IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 22, 2019, accepted July 9, 2019, date of publication July 16, 2019, date of current version August 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2929299

Fast and Power-Analysis Resistant Ring Lizard Crypto-Processor Based on the Sparse Ternary Property

PILJOO CHOI[1], **(Member, IEEE), JI-HOON KIM**[2], **(Senior Member, IEEE), AND DONG KYUE KIM**[3], **(Member, IEEE)**
[1]Software Education Committee, Hanyang University, Seoul 04763, South Korea
[2]Department of Electronic and Electrical Engineering, Ewha Womans University, Seoul 04763, South Korea
[3]Department of Electronic Engineering, Hanyang University, Seoul 04763, South Korea

Corresponding author: Ji-Hoon Kim (jihoonkim@ewha.ac.kr)

ABSTRACT Ring Lizard (RLizard) is a quantum-resistant public-key cryptosystem based on the ideal lattice. RLizard uses a sparse ternary polynomial, which facilitates implementation with lower complexity. The Lizard scheme's proposal for the National Institute of Standards and Technology's post-quantum cryptography standardization included its reference hardware design using the sparse ternary property; however, in this paper, we present the RLizard crypto-processor with the improved processing speed and security level against power analysis attacks. By additionally utilizing unused values for each memory access in the conventional RLizard crypto-processor, the processing speed of the proposed RLizard crypto-processors can increase by a factor of two or up to four times. The implementation results with three different FPGA devices show that the area overhead is approximately 50–100 flip-flops (FFs) and 50–300 lookup tables (LUTs), occupying approximately 2%–3% of the total area. The vulnerability to power analysis attacks and the proposed countermeasures were also analyzed. The experimental results prove the vulnerability of unprotected implementation, and the implementation results show that the masking and hiding countermeasures additionally require approximately 50–120 FFs and 100–360 LUTs. In addition, our idea can be applied to other ideal-lattice-based cryptosystems using a sparse binary or ternary polynomial, such as NTRU and Round5.

INDEX TERMS Coprocessors, digital circuits, field programmable gate arrays, side-channel attacks, post-quantum cryptography.

I. INTRODUCTION

Modern public-key cryptosystems, such as RSA [1] and elliptic curve cryptography (ECC) [2], [3], are based on factoring and discrete logarithm problems, but both problems could be more easily solved on a quantum computer by Shor's algorithm [4]. As alternatives to these problems, lattice problems, such as a learning with errors (LWE) problem [5] and a learning with rounding (LWR) problem [6], have been known to be quantum-resistant and are attracting considerable attention as new digital signature [7] and encryption [8] methods. Although cryptosystems based on the standard lattice, which operates over matrices, require large sizes of keys, the key size can be reduced by using the ideal lattice [9], which operates over polynomial rings.

The dominant operation of the ideal-lattice-based cryptosystems is a *convolution*, which is the multiplication of two polynomials over a polynomial ring. For fast convolution, a number theoretic transform (NTT) is widely used [10]–[13], whereas some cryptosystems [14]–[20] use sparse binary or ternary polynomials to lower the computational complexity of a convolution. Binary or ternary polynomials have coefficients $\in \{0, 1\}$ or $\{0, 1, -1\}$, and "sparse" means most parts of the coefficients are zeros. Hence, a large part of the computations with zero coefficients can be removed, and multiplications with ± 1 can be simplified to additions and subtractions.

From the proposals to the National Institute of Standards and Technology's (NIST's) post-quantum cryptography

The associate editor coordinating the review of this manuscript and approving it for publication was Fan Zhang.

standardization [21], we could find following some ideal-lattice-based cryptosystems using a sparse binary or ternary polynomials: NTRU [15], [16], Round5 [17], and ring Lizard (RLizard) [18]–[20]. These cryptosystems can be implemented with a simple structure and fast processing speed by using the sparse binary or ternary property. However, to our knowledge, there are only a few studies on their hardware implementation. The NTRU crypto-processor in [22] calculates all the coefficients of convolution product in parallel. This parallel style requires many resources, and the ternary property is not used. In [23], the sparse ternary property is also not used, so n^2 multiplications are required, where n is the order of the polynomial. In [24], the sparse ternary property seems to be used, but the detailed hardware structure is not explained. These three works regard the hardware implementation of the classic NTRU, and the proposals [15]–[17] of the NTRU and Round5 to the NIST's post-quantum cryptography standardization included only their software implementation. As for RLizard, the work in [19] included only its software implementation. The proposal of the Lizard scheme [20] included even the Verilog source code for its hardware implementation that uses sparse ternary property; however, its RLizard crypto-processor can operate at only a single basic processing speed.

Not only due to the vulnerability of the cryptographic scheme itself, the secret key can also be exposed by well-known side channel attacks, which are known to be very effective. Hence, when implementing a cryptographic scheme, resistance to existing side channel attacks must be seriously considered. This point is supported by that the resistance to side channel attacks are required in standards such as Common Criteria (CC) [25] and FIPS 140-3 [26]. However, the previous research [15]–[17], [20] only addressed the hardware implementation of NTRU and RLizard, but did not include the countermeasure methods against side channel attacks.

In this paper, we show the improved hardware design of the RLizard cryptosystem. The contributions are as follows:

- The processing speed of the conventional RLizard crypto-processor [20] was significantly improved with almost no area overhead.
- The vulnerabilities to power analysis [27]–[30] were analyzed, and countermeasures were proposed.

If the order of polynomials is large, memory usage is essential, and managing memory accesses can have a significant impact on the processing speed. The main difficulty is that the amount of data that can be accessed from memory at one time is limited, so it is not simple to apply parallel processing with area increase for speed improvement. Our contribution is that under the data access limitation, we cleverly improved the processing speed by using discarded values for each memory access. This contrasts to the conventional RLizard crypto-processor in [20] that uses only one of the loaded values for each memory access. Additionally, the proposed method is applicable to hardware implementations of other cryptosystems using a binary or ternary polynomial, such as NTRU and Round5.

We also show the conventional RLizard crypto-processor's vulnerabilities to the power analysis. Although [31] has already conducted similar research on NTRU, their research focused on software implementation, and its computation of convolution is slightly different from that of our implementation. As a result, the power analysis experimental results were also different, and different countermeasures were proposed.

The organization of this work is as follows. Section II provides information regarding the cryptographic algorithms and main operations of the RLizard scheme. In Section III, the common overall structure is shown, and the proposed speed-up method is presented. In Section IV, the vulnerability to power analysis attacks and the proposed countermeasures are shown with the experimental results. Section V compares the implementation results of the RLizard crypto-processors. Section VI concludes this work.

II. PRELIMINARY

In this section, the cryptographic algorithms of the RLizard scheme are described. Then, the main operations of the algorithms for the hardware implementation are explained.

A. RLIZARD SCHEME

The RLizard scheme has three algorithms: RLizard.KeyGen, RLizard.Enc, and RLizard.Dec. Their inputs and outputs, such as keys, plaintexts, and cipher-texts, are expressed as polynomials, and the algorithms perform polynomial operations. The detailed operations are shown in Table 1, where p and q are powers of two ($p < q$), and polynomials and their operations are defined over a ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

TABLE 1. Three algorithms of RLizard scheme.

