Novel Side-Channel Attacks on Quasi-Cyclic Code-Based Cryptography

Bo-Yeon Sim[1], Jihoon Kwon[2], Kyu Young Choi[2], Jihoon Cho[2], Aesun Park[3] and Dong-Guk Han[1,3,†]

[1] Department of Mathematics, Kookmin University, Seoul, Republic of Korea
{qjdusls,christa}@kookmin.ac.kr
[2] Security Research Team, Samsung SDS, Inc., Seoul, Republic of Korea
{jihoon.kwon,ky12.choi,jihoon1.cho}@samsung.com
[3] Department of Financial Information Security, Kookmin University, Seoul, Republic of Korea
aesons@kookmin.ac.kr

Abstract. Chou suggested a constant-time implementation for quasi-cyclic moderate-density parity-check (QC-MDPC) code-based cryptography to mitigate timing attacks at CHES 2016. This countermeasure was later found to become vulnerable to a differential power analysis (DPA) in private syndrome computation, as described by Rossi *et al.* at CHES 2017. The proposed DPA, however, still could not completely recover accurate secret indices, requiring further solving linear equations to obtain entire secret information. In this paper, we propose a multiple-trace attack which enables to completely recover accurate secret indices. We further propose a single-trace attack which can even work when using ephemeral keys or applying Rossi *et al.*'s DPA countermeasures. Our experiments show that the BIKE and LEDAcrypt may become vulnerable to our proposed attacks. The experiments are conducted using power consumption traces measured from ChipWhisperer-Lite XMEGA (8-bit processor) and ChipWhisperer UFO STM32F3 (32-bit processor) target boards.

Keywords: Side-Channel Attack · Quasi-Cyclic Code-Based Cryptography · QC-MDPC · QC-LDPC · Multiple-Trace Attack · Single-Trace Attack

1 Introduction

The security of public key cryptosystems (PKCs) primarily is based on the difficulty of number theory problems, such as factoring large integers or finding discrete logarithms. Shor, however, proposed an algorithm that can solve such problems in polynomial time, given a practical large-scale quantum computer [Sho94]. Since quantum computers become critical threats to the current PKCs, such as Rivest-Shamir-Adleman (RSA) and elliptic curve cryptography (ECC) [RSA78, Mil85, Kob87], there are an increasing needs for post-quantum cryptography (PQC) that is secure against both quantum and classical computers.

The National Security Agency (NSA) thus announced that the list of Suite B cryptographic algorithms would be updated to PQC algorithms [Age15]. The National Institute of Standards and Technology (NIST) also released an internal report (NISTIR) 8105: Reports on PQC [CCJ+16], giving an analysis of the current state of quantum computing and then discussing the need of PQC standardization. In December 2016, the NIST

[†]Corresponding Author. 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).

announced a call for proposals for PQC standardization [NIS16]. In contrast to the Advanced Encryption Standard (AES) and Secure Hash Algorithm version 3 (SHA-3) competitions, which selected a single algorithm, the NIST aims to recommend several PQC algorithms [NIS97, NIS07]. In the first-round submissions, sixty-nine proposals on public key encryption, key establishment, and digital signature algorithms were accepted. In the following second-round, twenty-six candidates have been survived [NIS19], and seven candidates are code-based cryptographic algorithms.

Code-based cryptography is based on coding theory, which aims to detect and correct errors on transmitted data through a noisy channel. McEliece proposed the first code-based PKC based on binary Goppa codes [McE78]. The security of the cryptosystem is based on the difficulty of the Syndrome Decoding (SD) problem and the Goppa Code Distinguishing (GCD) problem [BMvT78, FGO$^+$11]. The main drawback of the original McEliece cryptosystem is the large size of its public key. For the 80-bit security level, the public key size of the McEliece cryptosystem requires 460,647 bits, approximately 450 times larger than the one of the RSA cryptosystem. To reduce the public key size, several variants of the McEliece cryptosystem have been proposed, by replacing the Goppa codes of the McEliece cryptosystem with other efficient codes, for example, generalized Reed-Solomon (GRS), low-density parity-check (LDPC), and moderate-density parity-check (MDPC) codes [Nie86, BS08, MTSB13, BCS13, Cho16, Cho17].

Niederreiter [Nie86] proposed a variation of the McEliece cryptosystem, using a parity-check matrix as a secret key instead of a generator matrix, and its security has been proven to be equivalent to that of the McEliece cryptosystem [LDW94]. Niederreiter used the GRS code, but it can leak information much more easily than the binary Goppa code [SS92]. Biswas and Sendrier proposed a hybrid McEliece encryption scheme [BS08], reducing the size of public key by using a row echelon form of a generator matrix. Bernstein *et al.* proposed a key encapsulation mechanism (KEM)/data encapsulation mechanism (DEM) called McBits [BCS13], using the Niederreiter cryptosystems as the underlying scheme. McBits provides an additive fast Fourier transform (FFT) decoding algorithm for fast root and syndrome computations, and also a countermeasure against cache-timing attacks by utilizing a constant-time implementation. Chou further improved performance of McBits [Cho17] with a constant-time implementation for key generation and encryption and internal parallelism for decryption.

The MDPC and quasi-cyclic MDPC (QC-MDPC) codes, in particular, have recently received extensive attention due to the smaller key sizes and efficiency in terms of computational complexity. Misoczki *et al.* [MTSB12] proposed two variants of the McEliece cryptosystem called MDPC-McEliece, employing the MDPC and QC-MDPC codes to realize smaller key size. For the 80-bit security level, the public key of QC-MDPC McEliece requires only 4801 bits. Chou proposed a variant of the hybrid (KEM/DEM) Niederreiter encryption scheme using QC-MDPC codes called QcBits [Cho16].

Kocher first presented side-channel attacks (SCAs) [Koc96], which enable to recover secret credentials, i.e. cryptographic keys, by analyzing side-channel information such as execution time, power consumption, electromagnetic emission, and photonic emission, when cryptographic algorithms are running on devices. Such side-channel attacks include timing attack (TA), simple power analysis (SPA), differential power analysis (DPA), correlation power analysis (CPA), and profiling attack [MOP07].

Strenzke *et al.* [STM$^+$08] first proposed a SCA against the McEliece cryptosystem. They presented a TA on the degree of error locator polynomial in the Patterson algorithm [Pat75] exploiting the fact that the difference in computation time depends on the polynomial degree. Other types of TAs against the McEliece have been followed as in [SSMS09, Str10, Str11, Str13, BCDR17]. Strenzke *et al.* [STM$^+$08] also described power analysis against the parity-check matrix of McEliece key generation. Various SPAs and DPAs against the McEliece cryptosystem can be found in [HMP10, MSSS11, vMG14b, CEvMS15a, PRD$^+$15, PRD$^+$16, FGH16], and fault injection attacks in [CD10, Str11].

