [3]	N/A	2,207,787	–
Boorghany et al. [1]	N/A	N/A	–
Pöppelmann et al. [17]	N/A	855,595	–
Liu et al. [12]	N/A	441,572	–
Liu et al. [8]	70	516,971	√
Seo et al. [26]	66	403,224	√
This work	47	344,288	√

4.3 Comparison of Ring-LWE

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

Implementation	`NTT/FFT`	`Key-Gen`	`Enc`	`Secure`
Boorghany et al. [1]	2,207,787	N/A	N/A	–
Pöppelmann et al. [17]	855,595	N/A	3,279,142	–
Liu et al. [12]	441,572	2,165,239	2,617,459	–
Liu et al. [8]	516,971	N/A	1,975,806	√
Seo et al. [26]	403,224	N/A	1,754,064	√
This work	344,288	1,325,171	1,430,601	√

With optimized NTT implementation, we evaluated the Ring-LWE encryption scheme with parameter sets (n, q, σ) with (512,12289, $12.18/\sqrt{2\pi}$) for security levels of 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 sets were also used in most of the previous software implementations, e.g., [1, 3, 5, 12, 8, 26].

Table 2 compares software implementations of 256-bit security lattice-based cryptosystems on the 8-bit AVR processors. We compare the previous work [1, 3, 17, 12, 8, 26] with ours. Proposed 256-bit security implementation requires 344K, 1,325K, and 1,430K cycles for NTT, key generation, and encryption, respectively. Compared to the recent work [26], the NTT operation is significantly improved because we used compact modular multiplication routine. The performance improvements of NTT accelerate the key generation and encryption for Ring-LWE implementations. The proposed encryption implementation outperforms previous works by 18.4%. Furthermore, the proposed implementations are constant timing, which ensures a secure computation against simple power analysis and timing attacks.

5 Conclusion

This paper presents optimization techniques for efficient and secure implementation of NTT and its application Ring-LWE key generation and encryption on the low-end 8-bit AVR platform. The proposed NTT implementation achieved new speed records for secure 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 storage. Second, we will further investigate the side channel attacks for Ring-LWE scheme on the low-end embedded processors.

6 Acknowledgement

This work was supported as part of Military Crypto Research Center(UD170109ED) funded by Defense Acquisition Program Administration(DAPA) and Agency for Defense Development(ADD).

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. E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, P. S. T. Pöppelmann, and D. Stebila. Newhope. Technical report, Technical report, National Institute of Standards and Technology, 2017 ....
3. 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.
4. J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. V. Lyubashevsky, C. Peikert, and O. Regev. On Ideal Lattices and Learning with Errors Over Rings. Cryptology ePrint Archive, Report 2012/230, 2012.
16. 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.
17. 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.
18. 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.
19. 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.
20. S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Compact and side channel secure discrete gaussian sampling. 2014.
21. 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.

22. 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.
23. H. Seo and H. Kim. MoTE-ECC based encryption on MSP430. *Journal of information and communication convergence engineering*, 15(3):160–164, 2017.
24. 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.
25. 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.
26. H. Seo, Z. Liu, T. Park, H. Kwon, S. Lee, and H. Kim. Secure number theoretic transform and speed record for Ring-LWE encryption on embedded processors. In *International Conference on Information Security and Cryptology*, pages 175–188. Springer, 2017.
27. 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.