Algorithm	Polynomial operations	Output
RLizard. KeyGen	$b(x) = a(x) \cdot s(x) + e(x)$	$b(x)$
RLizard.Enc	$c_1'(x) = a(x) \cdot r(x)$, $c_2'(x) = b(x) \cdot r(x) + \frac{q}{t} m(x)$	$c_1(x) = \left\lfloor \frac{p}{q} c_1'(x) \right\rceil$, $c_2(x) = \left\lfloor \frac{p}{q} c_2'(x) \right\rceil$
RLizard.Dec	$m'(x) = c_2(x) - c_1(x) \cdot s(x)$	$m''(x) = \left\lfloor \frac{t}{p} m'(x) \right\rceil$

In Table 1, a private key $s(x)$ and temporary secret $r(x)$ are sparse ternary secrets, so most of their coefficients are zero, and only h coefficients have nonzero values of ± 1. An error polynomial $e(x)$ has coefficients in the range $(-7, 7)$ according to Gaussian distribution. $a(x)$ and $b(x)$ are public keys with coefficients in $\mathbb{Z}_q$. $m(x)$ and $m''(x)$ are binary polynomials representing plaintext. $\left(c_1'(x), c_2'(x)\right) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^n$ is the intermediate result of the encryption algorithm and is scaled down by rounding off, so a cipher-text $c = (c_1(x), c_2(x)) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^n$.

B. MAIN OPERATIONS OF RLIZARD

If RLizard.Enc is divided into RLizard.Enc1 for $c_1(x)$ and RLizard.Enc2 for $c_2(x)$, RLizard has four algorithms. Let their common polynomial operations be

$$f(x) = u(x) \cdot v(x) + w(x)\,. \tag{1}$$

The polynomials of (1) for each algorithm are shown in Table 2. Note that $q/2p$ in w_i of the RLizard.Enc1 and Enc2 changes rounding off to rounding down. RLizard.Dec also requires rounding off, but m_i'' can be simply obtained by XORing the two most significant bits of m_i'.

TABLE 2. Coefficients of polynomials used in each algorithm.

Algorithm	f_i	u_i	v_i	w_i
RLizard.KeyGen	b_i	s_i	a_i	e_i
RLizard.Enc1	$c'_{1,i}$	r_i	a_i	$q/2p$
RLizard.Enc2	$c'_{2,i}$	r_i	b_i	$(q/2)m_i + q/2p$
RLizard.Dec	m'_i	s_i	$-c_{1,i}$	$c_{2,i}$

Since $x^n \equiv -1 \pmod{x^n+1}$, (1) can be represented as

$$f_k = w_k + \sum\nolimits_{i+j \equiv k \bmod n} (-1)^{\left\lfloor \frac{i+j}{n} \right\rfloor} \cdot u_i v_j. \tag{2}$$

$u(x)$ is the ternary polynomial of $s(x)$ or $r(x)$, so instead of (2), we can use

$$f_k = w_k + \sum\nolimits_{u_{pos}[i]+j \equiv k \bmod n} (-1)^{u_{sign}[i] + \left\lfloor \frac{u_{pos}[i]+j}{n} \right\rfloor} \cdot v_j, \tag{3}$$

where u_{pos} and u_{sign} are arrays representing the positions of the nonzero coefficients and their negative signs of $u(x)$, respectively. For example, when $n = 8$, $h = 4$, and $u(x) = 1 - x^4 + x^5 + x^7$, $u_{pos} = [0, 4, 5, 7]$ and $u_{sign} = [0, 1, 0, 0]$. Using (3), (1) can be computed as shown in Algorithm 1. f_k for $0 \le k < n$ is initialized with w_k in line 2, and $\pm v_{k-u_{pos}[i]}$ $(0 \le i < h)$ are accumulated to f_k in lines 3-7. Except for the initial value of f_k in line 2, Algorithm 1 is similar to the convolution computation algorithm in [31]; however, Algorithm 1 is more suitable for hardware implementation, and a ternary polynomial is used instead of a binary polynomial.

Algorithm 1 Convolution Product and Addition

```
Input: $u_{pos}$, $u_{sign}$, $v(x)$, $w(x)$
Output: $f(x)$
1:  for $k = 0$ to $n - 1$
2:      $f_k \leftarrow w_k$
3:      for $i = 0$ to $h - 1$
4:          $j \leftarrow k - u_{pos}[i]$
5:          $neg \leftarrow (j < 0) \oplus u_{sign}[i]$   // $\oplus$ : XOR operation
6:          $f_k \leftarrow f_k + (-1)^{neg} \cdot v_{j \bmod n}$
7:      end for
8:  end for
```

III. HARDWARE DESIGN OF RLIZARD CRYPTO-PROCESSORS

In this section, the overall hardware structure of the conventional RLizard crypto-processor in [20] is first explained. Then, our speed-up method is presented, and the modified hardware design is compared with the conventional one. For simple notation, the conventional RLizard crypto-processor is called *RLZDx1*, and the two proposed RLizard crypto-processors are called *RLZDx2* and *RLZDx4*, as these crypto-processors are up to two and four times faster than RLZDx1, respectively.

A. COMMON HARDWARE STRUCTURE

The overall structure of the RLizard crypto-processor in [20] is shown in Fig. 1. The adders and registers for accumulation only require a small number of resources, and the dual-port memory for storing polynomials occupies most of the area. For example, key generation requires a total 1152-word space for 512-word $a(x)$, 128-word $s(x)$ and 512-word $b(x)$ when the recommended parameters, $p = 256$, $q = 1024$, $t = 2$, $n = 1024$, and $h = 128$ [18] are used. From now, we use the recommended parameters if not mentioned. The required spaces of polynomials are shown in Table 3.

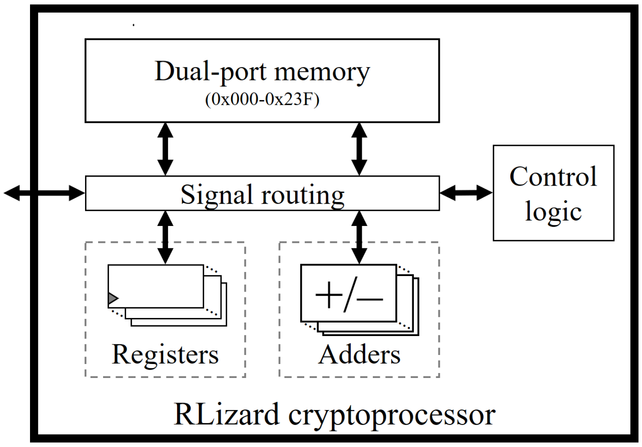


FIGURE 1. Overall structure of RLizard crypto-processors.

TABLE 3. Required memory sizes of input polynomials.

Polynomials	Required memory size
$s(x), r(x)$	11-bit × $h = h/2$ words = 64 words
$a(x), b(x)$	10-bit × $n = n/2$ words = 512 words
$c_1(x), c_2(x)$	8-bit × $n = n/4$ words = 256 words

To reduce the memory size, the RLizard crypto-processor in [20] used the following two methods. First, $e(x)$ and $m(x)$ are stored in the remaining bits of the words for $a(x)$ and $b(x)$, as 12-bit space is still unused after storing two 10-bit coefficients of $a(x)$ or $b(x)$ within a word. Second, the output of each algorithm, $f(x)$ is not stored in the memory. Instead, whenever one-word size of coefficients of $f(x)$ are computed and collected, the word is outputted. For example, key generation requires only 640 words for input polynomials, $a(x)$ and $s(x)$, instead of 1152 words to additionally store the output polynomial, $b(x)$. The proposed RLizard crypto-processors also have the same hardware structure.