There have been, in particular, several SCA results against cryptosystems based on QC-MDPC code. Chou suggested a constant-time implementation for QC-MDPC code-based cryptography to mitigate TAs [Cho16]. This countermeasure was later found to become vulnerable to a DPA in private syndrome computation, as described by Rossi *et al.* [RHHM17]. The proposed attack, however, still could not completely recover accurate secret indices, requiring further solving linear equations to obtain entire secret information.

Our Contributions. The main contributions of this paper can be summarized as below.

1. **Enhancing existing multiple-trace attack on TA countermeasure**

   We propose a multiple-trace attack on the `constant-time multiplication` introduced by Chou for secure syndrome computation [Cho16]. Contrary to the attack results by Rossi *et al.* [RHHM17], we demonstrate that our attack recovers entire secret indices using multiple traces, eliminating the need of additionally solving linear equations. Previously, it was not even feasible to solve such equations with target cryptosystems running in 64-bit processor devices.

2. **Proposing a novel single-trace attack**

   The proposed single-trace attack allows to recover secret indices even when using ephemeral keys or DPA countermeasures [RHHM17, CEvMS15b]. In particular, if a processor only provides single bit shift instructions, it is possible to find the whole bits of secret indices. Furthermore, even if processors do not provide single bit shift instructions, we can extract substantial parts of secret indices. The proposed attack exploits the fact that rotation is always carried out, and also that the *mask* value as determined by the value of the secret bit is used to obtain accurate results. Hence, our attack can make the latest countermeasures proposed for secure private syndrome computation obsolete.

3. **Case study: NIST round 2 QC code-based cryptography**

   The BIKE and LEDAcrypt are constructed using QC-MDPC and QC-LDPC codes, respectively, and they are the second-round candidates of the NIST PQC standardization. Since syndrome computations of these two schemes were not designed to resist SCAs, we assume that the countermeasures [Cho16, RHHM17, CEvMS15b] are applied to remove each of TA and DPA vulnerability. Our experiment results show that these two schemes may become vulnerable to the proposed multiple/single-trace attacks when they use long-term key pairs. These schemes may become vulnerable to our single-trace attack even when using ephemeral keys.

The experiment makes use of power consumption traces measured from ChipWhisperer-Lite XMEGA (8-bit processor) and ChipWhisperer UFO STM32F3 (32-bit processor) target boards [Inca, Incb]. The proposed multiple-trace attack with 50 traces collected at 7.38 MS/s sampling rate allows to recover the whole bits of secret indices. The experiment, only using single-trace, describes how to recover the whole bits of secret indices with a processors providing single bit shift instruction, and also to extract substantial part of secret indices even with a processor not providing such single bit shift instructions. An attack flow chart can be found as in Figure 1.

Organization. The rest of this paper is organized as follows. In Section 2, we briefly describe the basics of coding theory and describe the literature for SCAs on QC-MDPC code-based cryptography. In Section 3, we then explain our target algorithm and recent DPA results. In Section 4 and Section 5, we propose a multiple-trace attack and a single-trace attack with experiment results. We then describe how the proposed attacks could be

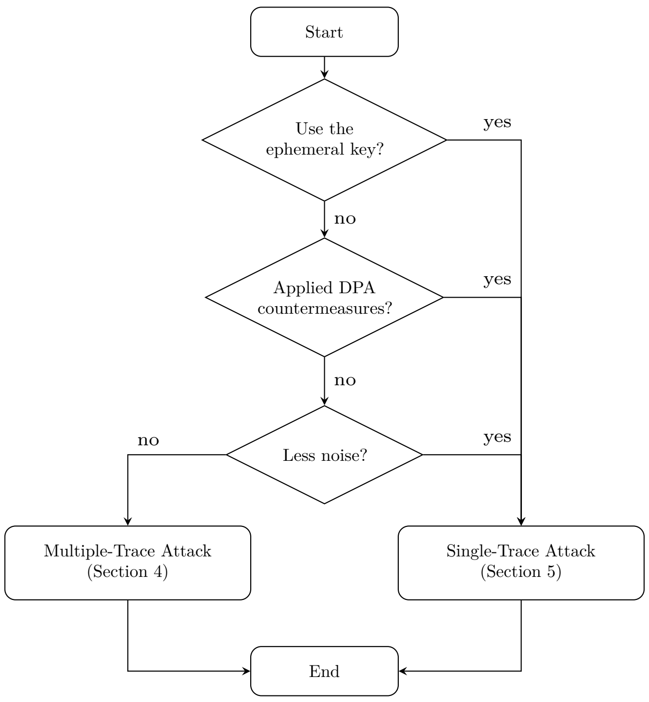

Start
Use the ephemeral key?
yes
no
Applied DPA countermeasures?
yes
no
Less noise?
no
yes
Multiple-Trace Attack (Section 4)
Single-Trace Attack (Section 5)
End

Figure 1: Flowchart of our proposed attack

applied to other QC-MDPC/LDPC code-based cryptography in Section 6. We finally give a conclusion in Section 7.

2 Preliminaries

2.1 Basics of Coding Theory

Code-based cryptography is based on the decoding problem of random error-correcting codes, i.e. Syndrome Decoding, which is known to be NP-hard [BMvT78]. In other words, it is based on the difficulty of finding the closest codeword x to a given $y \in \mathbb{F}_q^n$, assuming that there is a unique closest codeword. Table 1 shows the definition of notations used in this paper.

Definition 1. [Linear Code] An (n, k)-linear code $\mathcal{C}$ of length n and dimension k over a field $\mathbb{F}_q$ is a k-dimensional vector subspace of $\mathbb{F}_q^n$.

Definition 2. [Generator Matrix] A $k \times n$ matrix G of rank k is a generator matrix for a (n, k)-linear code $\mathcal{C}$ if $\mathcal{C} = \{mG \mid m \in \mathbb{F}_q^k\}$, i.e. the k rows of the matrix G span code $\mathcal{C}$.

Definition 3. [Parity-Check Matrix] An $(n-k) \times n$ matrix H is a parity-check matrix for $\mathcal{C}$ if $Hc^\intercal = 0$ for all $c \in \mathcal{C}$, that is, the codewords are all the vectors in the right null space of H.

Definition 4. [Hamming Weight] The Hamming weight of a vector $x \in \mathbb{F}_q^n$ is the number of all its non-zero components;

$$wt(x) = \#\{i \in [0, n-1] \mid x_i \neq 0\}.$$

Table 1: Notations

Notation	Description
$\mathbb{F}_q$	the finite field of order q
n	the length of the codeword
k	the dimension of the code
r	the co-dimension of the code
w	the Hamming weight (or simply weight) of a vector
t	the number of errors that can be decoded
c	codeword $c = (c_0, c_1, \cdots, c_{n-1}) \in \mathbb{F}_q^n$
$\mathcal{C}$	a linear code, which is a k-dimensional subspace of $\mathbb{F}_q^n$
g_i	one of k linearly independent row vectors $g_i \in \mathbb{F}_q^n$, where $0 \leq i < k$
G	a $k \times n$ generator matrix of $\mathcal{C}$, whose rows are g_i vectors
H	a $(n-k) \times n$ parity-check matrix of $\mathcal{C}$
s	a syndrome of a received vector $e \in \mathbb{F}_q^n$ is $s = He^\intercal \in \mathbb{F}_q^{n-k}$
$\mathcal{R}$	the cyclic polynomial ring $\mathbb{F}_q[x]/\langle x^r - 1\rangle$

Definition 5. [**Hamming Distance**] The Hamming distance between two vectors $x, y \in \mathbb{F}_q^n$ is the number of components in which they differ;

$$dist(x, y) = \#\{i \in [0, n-1] \mid x_i \neq y_i\}.$$

Definition 6. [**Minimum Distance**] The minimum distance of a linear code $\mathcal{C}$ is

$$d(\mathcal{C}) = \min\{dist(x, y) \mid x, y \in \mathcal{C}, x \neq y\} = \min\{wt(x) \mid x \in \mathcal{C}, x \neq 0\}.$$

The minimum distance gives the smallest number of errors needed to change one codeword into another. Therefore, it induces the error correction capability of the linear code $\mathcal{C}$. If the code can correct up to t errors, and changes are made at t or fewer places in a codeword c, then the closest codeword is still c. In coding theory, if the errors occur less than half the $d(\mathcal{C})$, it could be corrected. Namely, the linear code $\mathcal{C}$ can correct up to t errors if $d(\mathcal{C}) \geq 2t + 1$.

Definition 7. [**Circulant Matrix**] An $r \times r$ matrix is a circulant matrix if its rows are successive cyclic shifts of its first one. The top row (or the leftmost column) of a circulant matrix is the generator of the circulant matrix.

Definition 8. [**Quasi-Cyclic Matrix**] An $r \times n$ matrix $M = [M_0 \mid M_1 \mid \cdots \mid M_{n_0-1}]$ where $n = n_0 r$ is a QC matrix if the submatrices $M_0, M_1, \cdots, M_{n_0-1}$ are $r \times r$ circulant matrices.

Definition 9. [**Quasi-Cyclic Code**] An (n, k)-linear code $\mathcal{C}$ of length $n = n_0 r$, dimension $k = k_0 r$, and co-dimension $r = n - k$ is a QC code if every cyclic shift of a codeword by r positions results in another codeword in $\mathcal{C}$. The n_0 is called the index.

There is a ring isomorphism denoted as φ between the $r \times r$ circulant matrices and the quotient polynomial ring $\mathcal{R} = \mathbb{F}_q[x]/\langle x^r - 1\rangle$. Thus, a circulant matrix A whose first row is $(a_0, \cdots, a_{r-1})$ is mapped to the polynomial $\varphi(A) = a_0 + a_1 x + \cdots + a_{r-1} x^{r-1}$, and the (n, k)-QC code can be viewed as a cyclic code over the ring $\mathcal{R} = \mathbb{F}_q[x]/\langle x^r - 1\rangle$.

Definition 10. [**QC-MDPC/LDPC Code**] An (n, r, w)-linear code $\mathcal{C}$ of length $n = n_0r$, dimension $k = k_0r$, and co-dimension $r = n - k$ admitting a parity-check matrix H with constant row weight w is a QC-LDPC or MDPC code. LDPC and MDPC codes only differ in the row weight w. If the code is defined by a parity-check matrix H with constant row weight $w = O(\sqrt{n\log(n)})$, it is a MDPC code; while LDPC codes have small constant row weights, usually less than 10. The parity-check matrix H of QC-MDPC code is $(n-k) \times n$ QC matrix.

2.2 Side-Channel Attacks on QC-MDPC Code-Based Cryptography

Maurich *et al.* [vMG14b] presented SPAs on QC-MDPC McEliece cryptosystem, i.e. a message recovery attack and a private key recovery attack. For a private key recovery, they exploited the fact that different patterns of power consumption are observed depending on whether the conditional branch instruction is executed or not, when generating the next row of the private key in the syndrome computation. They presented experiment results based on two types of software implementations, i.e. AVR and ARM. They also proposed a constant-time implementation as a countermeasure using the ARM Thumb-2 assembly language. More specifically, they adopted the *mask* value, which is either zero or all bits are 1, and the logical AND instruction to choose which data to use. We classify the property used in the attacks as follows.

Property 1. *If an algorithm behaves irregularly according to the secret value, then the algorithm is vulnerable to simple power analyses (or timing attacks).*

Chen *et al.* [CEvMS15a] presented a horizontal DPA on the QC-MDPC McEliece cryptosystem, which is a private key recovery attack on the asymmetric decryption algorithm using the chosen ciphertexts. They successfully recovered substantial parts of the private key by a DPA during syndrome computation and key rotation. They make use of the public key to recover the whole private key or to correct remaining errors using an algebraic step. Their attack target was the field programmable gate array (FPGA) implementation presented at DATE 2014 [vMG14a]. Since hardware implementations operate in parallel, they applied the chosen ciphertext DPA. They also suggested a threshold implementation based on boolean masking as a countermeasure [CEvMS15b]. The further analysis and countermeasure are also proposed [CEvMS16]. We classify the property used in the attacks as follows.

Property 2. *If an algorithm uses a fixed secret k, and if it is possible to calculate hypothetical intermediate states $v_{i,j} = f(d_i, k_j)$ for all $\mathcal{D}$ known values d_i and for all $\mathcal{K}$ candidates k_j of k, then the algorithm is vulnerable to differential power analyses (or correlation power analyses). At this time, $\mathcal{K}$ should be small enough so that all hypotheses $v_{i,j}$ can be exhausted.*

Chaulet *et al.* [CS16] discussed that variable time decoders, such as bit flipping (BF) algorithm, may leak partial information. Since the number of iterations of the algorithm depends on the error pattern as well as the parity-check matrix, the algorithm may leak information about a private key and consequently allow a successful TA. They thus proposed minimizing the number of iterations by adapting threshold values as a function of the syndrome weight to make a constant-time decoder. This attack is based on Property 1.