B. PROPOSED SPEED-UP METHOD

To present the proposed speed-up method, this subsection compares the algorithms of the RLizard crypto-processors. At first, the algorithm of RLZDx1 is shown. Then, it is shown how the proposed method can improve the processing speed.

1) THE CONVENTIONAL METHOD IN THE RLZDX1

RLZDx1 performs (1) according to Algorithm 2. Compared to Algorithm 1, codes about memory access and uses of registers are added. Note that Mem0, Mem1, and Mem2 are logically separated for simple explanation. They are implemented as one dual-port memory in the crypto-processors. The function *HalfWrd* in lines 2, 5, and 9 selects the upper or lower half word. In decryption, w_i and v_i are one-byte size, so the lines 2 and 9 are replaced with the following codes: $f_k \leftarrow$ Byte(Mem2 $[k \gg 2]$, k mod 4) and $v_{j'} \leftarrow$ Byte(Mem1$[j' \gg 2], j$ mod 4), where the function *Byte* selects one byte from the given word.

Algorithm 2 Convolution and Addition in RLZDx1

```
Input: Mem0 $\leftarrow (u_{pos}, u_{sign})$, Mem1 $\leftarrow v(x)$, Mem2 $\leftarrow w(x)$
Output: $f(x)$
1:  for $k = 0$ to $n-1$
2:      $w_k \leftarrow$ HalfWrd(Mem2 $[k \gg 1]$ , $k$ mod 2)
3:      $t_0 \leftarrow w_k$
4:      for $i = 0$ to $h-1$
5:          $(u_{\text{sign}}[i], u_{pos}[i]) \leftarrow$ HalfWrd(Mem0$[i \gg 1]$, $i$ mod 2)
6:          $j \leftarrow k - u_{pos}[i]$
7:          $j' = j$ mod $n$
8:          $neg \leftarrow (j < 0) \oplus u_{sign}[i]$
9:          $v_{j'} \leftarrow$ HalfWrd(Mem1$[j' \gg 1]$, $j$ mod 2)
10:         $t_0 \leftarrow t_0 + (-1)^{neg} \cdot v_{j'}$
11:     end for
12:     $f_k \leftarrow t_0$
13: end for
```

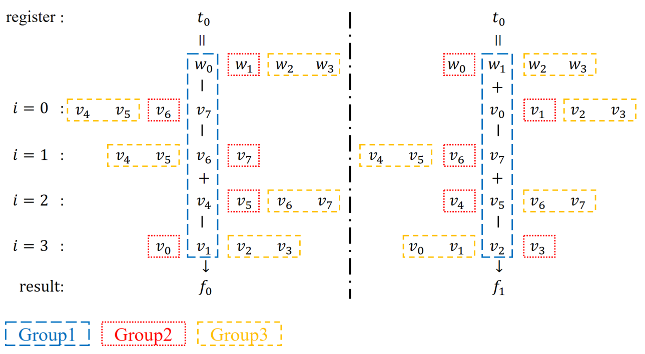


FIGURE 2. Execution example of Algorithm 2 for $n = 8$, $h = 4$, $U_{pos} = [1, 2, 4, 7]$, and $U_{sign} = [0, 0, 1, 0]$.

Fig. 2 shows an execution example of Algorithm 2. The coefficients in the blue rectangle (Group1), w_i and $v_{j'}$ are accumulated to variable t_0 to calculate f_k. Note that when loading the coefficients that belong to Group1, their adjacent coefficients are also loaded together for each memory access. In addition to Group1, one more coefficient is loaded for each memory access in the key generation and encryption. These additional coefficients are marked by the red rectangle (Group2) in the Fig. 2. For example, in Fig. 2, when w_0, v_6 and v_4 (Group1) are loaded for f_0, w_1, v_7, and v_5 (Group2) are also loaded together, respectively. Although some of coefficients in Group2, such as w_1, v_7, and v_5, can be used to calculate f_1, RLZDx1 do not use them. If they are also used, the processing speed could be improved. However, this cannot be achieved simply by adding only one adder and one register, because only some coefficients in Group2 are required for f_{k-1} and the others are required for f_{k+1}. As shown in Fig. 2, the positions of the red rectangles are not fixed.

In decryption, w_i and v_i are one-byte size, so two more coefficients, marked by the yellow rectangles (Group3) in Fig. 2, were additionally loaded together. This means that the processing speed of decryption can be higher than that of key generation and encryption.

2) PROPOSED METHOD IN THE RLZDX2 AND RLZDX4

In RLZDx2 and RLZDx4, the adjacent coefficients abandoned in RLZDx1 were additionally used. RLZDx2 uses one more coefficient (Group2), so its processing speed is double that of RLZDx1. The pseudo-code is shown in Algorithm 3. In lines 13-14 of Algorithm 3, $t_{j''}$ and $t_{j''+1}$ are two of t_{-1}, t_0, and t_1, so f_k and one of f_{k-1} and f_{k+1} are calculated together.

Algorithm 3 Convolution and Addition in RLZDx2

```
Input: Mem0 $\leftarrow (u_{pos}, u_{sign})$, Mem1 $\leftarrow v(x)$, Mem2 $\leftarrow w(x)$
Output: $f(x)$
1:  $t_{-1} \leftarrow 0$
2:  for $k = 0$ to $n-2$ step 2
3:      $(w_k, w_{k+1}) \leftarrow$ Mem2$[k \gg 1]$
4:      $t_0 \leftarrow w_k$
5:      $t_1 \leftarrow w_{k+1} - IV$          // $IV = t'_{-1}$ if $k = n-2$, or 0
6:      for $i = 0$ to $h-1$
7:          $(u_{\text{sign}}[i], u_{pos}[i]) \leftarrow$ HalfWrd(Mem0$[i \gg 1]$, $i$ mod 2)
8:          $j \leftarrow k - u_{pos}[i]$
9:          $j' = (j \bmod n) - (j \bmod 2)$
10:         $j'' = -(j \bmod 2)$
11:         $neg \leftarrow (j < 0) \oplus u_{sign}[i]$
12:         $(v_{j'}, v_{j'+1}) \leftarrow$ Mem1$[j \gg 1]$
13:         $t_{j''} \leftarrow t_{j''} + (-1)^{neg} \cdot v_j$          // $t_{j''}$: $t_{-1}$ or $t_0$
14:         $t_{j''+1} \leftarrow t_{j''+1} + (-1)^{neg} \cdot v_{j''+1}$     // $t_{j''+1}$: $t_0$ or $t_1$
15:     end for
16:     if $k = 0$ then
17:         $t'_{-1} \leftarrow t_{-1}$
18:     else then
19:         $f_{k-1} \leftarrow t_{-1}$
20:     end if
21:     $f_k \leftarrow t_0$
22:     $t_{-1} \leftarrow t_1$
23: end for
24: $f_{n-1} \leftarrow t_{-1}$
```