3 Related Works

3.1 QcBits: Constant-Time Implementation of QC-MDPC Decoding

QcBits, proposed by Chou [Cho16], is the constant-time implementation of QC-MDPC code-based cryptography to mitigate TAs. $H \in \mathbb{F}_2^{r\times n}$ $(n = 2r)$ is used as a parity-check matrix. The random parity-check matrix $H = [H_0 \mid H_1]$ is a private key with $H_0, H_1 \in \mathbb{F}_2^{r\times r}$ and row weight of w. QcBits uses $r = 4801, w = 90$, and $t = 80$ for the 80-bit security.

$$H_0 = \begin{bmatrix} H^{(0)}_{0,0} & H^{(0)}_{0,1} & \cdots & H^{(0)}_{0,r-1} \\ H^{(0)}_{1,0} & H^{(0)}_{1,1} & \cdots & H^{(0)}_{1,r-1} \\ \vdots & \vdots & \ddots & \vdots \\ H^{(0)}_{r-1,0} & H^{(0)}_{r-1,1} & \cdots & H^{(0)}_{r-1,r-1} \end{bmatrix}, H_1 = \begin{bmatrix} H^{(1)}_{0,0} & H^{(1)}_{0,1} & \cdots & H^{(1)}_{0,r-1} \\ H^{(1)}_{1,0} & H^{(1)}_{1,1} & \cdots & H^{(1)}_{1,r-1} \\ \vdots & \vdots & \ddots & \vdots \\ H^{(1)}_{r-1,0} & H^{(1)}_{r-1,1} & \cdots & H^{(1)}_{r-1,r-1} \end{bmatrix}.$$

Since H_0 and H_1 are circulant matrices, i.e. $H^{(k)}_{(i+1) \bmod r,\ (j+1) \bmod r} = H^{(k)}_{i,j}$, where $k = 0, 1$ and $0 \le i, j \le r-1$; the first row of H can represent the whole matrix. An array of indices in $I_0 = \{j \mid H^{(0)}_{0,j} = 1\}$ and an array of indices in $I_1 = \{j \mid H^{(1)}_{0,j} = 1\}$ are enough to represent H. Then, a syndrome of a vector $c = [c_{(0)} \mid c_{(1)}] \in \mathbb{F}_2^n$, i.e. $Hc^\intercal$, is calculated by

$$Hc^\intercal = [H_0 \mid H_1] \begin{bmatrix} c^\intercal_{(0)} \\ c^\intercal_{(1)} \end{bmatrix} = H_0 c^\intercal_{(0)} + H_1 c^\intercal_{(1)},$$

$$H_0 c^\intercal_{(0)} = \sum_{i\in I_0} R_i(c_{(0)})^\intercal,\ H_1 c^\intercal_{(1)} = \sum_{i\in I_1} R_i(c_{(1)})^\intercal,$$

where $R_i(c_{(k)})$ is an i-bit left rotation of $c_{(k)}$. For any vector $a = (a_0, \cdots, a_{r-1}) \in \mathbb{F}_2^r$, there exists a ring isomorphism φ that maps a to the polynomial $\varphi(a) = a_0 + a_1 x + \cdots + a_{r-1}x^{r-1} \in \mathbb{F}_2[x]/\langle x^r - 1\rangle$. Thus, $c_{(k)}$ can be considered to be a polynomial, and $R_i(c_{(k)})$ can be calculated by the multiplication $x^d c_{(k)}$ in $\mathbb{F}_2[x]/\langle x^r - 1\rangle$, where $d = r - i$. For the remainder of this paper, $c_{(k)}$ is considered as a polynomial element. Here, the parity-check matrix H is the private key. Chou then suggested a **`constant-time multiplication`** $x^d c_{(k)}$, as shown in the Algorithm 1. This allows secure private syndrome computation $Hc^\intercal$ as a countermeasure against a TA.

Since $0 \le d \le r-1$, the binary representation of d is $(d_{l-1}, d_{l-2}, \cdots, d_0)_2$, where $l = \lceil \log_2(r-1) \rceil$. Then, one can calculate rotated intermediate values using W-bit word unit rotation for d_j from d_{l-1} to $d_{\log_2 W}$. A rotation by 2^j-bit, i.e. a power-of-2 shifts operation, can be calculated by a rotation by $2^{j-\log_2 W}$ words unit, where $l-1 \le j \le \log_2 W$. To make it perform in constant-time, the rotation is always carried out independent of d_j value. Then, one of the unrotated vector and the rotated vector is chosen according to the d_j value. The author implemented the algorithm using the $mask$ value, which is zero when $d_i = 0$, and all the bits are 1 when $d_i = 1$. The $\neg mask$ value is also used, where $\neg$ refers to negation. Therefore, the result is obtained as follows:

$$(\text{rotated value} \ \ \& \ \ mask) \oplus (\text{unrotated value} \ \ \& \ \ \neg mask).$$

The Algorithm 1 shows a simplified algorithm scheme, detailed algorithm scheme and toy example are shown in Appendix A. When the bit length of the word W is 8, the $mask$ value is as shown below:

$$mask = \begin{cases} \texttt{0x00} & , if\ d_i = 0; \\ \texttt{0xff} & , if\ d_i = 1. \end{cases}$$

The variable us indicates how many words are rotated to the left. Thus, the result w is given by:

$$w[j] \leftarrow (v[(j + us) \bmod L] \ \ \& \ \ mask) \oplus (v[j] \ \ \& \ \ \neg mask), \tag{1}$$

Algorithm 1 Constant-Time Multiplication in $\mathbb{F}_2[x]/\langle x^r - 1\rangle$ (refer to [Cho16])

```
  Input : $d = (d_{l-1}, \cdots, d_0)_2$, $0 \le d \le r-1$, $c_{(k)} = (c_{L-1}, \cdots, c_0)_{2^W}$, $L = \lceil r/W \rceil$
  Output : $x^d c_{(k)}$
 1: $v \leftarrow 0$, $w \leftarrow c_{(k)}$
 2: for $i = l-1$ down to $\log_2 W$ do    ▶ word unit rotation is from 2 to 13
 3:    $d_i \leftarrow (d \gg (l-1-i))\ \&\ 1$
 4:    $mask \leftarrow 0 - d_i$
 5:    $us \leftarrow 1 \ll (i - \log_2 W)$
 6:    $ptr \leftarrow v$, $v \leftarrow w$, $w \leftarrow ptr$
 7:    for $j = 0$ up to $L-1-us$ do
 8:       $w[j] \leftarrow (v[j+us]\ \&\ mask) \oplus (v[j]\ \&\ \neg mask)$
 9:    end for
10:    for $j = 1$ up to $us$ do
11:       $w[j+L-1-us] \leftarrow (v[j-1]\ \&\ mask) \oplus (v[j+L-1-us]\ \&\ \neg mask)$
12:    end for
13: end for
14: $low \leftarrow d\ \&\ ((1 \ll \log_2 W) - 1)$    ▶ bit rotation is from 14 to 22
15: $high \leftarrow W - low$
16: $tmp \leftarrow w[0]$
17: for $j = 0$ up to $L-2$ do
18:    $w[j] \leftarrow w[j] \gg low$
19:    $w[j] \leftarrow w[j] \mid (w[j+1] \ll high)$
20: end for
21: $w[L-1] \leftarrow w[L-1] \gg low$
22: $w[L-1] \leftarrow w[L-1] \mid (tmp \ll high)$
23: Return $w$
```

for j from 0 to $L-1$, where $L = \lceil r/W \rceil$. Based on the published source code of [Cho16], Equation (1) is calculated by dividing into two parts: steps 7 to 9 and steps 10 to 12 of the Algorithm 1. In those parts, one selects the rotated value $v[j+us]$ when $d_i = 1$ and the unrotated value $v[j]$ when $d_i = 0$.

A sequence of logical instructions is utilized for j from $\log_2 W - 1$ to 0, i.e. the shifts inside the units, as shown in steps 14 to 22 of the Algorithm 1. In this paper, we defined the power-of-2 shifts operation for $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$ as a **`word unit rotation`** and the shifts operation inside the units for $(d_{\log_2 W-1}, \cdots, d_1, d_0)$ as a **`bit rotation`**.

3.2 A Side-Channel Assisted Cryptanalytic Attack on QcBits

Rossi *et al.* [RHHM17] proposed a DPA on QcBits with the Property 2 described in Section 2, targeting private syndrome computation, i.e. **`constant-time multiplication`** $x^d c_{(k)}$. The authors analyzed power consumption traces acquired while the results of $x^d c_{(k)}$ were stored in memory. In software implementations, the power consumption is affected by the Hamming weight of the intermediate value. They thus applied the DPA based on the leftmost bit of each rotated result $x^d c_{(k)}$ calculated by estimating d. Since from the $(i+1)$ to $(i+W)$-th bits are saved into the same register, there will be W candidates for d. Hence, it is impossible to find accurate secret indices, merely reducing the candidates. For each secret index d, there are 8 candidates in an 8-bit processor and 64 candidates in a 64-bit processor. For the full recovery of the secret indices, it is thus required to solve linear equations. As shown in the Table 2, for 128-bit security, the linear equations can be solved within a reasonable time on 8-bit, 16-bit, and 32-bit processors. It is, however, not feasible on 64-bit processors.

Table 2: Approximate solving times of linear equations according to the operation unit of the processors (in SAGE on one core)

	8-bit	16-bit	32-bit	64-bit
80-bit security	0.4 seconds	15 seconds	16 hours	$\approx$ 530 years
128-bit security	2 seconds	4 minutes	$\approx$ 7 days	$\approx$ 790,000 years

The authors also proposed a codeword masking, i.e. adding a random codeword prior to the syndrome computation. The result of the syndrome calculation remains unchanged due to the fact that QC-MDPC codes are linear, and they discussed that the proposed countermeasure effectively removes the information leak against DPAs.

$$H \cdot ((c \mid 0) \oplus c_m)^\intercal = H \cdot (c \mid 0)^\intercal \oplus H \cdot c_m^\intercal = H \cdot (c \mid 0)^\intercal. \tag{2}$$

Similarly, as a DPA countermeasure, Chen *et al.* [CEvMS15b] proposed a masked syndrome computation, splitting H into two shares H_m and M, where M is a matrix for masking, as below.

$$H \cdot (c \mid 0)^\intercal = (H_m \oplus M) \cdot (c \mid 0)^\intercal = (H_m \cdot (c \mid 0)^\intercal) \oplus (M \cdot (c \mid 0)^\intercal). \tag{3}$$

In the subsequent sections, we propose multiple- and single-trace attacks on the **`constant-time multiplication`** $x^d c_{(k)}$ which entirely recover the secret index d eliminating the need of solving linear equations. If a processor only provides single bit shift instructions, it is possible to find the accurate secret index d using a single trace even when DPA countermeasures, e.g. Equation (2) and Equation (3), are applied. If a processor provides multiple bit shift operations, W candidates for d will be derived, similarly to the results obtained in [RHHM17].

4 Proposed Multiple-Trace Attack on Constant-Time Multiplication for Syndrome Computation

In this section, we propose a multiple-trace attack on the **`constant-time multiplication`** $x^d c_{(k)}$. We show that it is possible to completely recover secret indices using multiple traces. In contrast to the attack presented in Subsection 3.2 that has W candidates for each d, our attack can extract the entire secret index $d = (d_{l-1}, d_{l-2}, \cdots, d_0)_2$; solving linear equations is not required anymore. Based on the structure of the **`constant-time multiplication`** shown in the Algorithm 1, we divide the attack position into two parts to find d: the **`word unit rotation`** to find $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$ (Subsection 4.1), and the **`bit rotation`** to find $(d_{\log_2 W-1}, \cdots, d_1, d_0)$ (Subsection 4.2). Since software implementation is considered here, the power consumption is assumed to be affected by the Hamming weight of the intermediate value [MOP07].

4.1 Multiple-Trace Attack on the Word Unit Rotation

We here describe the multiple-trace attack methodology on the **`word unit rotation`** and present our experiment results. For the attack methodology construction, we first categorize properties of the **`word unit rotation`**. We then describe how to find $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$ based on these properties.

Attack Methodology. The following operation is executed to mitigate TAs as described in Subsection 3.1.

$$w[j] \leftarrow (v[j+us] \ \& \ mask) \oplus (v[j] \ \& \ \neg mask)$$

Accordingly, the $v[j+us]$ and $v[j]$ values are always loaded. In this step, one selects the rotated value $v[j+us]$ when $d_i = 1$ and the unrotated value $v[j]$ when $d_i = 0$. Therefore, when the bit length of the word W is 8,

$$w[j] = \begin{cases} (v[j+us] \ \& \ \texttt{0x00}) \oplus (v[j] \ \& \ \texttt{0xff}) = v[j] & , \ if \ d_i = 0; \\ (v[j+us] \ \& \ \texttt{0xff}) \oplus (v[j] \ \& \ \texttt{0x00}) = v[j+us] & , \ if \ d_i = 1. \end{cases}$$

This is shown in the steps 7 to 9 of the Algorithm 1. The index of the array $v[*]$ to be saved in $w[j]$ is determined based on the d_i value. We thus define the following two properties.