Fig. 3 shows an execution example of Algorithm 3. While f_0 is calculated, w_1, v_7, and v_5 in the first red rectangle are

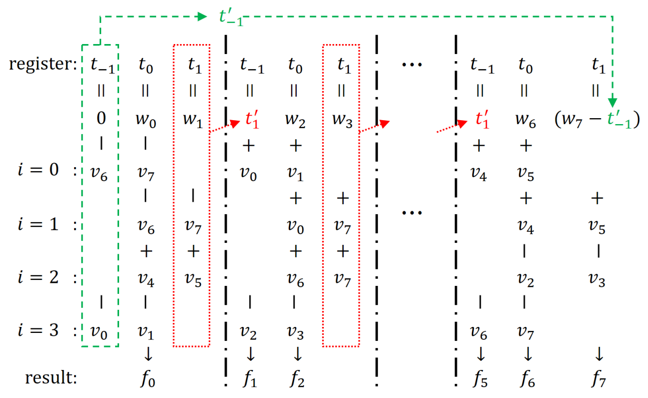


FIGURE 3. Execution example of Algorithm 3 for $n = 8$, $h = 4$, $U_{pos} = [1, 2, 4, 7]$, and $U_{sign} = [0, 0, 1, 0]$.

accumulated to a register t_1. Then, t_1 is passed to t_{-1}, and v_0 and v_2 are accumulated. From t_{-1} and t_0, f_1 and f_2 are obtained together. In the same way, $f_3, f_4, \ldots, f_{n-2}$ can be calculated. Only f_{n-1} is calculated in a different way. In Fig. 3, v_6 and v_0 in the green rectangle are accumulated to a register t_{-1}. The value in t_{-1} is kept in t'_{-1} until f_6 is calculated. Then, t'_{-1}, w_7, v_5, and v_3 are accumulated to a register t_1, and f_7 is obtained. RLZDx2 requires three additional registers, t'_{-1}, t_{-1}, and t_1. Furthermore, another adder is required to calculate one of t_{-1} and t_1.

RLZDx4 uses Algorithm 3 for key generation and encryption, and Algorithm 4 for decryption. Algorithm 4 processes four coefficients together including three adjacent coefficients (Group2 and Group3). Consequently, RLZDx4 can process decryption four times faster than RLZDx1. The additional coefficients are processed in a similar way to the method in Fig. 3.

C. HARDWARE DESIGN OF RLIZARD CRYPTO-PROCESSORS

The hardware structure of RLZDx1 and RLZDx2 is shown in Fig. 4. The largest component is the dual-port memory, and only a few small registers and adders are additionally required. The values of u_{pos} and u_{sign} are loaded through $W0$, and coefficients of $w(x)$ and $v(x)$ are loaded through $W1$. The bit-selection to extract one or two coefficients from one-word

Algorithm 4 Convolution and Addition in RLZDx4

```
Input: Mem0 ←($u_{pos}$, $u_{sign}$), Mem1 ←$v(x)$, Mem2 ←$w(x)$
Output: $f(x)$
1:  $(t_{-3}, t_{-2}, t_{-1}) \leftarrow (0, 0, 0)$
2:  for $k = 0$ to $n-4$ step 4
3:      $(w_k, w_{k+1}, w_{k+2}, w_{k+3}) \leftarrow \text{Mem2}[k \gg 2]$
4:      $t_0 \leftarrow w_k$
5:      $t_1 \leftarrow w_{k+1} - IV1$        // $IV1 = t'_{-3}$ if $k = n-4$, or 0
6:      $t_2 \leftarrow w_{k+2} - IV2$        // $IV2 = t'_{-2}$ if $k = n-4$, or 0
7:      $t_3 \leftarrow w_{k+3} - IV3$        // $IV3 = t'_{-1}$ if $k = n-4$, or 0
8:      for $i = 0$ to $h-1$
9:          $(u_{\text{sign}}[i], u_{pos}[i]) \leftarrow \text{HalfWrd}(\text{Mem0}[i \gg 1], i \bmod 2)$
10:         $j \leftarrow k - u_{pos}[i]$
11:         $j' = (j \bmod n) - (j \bmod 4)$
12:         $j'' = -(j \bmod 4)$
13:         $neg \leftarrow (j < 0) \oplus u_{sign}[i]$
14:         $(v_{j'}, v_{j'+1}, v_{j'+2}, v_{j'+3}) \leftarrow \text{Mem1}[j' \gg 2]$
15:         $t_{j''} \leftarrow t_{j''} + (-1)^{neg} \cdot v_{j''}$        // $t_{j''}$: $t_{-3}, t_{-2}, t_{-1}$, or $t_0$
16:         $t_{j''+1} \leftarrow t_{j''+1} + (-1)^{neg} \cdot v_{j''+1}$  // $t_{j''+1}$: $t_{-2}, t_{-1}, t_0$, or $t_1$
17:         $t_{j''+2} \leftarrow t_{j''+2} + (-1)^{neg} \cdot v_{j''+2}$  // $t_{j''+2}$: $t_{-1}, t_0, t_1$, or $t_2$
18:         $t_{j''+3} \leftarrow t_{j''+3} + (-1)^{neg} \cdot v_{j''+3}$  // $t_{j''+3}$: $t_0, t_1, t_2$, or $t_3$
19:     end for
20:     if $k = 0$ then
21:         $(t'_{-3}, t'_{-2}, t'_{-1}) \leftarrow (t_{-3}, t_{-2}, t_{-1})$
22:     else then
23:         $(f_{k-3}, f_{k-2}, f_{k-1}) \leftarrow (t_{-3}, t_{-2}, t_{-1})$
24:     end if
25:     $f_k \leftarrow t_0$
26:     $(t_{-3}, t_{-2}, t_{-1}) \leftarrow (t_1, t_2, t_3)$
27: end for
28: $(f_{n-3}, f_{n-2}, f_{n-1}) \leftarrow (t_{-3}, t_{-2}, t_{-1})$
```

data, $W0$ and $W1$ is simply expressed by bit organizers (BOs) such as BO_1 and BO_2 in Fig. 4. The accumulation part using $W1$ is different depending on the crypto-processor version. The RLZDx1 in Fig. 4 (a) chooses one coefficient (Group1) from $W1$ and accumulates it to the register t_0 using a single adder. After accumulation, the final value piles up in the register $WrdOut$. Meanwhile, the RLZDx2 in Fig. 4 (b) chooses two coefficients (Group1 and Group2) from $W1$ and accumulates one coefficient to the register t_0 and the other to the register t_{-1} or t_1 using one additional adder. As explained in the previous subsection, register t'_{-1} is additionally required