New Property 1. *The mask value is $0 - d_i$; therefore, it is* `0x00` *when $d_i = 0$. Consequently, in the steps 7 to 9 of the Algorithm 1, it is*

$$w[j] \leftarrow (v[j+us] \ \& \ \texttt{0x00}) \oplus (v[j] \ \& \ \texttt{0xff}).$$

on an 8-bit processor, i.e. $v[j]$ is saved to $w[j]$. Thus, $v[j]$ is loaded and saved, but $v[j+us]$ is only loaded. Contrariwise, when $d_i = 1$, the mask value is `0xff` *on an 8-bit processor. Consequently, in the steps 7 to 9 of the Algorithm 1, it is*

$$w[j] \leftarrow (v[j+us] \ \& \ \texttt{0xff}) \oplus (v[j] \ \& \ \texttt{0x00}),$$

i.e. $v[j+us]$ is saved to $w[j]$. Thus, $v[j+us]$ is loaded and saved, but $v[j]$ is only loaded.

New Property 2. *If $d_i = 0$, then the unrotated value is chosen, i.e. $v[j]$ is saved to $w[j]$, which has the same index. Contrariwise, when $d_i = 1$, the rotated value is chosen, i.e. $v[j+us]$ is saved to $w[j]$, which has a different index.*

During the algorithm execution, the specific power consumption pattern can be observed depending on the intermediate values. Thus, the power consumption P_{total} at each point can be modeled as the sum of a data-dependent component P_{data} and Gaussian noise P_{noise}, i.e. $P_{total} = P_{data} + P_{noise}$ [MOP07]. Since we assume that the Hamming weight of the intermediate value contributes to the power consumption, we can remodel P_{total} as $\epsilon \cdot wt(data) + P_{noise}$, where ϵ is a constant, i.e. there is a linear relationship between P_{total} and $wt(data)$. It is thus possible to specify the positions where the $v[j]$ value was used by calculating the Pearson correlation coefficient between the Hamming weight of the $v[j]$ values and power consumption traces. That is, the positions with high correlations are related to the operation using the intermediate value $v[j]$.

If $d_i = 0$, then the *mask* value is `0x00`; therefore, the power consumption with respect to the $v[j]$ value occurs sequentially **twice** in the steps 7 to 9 of the Algorithm 1 according to the New Property 1. Contrariwise, when $d_i = 1$, the *mask* value is `0xff` on an 8-bit processor; therefore, the power consumption with respect to the $v[j]$ value occurs **once** in the steps 7 to 9 of the Algorithm 1. Thus, one can find d_i by identifying whether a high correlation occurs sequentially **twice** in the steps 7 to 9 of the Algorithm 1.

The value $w[j]$ based on d_{i+1} is the same with the value $v[j]$ based on d_i in the steps 2 to 13 of the Algorithm 1, where $\log_2 W \leq i < l - 1$. Thus, the power consumption related to the $v[j]$ value occurs sequentially **twice** in the $(j+1)$-th iteration of the steps 7 to 9 of d_{i+1}. Besides, based on the New Property 1 and the New Property 2, if $d_i = 0$, then the power consumption related to the $v[j]$ value occurs sequentially **twice** in the $(j+1)$-th iteration of the steps 7 to 9 of d_i. Consequently, the power consumption associated with the $v[j]$ value occurs sequentially **twice** in the same iteration where the loaded and saved operations are executed according to the prior key bits d_{i+1} when $d_i = 0$.

Algorithm 2 Multiple-Trace Attack on the Word Unit Rotation

```
Input : a trace set $T = \{T^1, \cdots, T^N\}$ and an input value set $C = \{c^1, c^2, \cdots, c^N\}$
Output : $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$
1: Calculate the correlation coefficient between $T$ and $C_0 = \{c^1[0], c^2[0], \cdots, c^N[0]\}$
2: if the high correlation occurs twice at the 1st iteration then      ▶ finding $d_{l-1}$
3:     $d_{l-1} \leftarrow 0$
4: else
5:     $d_{l-1} \leftarrow 1$
6: end if
7: for $i = l-2$ down to $\log_2 W$ do      ▶ finding $(d_{l-2}, \cdots, d_{\log_2 W})$
8:     if the high correlation occurs twice at the same position (iteration) with $d_{i+1}$ then
9:         $d_i \leftarrow 0$
10:    else
11:        $d_i \leftarrow 1$
12:    end if
13: end for
14: Return $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$
```

Otherwise, the power consumption related to the $v[j]$ value occurs sequentially **twice** in a different iteration from where the loaded and saved operations are executed based on the prior key bits d_{i+1} when $d_i = 1$. In other words, two peaks will occur at $2^{i-\log_2 W}$ words left-rotated position, i.e. $((j - 2^{i-\log_2 W}) \bmod L + 1)$-th iteration. Thus, one can find d_i by identifying whether a high correlation occurs sequentially **twice** in the same iteration as d_{i+1}. In other words, one can find d_i by identifying the position where the high correlation with the intermediate value $v[j]$ occurs sequentially **twice**.

It is possible to find $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$ using only the New Property 1. However, one has to chase the intermediate value determined by d_{i+1} when it is desired to find d_i, whereas, there is no need to chase the intermediate value if one make use of the New Property 2. Since the most significant bit can only be recovered based on the New Property 1, we combine the New Property 1 and the New Property 2 to construct an attack methodology.

Therefore, we recover the most significant bit d_{l-1} based on the New Property 1. Afterward, we recover the following bits $(d_{l-2}, \cdots, d_{\log_2 W})$ based on the New Property 2. Hence, we construct the attack flow, such as the Algorithm 2. In the step 1 of the Algorithm 2, T is a set of N traces and C_0 is a set of N input words $c^i[0]$, where each multiplication input value $c^i = (c^i[L-1], c^i[L-2], \cdots, c^i[0])$ and $1 \leq i \leq N$.

Consequently, it is possible to identify the position where the power consumption related to the set C_0 of input words occurs sequentially **twice** by calculating the correlation coefficient between T and C_0; the correlation coefficient represents the similarity between the two sets. Subsequently, the most significant bit d_{l-1} can be recovered. Similarly, by finding whether the power consumption related to the set C_0 occurs sequentially **twice** at the same position as d_{i+1} or not, we can recover $(d_{l-2}, \cdots, d_{\log_2 W})$ based on the New Property 2.