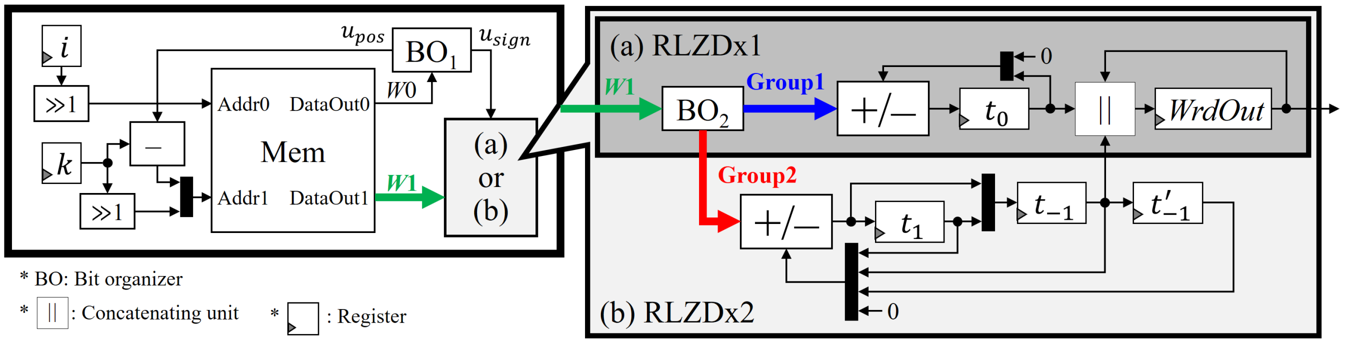


FIGURE 4. Hardware structures of RLizard crypto-processors: (a) RLZDx1 and (b) RLZDx2.

for f_{n-1}. Similarly, the RLZDx4 can be designed by adding six more registers and two more adders compared to in RLZDx2, although the RLZDx4 is not included in Fig. 4.

IV. COUNTERMEASURE AGAINST POWER ANALYSIS ATTACKS

All the above RLizard crypto-processors may be vulnerable to CPA attacks [28]. In this section, the vulnerability of RLZDx1 is analyzed representatively, and countermeasures against CPA attacks are proposed. Then, we present the power analysis experimental results on all the above RLizard crypto-processors.

A. VULNERABILITY TO POWER ANALYSIS ATTACKS

The power consumption depends on the processed data. By analyzing the correlation between data and power traces, secret data can be revealed. These attacks are called correlation power analysis (CPA). The main purpose of these attacks is to reveal the secret key; hence, crypto-processors are usually attacked while performing decryptions.

In this work, we conducted a CPA according to a typical attack model in [30] as follows.

- Step 1: the attacker selects an intermediate result which is calculated from the cipher-texts $(c_1(x), c_2(x))$ and key (s_{sign}, s_{pos}), where (s_{sign}, s_{pos}) includes the information of the non-zero coefficients of the private key, $s(x)$ like (u_{sign}, u_{pos}). When $k = 0$ and $i = 0$, the RLZDx1 sequentially performs

$$t_0 \leftarrow c_{2,0}, \quad t_0 \leftarrow t_0 - (-1)^{neg} \cdot c_{1,j'}, \tag{4}$$

  where $j' = -s_{pos}[0] \bmod n$, and $neg = \left(s_{pos}[0] < 0\right) \oplus s_{sign}[0]$. The attacker can select t_0 in (4) as the intermediate result for CPA.
- Step 2: while running the RLizard processor with a constant key and non-constant cipher-texts, the attacker measures the corresponding power traces and recodes the used cipher-texts and measured power traces.
- Step 3: the attacker calculates the hypothetical values of t_0 in (4) for the cipher-texts used in Step 2 and possible guesses at the key since it is supposed that the key is not known for the attacker.
- Step 4: the attacker maps the hypothetical intermediate values obtained in Step 3 to power consumption values. While (4) is performed, some bits of the register t_0 are changed, which causes power consumption. The number of changed bits in the register t_0 can be expressed as:

$$HD\left(c_{2,0}, c_{2,0} - (-1)^{neg} \cdot c_{1,j'}\right), \tag{5}$$

  where $HD(x, y)$ is the Hamming distance (HD) of x and y. This is an HD model, which is one of the most commonly used power models.
- Step 5: the attacker calculates correlation coefficients between the power traces measured in Step 2 and the hypothetical power consumption obtained in Step 4. If the guess at the key is correct, there must be a meaningful correlation. That is, $s_{pos}[0]$ and $s_{sign}[0]$ can be determined from the guess with the highest correlation value. After $s_{pos}[0]$ and $s_{sign}[0]$ are known, $s_{pos}[i]$ and $s_{sign}[i]$ for $1 \leq i < h$ can be sequentially found in the same way.

B. PROPOSED COUNTERMEASURE

Two types of countermeasures were used. The first method is masking, which eliminates the correlation between data and power consumption. A random number is usually mixed into the intermediate value related to the secret to prevent the HD for each guess from being calculated. In the RLizard crypto-processor, the intermediate value is stored in the register t_0. The second method is hiding, which makes the power consumption constant or randomized regardless of the processed data. In the RLizard crypto-processor, the order of loading $s_{pos}[i]$ and $s_{sign}[i]$ can be randomized. The RLZDx1 with these countermeasures against CPA attacks is called *RLZDx1-AntiCPA*, and the pseudo-code of RLZDx1-AntiCPA for decryption is shown in Algorithm 5.

Algorithm 5 Convolution and Addition in RLZDx1-AntiCPA

```
Input: Mem0 $\leftarrow (u_{pos}, u_{sign})$, Mem1 $\leftarrow v(x)$, Mem2 $\leftarrow w(x)$
Output: $f(x)$
1:  $r_H \leftarrow GenRN(\log_2 h)$ //$GenRN(l)$: generates $l$ random bits
2:  for $k = 0$ to $n-1$
3:      $t_0 \leftarrow$ HalfWord(Mem2 $[k \gg 1]$, $k$ mod 2) // $t_0 \leftarrow w_k$
4:      $r_M \leftarrow 0$
5:      for $i = r_H + 1$ to $r_H$ step 1 (mod $h$)
6:          $r_T \leftarrow GenRN(\log_2 p)$
7:          $(u_{sign}[i], u_{pos}[i]) \leftarrow$ HalfWrd(Mem0$[i \gg 1]$, $i$ mod 2)
8:          $j \leftarrow k - u_{pos}[i]$
9:          $j' = j \bmod n$
10:         $neg \leftarrow (j < 0) \oplus u_{sign}[i]$
11:         $v_{j'} \leftarrow$ Byte(Mem1$[j' \gg 2]$, $j$ mod 4)
12:         $t_0 \leftarrow t_0 + (-1)^{neg} \cdot (v_{j'} + r_T)$
13:         $r_M \leftarrow r_M + (-1)^{neg} \cdot r_T$
14:     end for
15:     $f_k \leftarrow t_0 - r_M$
16: end for
```

For masking, we tried two methods. First, we randomized the initial value of the register t_0. As a result, the actual power consumed corresponds to

$$HD\left(c_{2,0} + r, c_{2,0} + r - (-1)^{neg} \cdot c_{1,j'}\right), \tag{6}$$

where r is a random number. However, our experimental result shows that this masking method is not effective; hence, this masking method was not used and was excluded from Algorithm 5. This is because (5) and (6) have a high correlation value. This can be confirmed by that $HD(r_0, r_0 + r_1)$ and $HD(r_0 + r_2, r_0 + r_2 + r_1)$ have a correlation value 0.45, where r_0, r_1, and r_2 are random numbers.

Alternatively, we randomized $c_{1,j'}$, which is each coefficient of $c_1(x)$ accumulated to the register t_0. As a result, actually consumed power corresponds to

$$HD\left(c_{2,0}, c_{2,0} - (-1)^{neg} \cdot (c_{1,j'} + r_T)\right). \tag{7}$$

In lines 6 and 12 of Algorithm 5, a random number r_T is generated for each i and added to $c_{1,j'}$ before $c_{1,j'}$ is accumulated to the register t_0. r_T is also accumulated to r_M in line 13, and r_M is removed from f_k in line 15. This masking method blinds all selected coefficients of $c_1(x)$, so simple power analysis, which observes the operations related zero values after setting coefficients of input polynomial $v(x)$ as zeros and nonzeros, can also be prevented. Furthermore, different random values of r_T are used for $0 \leq i < h$; thus, it is expected that second-order CPA can also be prevented.

For hiding, we randomized the starting value of i in line 5 of Algorithm 5 to randomize the moment when $s_{pos}[0]$ and $s_{sign}[0]$ are loaded from memory and used. This prevents $s_{pos}[0]$ and $s_{sign}[0]$ from being processed at the same time point after the start. Moreover, the values previously accumulated in t_0 at that time also changes every time. As a result, the correlation for the right guess becomes too small to be distinguishable.

C. EXPERIMENTAL RESULTS

We implemented RLizard crypto-processors on XC3S1500 (Spartan 3) using Xilinx ISE 14.2 and acquired power traces using LeCroy DDA-3000 Oscilloscope with 10 GHz sample rate at 1 MHz of clock frequency. We used a fixed secret, $\left(s_{sign}[0], s_{pos}[0]\right) = (0, 491)$. 40,000 different cipher-texts $\left(c_1(x), c_2(x)\right)$ were used when attacking RLizard crypto-processors without countermeasure, and the number of cipher-texts was increased to 200,000 when any countermeasure was used.

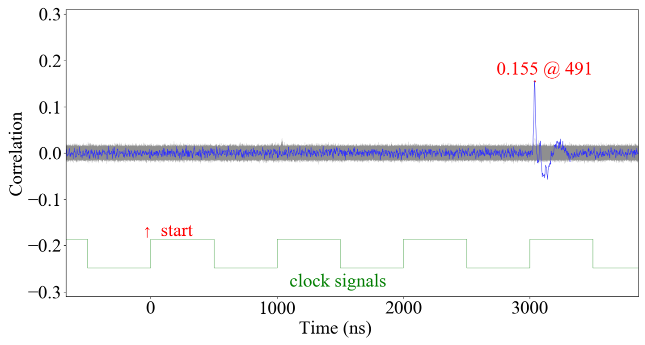


FIGURE 5. Correlation between the power consumption and the Hamming distances for each possible guess.

At first, we implemented RLZDx1 and attacked. Fig. 5 shows the correlations for all the possible guesses over time. The blue line represents the correlation of the right guess, which has the highest value at ~3000 ns after the start signal. Instead of correlations over time, observing only the maximum correlations for all the possible guesses is sufficient to determine the right guess. Fig. 6 shows those of RLZDx1, RLZDx2, and RLZDx4 with no countermeasure, with masking, and with hiding, where the possible guess at the value of $(s_{sign}[0], s_{pos}[0])$ has values from zero to 2047. Even though only the guess at $s_{pos}[0]$ is right, the maximum correlation can be high regardless of the guess at $s_{sign}[0]$, so we represented all the maximum correlations for the right guess $s_{pos}[0]$ as red dots.

In Figs. 6 (a), (d), and (g), RLZDx1, RLZDx2, and RLZDx4 have maximum correlation values of 0.155, 0.172, and 0.287 for the right guess. These values are higher than those for the other guesses. This means that attacks were successful. Note that the maximum correlation values of the RLZDx2 and RLZDx4 are higher than that of RLZDx1. This is because two and four coefficients instead of one are selected depending on $s_{pos}[0]$ in RLZDx2 and RLZDx4, respectively. This may make the attack on RLZDx2 and RLZDx4 easier than that on the RLZDx1.

Figs. 6 (b), (e), and (h) show the experimental results on RLizard crypto-processors with masking. The maximum correlation for the right guess is not distinguishable, so we can see that the countermeasure is effective. On the other hand, Fig. 7 shows the experimental results on RLZDx1 with the method that randomizes the initial value of t_0. t_0 is initialized as $c_{2,0} + r$ instead of $c_{2,0}$, and power consumption is affected by random number r as explained by (6). As shown in Fig. 7, the maximum correlation for the right guess was reduced, but is still higher than the others. This means that this method is not effective enough to prevent CPA, as explained in the previous sub-section.

Figs. 6 (c), (f), and (i) show the experimental results on RLizard crypto-processors with hiding, which are similar to those when masking is applied. That is, this method is also an effective countermeasure against CPA.

V. IMPLEMENTATION RESULTS

This section presents the implementation results of RLizard crypto-processors to analyze the area overhead of the proposed speed-up method and the proposed countermeasure against CPA.

A. PERFORMANCE COMPARISON OF RLIZARD CRYPTO-PROCESSORS

We implemented RLizard crypto-processors in three different Xilinx FPGA devices, XC3S1500, XC6SLX9 and XC6VCX75T, using Xilinx ISE 14.2. The implementation results after place and route are shown in Table 4. The source codes of [20] were slightly more optimized, so the results of RLZDx1 obtained here are slightly different from those of the source codes provided in [20]. In addition, XC3S1500 requires more FFs than the other devices. This is because the memory used in XC3S1500 does not support byte-write, which is useful when storing the coefficients of $e(x)$ and $m(x)$ in the remaining bits of each word after storing the coefficients of $a(x)$ and $b(x)$.

The RLZDx2 and RLZDx4 have processing speeds twice or four times faster than RLZDx1. Although the