In the Algorithm 2, the steps 2 to 6 for finding d_{l-1} are based on the New Property 1, and the steps 7 to 13 for finding $(d_{l-2}, \cdots, d_{\log_2 W})$ are based on the New Property 2. Thus, $(d_{l-2}, \cdots, d_{\log_2 W})$ is extracted by the multiple-trace attack on the `word unit rotation`. We describe the attack methodology to find the last $\log_2 W$-bit, $(d_{\log_2 W-1}, \cdots, d_1, d_0)$, in Subsection 4.2.

Experiment results on an 8-bit processor. Our experiment shows that the position with high correlation with the intermediate value $v[0]$, i.e. each multiplication input, is different depending on the secret bit d_i value. Since the target board is equipped with an 8-bit processor, we set $r = 256$ as a toy example. Thus, when W is 8, $L = \lceil r/W \rceil = 32$, $l = \lceil \log_2(r-1) \rceil = 8$, and $d = (d_7, \cdots, d_1, d_0)_2$. We measured 500 power consumption traces at 7.38 MS/s sampling rate with the Algorithm 1 operating on a ChipWhisperer-Lite XMEGA target board. Figure 2 shows one of the power consumption traces. Since $\log_2 W = 3$, we can find $(d_7, d_6, d_5, d_4, d_3)$, and 50 traces are sufficient for the attack.

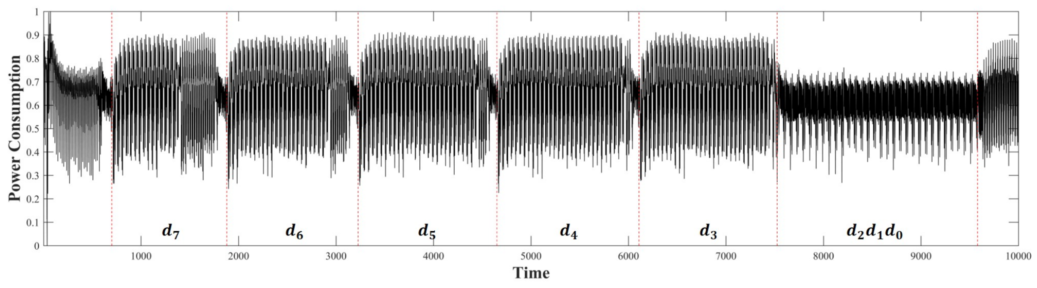
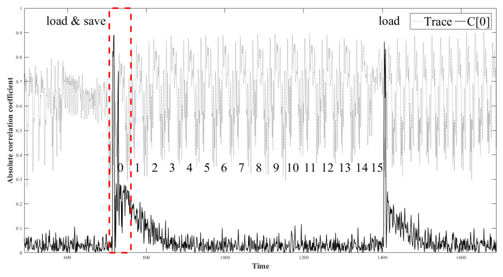
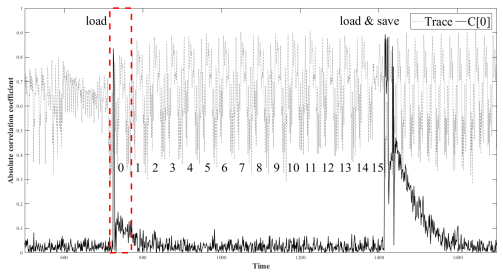


Figure 2: Power consumption trace of the constant-time multiplication ($W = 8$)



(a) $d_7 = 0$



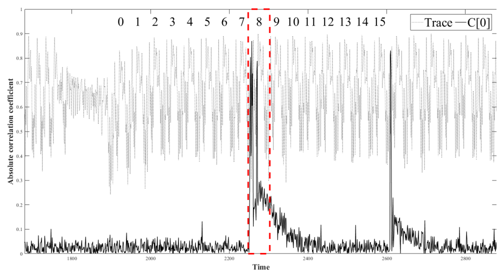
(b) $d_7 = 1$

Figure 3: Comparison of correlation coefficient values based on d_7 (New Property 1)

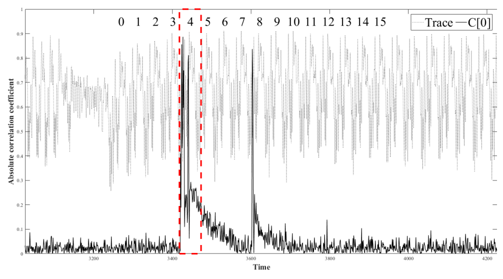
Figure 3 shows the experimental proof of the attack methodology to find the most significant bit d_{l-1} based on the New Property 1. The power consumption with respect to C_0 occurs sequentially **twice** in the 1st iteration of the steps 7 to 9 of the Algorithm 1 when $d_7 = 0$, as shown in Figure 3(a). Since $L = 32$, $l = 8$, and $\log_2 W = 3$, the steps 7 to 9 of the Algorithm 1 operate 16 times when d_7. Thus, the first 16 patterns in Figure 3(a) and Figure 3(b), marked from 0 to 15, are interesting domains. Contrariwise, the power consumption with respect to C_0 occurs **once** in the 1st iteration of the steps 7 to 9 of the Algorithm 1 when $d_7 = 1$, as shown in Figure 3(b).

Figure 3(b) and Figure 4 show the attack results of $d = (11101010)_2$. Each of the figures is a magnification of the computational portion of the corresponding bit of Figure 2. Figure 3(b) shows that the high correlation occurs only **once** in the steps 7 to 9 of the Algorithm 1 since d_7 is 1. Figure 4(a) shows that the high correlation occurs sequentially **twice** at a different position with d_7, because d_6 is 1. The same results can be observed in Figure 4(b) and Figure 4(d). In contrast, Figure 4(c) shows that the high correlation occurs sequentially **twice** at the same position with d_5, because d_4 is 0. Subsequently, one can find the accurate secret bits $(d_7, d_6, d_5, d_4, d_3)$ using the Algorithm 2. The attack methodology for finding the remaining bits (d_2, d_1, d_0) is described in Subsection 4.2.