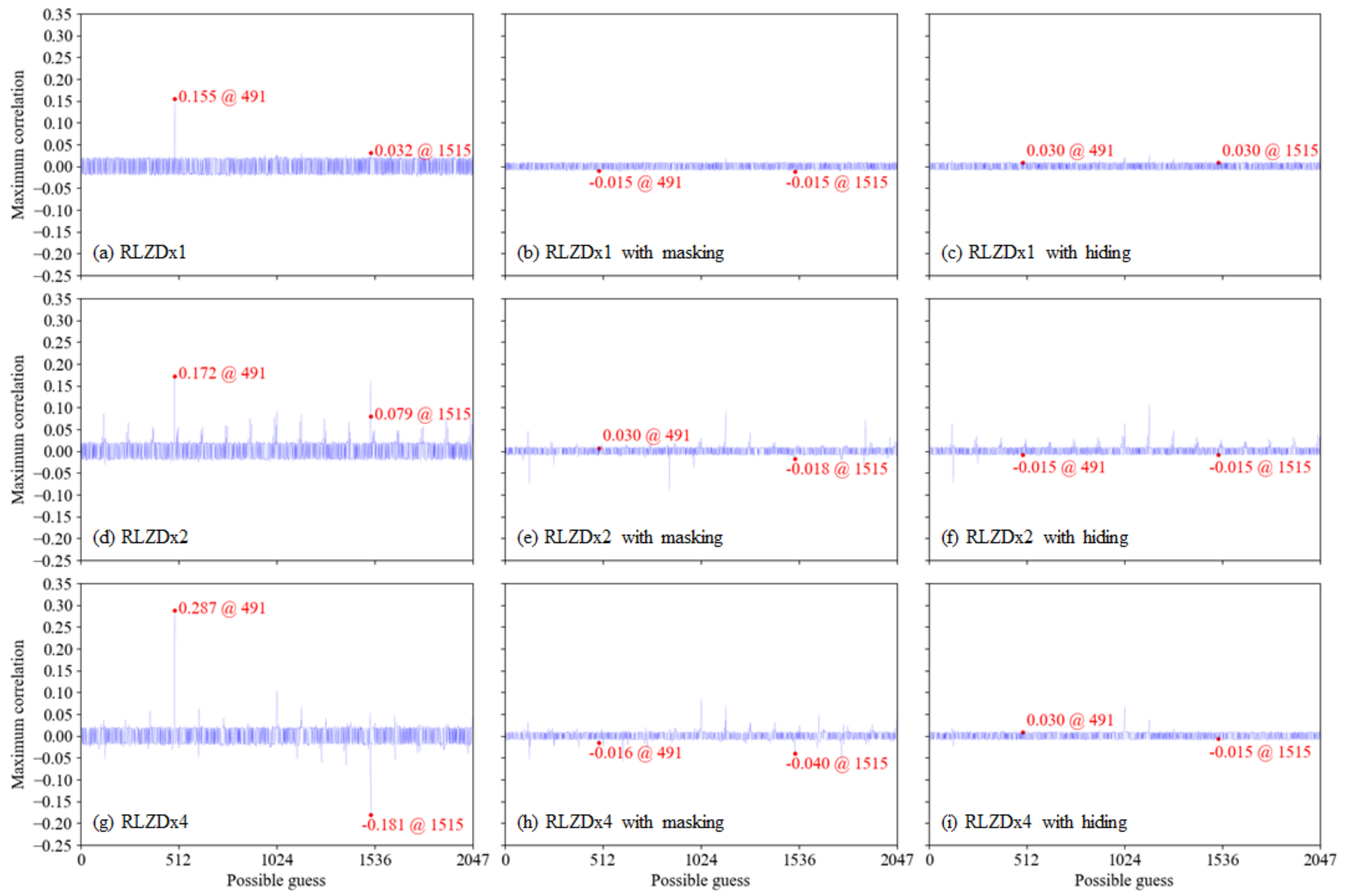


FIGURE 6. Maximum correlations for each possible guess of: (a) RZDx1, (b) RZDx1 with masking, (c) RZDx1 with hiding, (d) RZDx2, (e) RZDx2 with masking, (f) RZDx2 with hiding, (g) RZDx4, (h) RZDx4 with masking, and (i) RZDx4 with hiding.

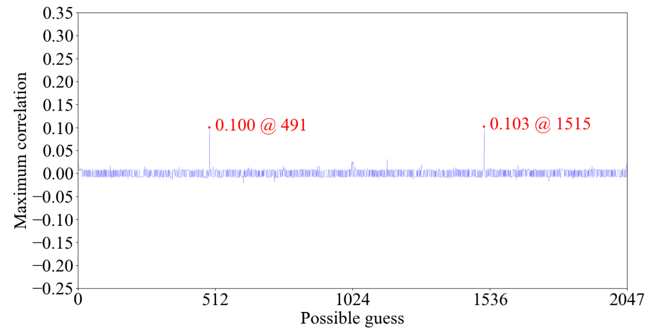


FIGURE 7. Maximum correlations for each possible guess of RLZDx1 with ineffective masking.

numbers of flip-flops (FFs) and lookup tables (LUTs) increase approximately 20% to 140%, it cannot be said that the required resources have increased much if two block RAMs (BRAMs), which are the largest components, are included. According to [20], the area of the memory part is 98.3k gate equivalents (GEs), and the area of the other part is only 1.4k GEs in ASIC implementation using the Samsung 65-nm CMOS process. The area except for memory is less than 2% of the total area. Therefore, the processing time of the RLizard crypto-processor was improved to be two and four times faster, but the area overhead is negligible.

B. IMPLEMENTATION OVERHEAD OF COUNTERMEASURE AGAINST POWER ANALYSIS ATTACKS

Table 4 also shows the area overhead when the proposed countermeasure is applied. Compared with RLZDx1, RLZDx1-AntiCPA has an additional 51 FFs and 96 LUTs in XC6SLX9, most of which are caused by the registers and a random number generator (RNG) to store and generate r_H and r_T. To reduce the overhead, a compact RNG [32], which requires only 15 FFs and 18 LUTs, was chosen. Because the RNG can be used to generate random errors in RLizard.KeyGen, the area of the RNG can be removed from the area overhead. In this case, the area overhead is only 36 FFs and 78 LUTs.

In terms of the processing speed, r_T for $i = 0$ and $k = 0$ and r_H can be generated while the crypto-processor is in an idle state, and r_T for $1 \le i < n$ can be generated in parallel with the calculation of f_i $(0 \le i < n-1)$ using 15-bit LFSR. The size of LFSR is determined according to $\log_2 hp = 15$, as r_H is used to blind hp-bit coefficients of $c_1(x)$. This does not increase the required clock cycles at all and reduces the maximum clock frequency, but not very large.

TABLE 4. Performance comparison with other lattice-based crypto-processors.

Processor	Device	Implementation results				Processing speed				Area overhead for anti-CPA		
		#BRAMs / #DSPs	#FFs	#LUTs	Max clk (MHz)	Clock cycles		Time (ms)		#FFs	#LUTs	Max clk (MHz)
						Enc.	Dec.	Enc.	Dec.			
RLZDx1 [20]	XC3S1500	2 / 0	87	468	99.6	264.2 k	132.1 k	2.65	1.33	+50	+142	87.2
RLZDx2 (Ours)			134	618	87.5	132.1 k	66.0 k	1.51	0.75	+74	+232	80.3
RLZDx4 (Ours)			183	781	79.7	132.1 k	33.0 k	1.66	0.41	+122	+365	73.5
RLZDx1 [20]	XC6SLX9	2 / 0	68	226	163.4	264.2 k	132.1 k	1.62	0.81	+51	+96	150.6
RLZDx2 (Ours)			115	287	153.5	132.1 k	66.0 k	0.86	0.43	+75	+151	141.3
RLZDx4 (Ours)			167	424	145.3	132.1 k	33.0 k	0.91	0.23	+123	+226	125.4
RLZDx1 [20]	XC6VCX75T	1 / 0	68	221	202.1	264.2 k	132.1 k	1.31	0.65	+47	+94	178.0
RLZDx2 (Ours)			115	267	193.8	132.1 k	66.0 k	0.68	0.34	+67	+154	175.0
RLZDx4 (Ours)			164	414	182.9	132.1 k	33.0 k	0.72	0.18	+115	+202	164.1

VI. CONCLUSION

In this work, we proposed hardware implementation methods to improve the processing speed and the security against CPA of RLizard crypto-processor. Our implementation results show that the proposed RLizard crypto-processor can have twice and four times faster processing speed with little area overhead than the conventional one. As for the security against CPA attacks, the vulnerability was analyzed, and the proposed countermeasures were validated by experimental results. The implementation results show that the overhead for the proposed countermeasures is also very little. Although we did not perform simple power analysis (SPA) [27] and second-order power analysis [29], we expect that our countermeasure method would be strong against them. In further work, such experiments would also be performed. In addition, we expect that our proposed methods can be applied to other ideal-lattice-based cryptosystems using binary or ternary polynomials such as NTRU and Round5.

ACKNOWLEDGMENT

The authors would like to thank Jae Hong Seo at Hanyang University and Mun-Kyu Lee at Inha University, who gave us comments and reviewed this manuscript. The chip fabrication and EDA tool were supported by the IC Design Education Center(IDEC), Korea.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. CRYPTO*, Santa Barbara, CA, USA, 1985, pp. 417–426.

[3] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, 1987.

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, 1999.

[5] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 34-1–34-40, 2009.

[6] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Proc.EUROCRYPT*, Cambridge, U.K., 2012, pp. 719–737.

[7] P. Zhang, H. Jiang, Z. Zheng, P. Hu, and Q. Xu, "A new post-quantum blind signature from lattice assumptions," *IEEE Access*, vol. 6, pp. 27251–27258, 2018.

[8] Y. Zhang, S. Wang, and Q. Du, "Revocable identity-based encryption scheme under LWE assumption in the standard model," *IEEE Access*, vol. 6, pp. 65298–65307, 2018.

[9] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proc. EUROCRYPT*, Monaco Nice, France, 2010, pp. 1–23.

[10] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. Huss, "On the design of hardware building blocks for modern lattice-based encryption schemes," in *Proc. CHES*, Leuven, Belgium, 2012, pp. 512–529.

[11] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," in *Proc. CHES*, Busan, Korea, 2014, pp. 371–391.

[12] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. Pao, and I. Verbauwhede, "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 1, pp. 157–166, Jan. 2015.

[13] C. Du, G. Bai, and X. Wu, "High-speed polynomial multiplier architecture for ring-LWE based public key cryptosystems," in *Proc. GLSVLSI*, Boston, MA, USA, May 2016, pp. 9–14.

[14] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Proc. ANTS*, Portland, OR, USA, 1998, pp. 267–288.

[15] OnBoardSecurity. *NTRUencrypt and pqNTRUsign Submission to NIST*. Accecced: Apr. 1, 2019. [Online]. Available: https://www.onboardsecurity.com/nist-post-quantum-crypto-submission

[16] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal, "NTRU prime: Reducing attack surface at low cost," in *Proc. SAC*, Ottawa, ON, Canada, 2017, pp. 235–260.

[17] S. Bhattacharya, O. Garcia-Morchon, T. Laarhoven, R. Rietman, M.-J. O. Saarinen, L. Tolhuizen, and Z. Zhang, (2018). *Round5: Compact and Fast Post-Quantum Public-Key Encryption*. [Online]. Available: https://round5.org/doc/round5paper.pdf

[18] J. H. Cheon, D. Kim, J. Lee, and Y. Song, "Lizard: Cut off the tail! A practical post-quantum public-key encryption from LWE and LWR," in *Proc. SCN*, Amalfi, Italy, 2018, pp. 160–177.

[19] J. Lee, D. Kim, H. Lee, Y. Lee, and J. H. Cheon, "RLizard: Post-quantum key encapsulation mechanism for IoT devices," *IEEE Access*, vol. 7, pp. 2080–2091, 2019.

[20] *National Institute of Standards and Technology, Post-Quantum Cryptography—Round 1 Submissions*. Accecced: Apr. 1, 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/Lizard.zip

[21] *National Institute of Standards and Technology, Post-Quantum Cryptography*. Accecced: Apr. 1, 2019. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[22] A. C. Atici, L. Batina, J. Fan, I. Verbauwhede, and S. B. O. Yalcin, "Low-cost implementations of NTRU for pervasive security," in *Proc. ASAP*, Leuven, Belgium, Jul. 2008, pp. 79–84.

[23] A. A. Kamal and A. M. Youssef, "An FPGA implementation of the NTRUEncrypt cryptosystem," in *Proc. ICM*, Marrakech, Morocco, Dec. 2009, pp. 209–212.

[24] F. Hu, K. Wilhelm, M. Schab, M. Lukowiak, S. Radziszowski, and Y. Xiao, "NTRU-based sensor network security: A low-power hardware implementation perspective," *Secur. Commun. Netw.*, vol. 2, no. 1, pp. 71–81, Jan./Feb. 2009.

[25] *Information Technology-Security Techniques-Evaluation Criteria for IT Security*, Standard ISO/IEC 15408, 2009.

[26] *Security Requirements for Cryptographic Modules*, Standard FIPS PUB 140-3, NIST, Gaithersburg, MD, USA, 2019.

[27] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. CRYPTO*, Santa Barbara, CA, USA, 1999, pp. 388–397.

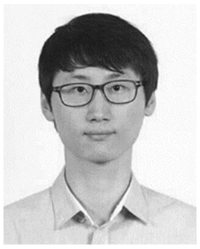
[28] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Proc. CHES*, Cambridge, MA, USA, 2004, pp. 16–29.

[29] T. S. Messerges, "Using second-order power analysis to attack DPA resistant software," in *Proc. CHES*, Worcester, MA, USA, 2000, pp. 238–251.

[30] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. New York, NY, USA: Springer, 2007.

[31] M.-K. Lee, J. E. Song, D. Choi, and D.-G. Han, "Countermeasures against power analysis attacks for the NTRU public key cryptosystem," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 93, no. 1, pp. 153–163, Jan. 2010.

[32] P. Choi, M.-K. Lee, and D. K. Kim, "Fast compact true random number generator based on multiple sampling," *Electron. Lett.*, vol. 53, no. 13, pp. 841–843, Jun. 2017.



PILJOO CHOI was born in Seoul, South Korea, in 1982. He received the B.S., M.S., and Ph.D. degrees in electronic computer engineering from Hanyang University, Seoul, in 2010, 2012, and 2018, respectively, where he is currently a Professor in the Software Education Committee. His research interests include security system on chip (SoC), crypto-coprocessors, and information security.

JI-HOON KIM received the B.S. *(summa cum laude)* and Ph.D. degrees in electrical engineering and computer science from KAIST, Daejeon, South Korea, in 2004 and 2009, respectively. In 2009, he joined Samsung Electronics. In 2018, he joined the faculty of the Department of Electronic and Electrical Engineering, Ewha Womans University, where he is currently an Associate Professor. His current interests include CPU/DSP, communication modem, and low-power SoC design for security/biomedical systems. He is a Technical Committee Member of the circuits and systems for communications and VLSI systems and applications in the IEEE Circuits and Systems Society. He was a recipient of the Best Design Award at the Dongbu HiTek IP Design Contest, in 2007, and the First Place Award at the International SoC Design Conference Chip Design Contest, in 2008.

DONG KYUE KIM was born in Seoul, South Korea, in 1968. He received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, in 1992, 1994, and 1999, respectively. From 1999 to 2005, he was an Assistant Professor with the Division of Computer Science and Engineering, Pusan National University. Since 2006, he has been a Professor with the Department of Electronic Engineering, Hanyang University. His research interests include security SoC, secure crypto-processors, crypto-coprocessors, and information security systems.