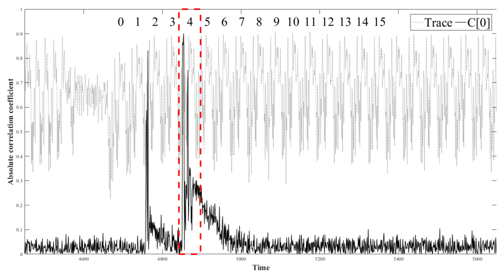
Experiment results on a 32-bit processor. The target board is equipped with a 32-bit processor, and we set the parameter $r = 4801$ for the 80-bit security. Since r is 4801 and



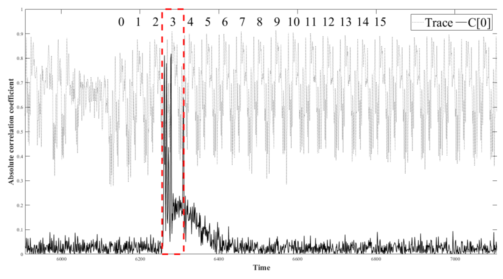
(a) $d_6 = 1$



(b) $d_5 = 1$



(c) $d_4 = 0$



(d) $d_3 = 1$

Figure 4: Finding d from d_6 to d_3 when $d = (11101010)_2$ (New Property 2)

W is 32, then $L = \lceil r/W \rceil = 151$, $l = \lceil \log_2(r-1) \rceil = 13$, and $d = (d_{12}, \cdots, d_1, d_0)_2$. We measured 500 power consumption traces at 7.38 MS/s sampling rate with the Algorithm 4 in Appendix A operating on a ChipWhisperer UFO STM32F3 target board. Figure 11 in Appendix B shows one of the power consumption traces. Since $\log_2 W = 5$, one can find $(d_{12}, d_{11}, d_{10}, d_9, d_8, d_7, d_6, d_5)$, and 50 traces are sufficient for the attack. The reader may refer to Appendix B.1 for a more detailed explanation.

4.2 Multiple-Trace Attack on the Bit Rotation

We now describe the multiple-trace attack methodology on the **`bit rotation`** and present our experiment results. To recover the remaining bits $(d_{\log_2 W-1}, \cdots, d_1, d_0)$, we apply a CPA based on the Property 2 in Section 2.

Attack Methodology. We guess W-bit values which are one words of the output $x^d c_{(k)}$ of the Algorithm 1. There is no candidate key, since we use a W-bit Hamming weight leakage model, in contrast with [RHHM17] which makes use of a 1-bit Hamming weight leakage model. One can calculate w, which is the result of the **`word unit rotation`** of the Algorithm 1, because $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$ can be found as described in Subsection 4.1. Thus, we only guess the *low* value from 0 to $W-1$ when we guess the leftmost word

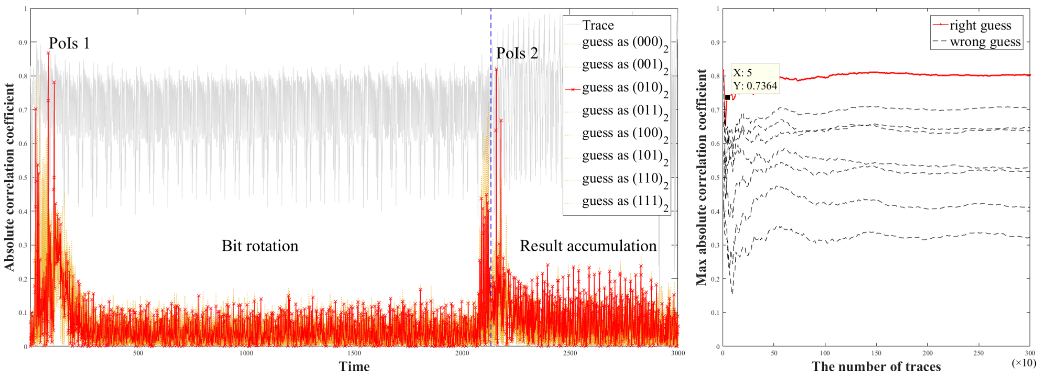
$$(w[0] \gg (low)) \mid (w[1] \ll (W - low)) \tag{4}$$

of the result of the **`bit rotation`** $x^d c_{(k)}$ of the Algorithm 1. At this point the *low* value and the last $\log_2 W$-bit value of d are the same.

After the **`bit rotation`** is performed, the result of Equation (4) is saved. Besides, after the Algorithm 1 execution, the result for calculating $H_0 c_{(0)}^\intercal$ or $H_1 c_{(1)}^\intercal$ is accumulated. Therefore, the result of Equation (4) is loaded. Accordingly, the power consumption associated with this intermediate value occurs in two places. We refer to these points as points of interest (PoI). Further, we mount a CPA using these two PoIs and find the last

$\log_2 W$-bit of d, i.e. $(d_{\log_2 W-1}, \cdots, d_1, d_0)$. The intermediate value is Equation (4) which is the leftmost word of $x^d c_{(k)}$.

Experiment results on an 8-bit processor. Here, we demonstrate that the last 3-bit of d can be found by an 8-bit CPA. Using the 8-bit CPA implies that we only use 8-bit of intermediate data, which does not denote the attack complexity. The attack complexity is 2^3 when we target an 8-bit processor since we only need to find the last $\log_2 8 = 3$-bit of d. It is 2^6 when we target a 64-bit processor since one has to find the last $\log_2 64 = 6$-bit of d. Thus, the attack is feasible. The measurement setup for power consumption traces is as described in Subsection 4.1, and two PoIs can be identified, as shown in Figure 5(a). Even if the CPA uses one of these PoIs can accurately derive the last 3-bit of d, i.e. (d_2, d_1, d_0). Figure 5(b) confirms that 50 traces are sufficient for the attack.



(a) Points of interest **(b)** 8-bit CPA results

Figure 5: Correlation power analysis results when $d = (11101010)_2$

Experiment results on a 32-bit processor. Since $\log_2 W = 5$, the attack complexity is 2^5, and one can find $(d_4, d_3, d_2, d_1, d_0)$. As with the previous cases, 50 traces are sufficient. An interested reader may refer to Appendix B.2 for a more detailed explanation.

4.3 Comparison with the Previous Attack

As described in Subsection 3.2, the attack suggested by Rossi *et al.* reduces to a certain number of candidates for each d but still requires to solve linear equations. Such a computation is even not feasible for 64-bit processors. Our multiple-trace attack, however, allows to recover all secret indices regardless of word size and security level. This becomes possible because we divided the attack position into two parts based on the structure of the **`constant-time multiplication`**. In particular, we categorize new properties of the **`word unit rotation`** and only need to guess the last $\log_2 W$-bit of d when we attack the **`bit rotation`**. Moreover, to the best of our knowledge, the attack presented in Subsection 3.2 requires approximately 200 power traces sampled at 96 MS/s; however, our attack only requires 50 traces sampled at 7.38 MS/s. We measured the power consumption traces from the same target board as used in Subsection 3.2, i.e. ChipWhipserer-Lite XMEGA.

5 Proposed Single-Trace Attack on Constant-Time Multiplication for Syndrome Computation

In this section, we propose a single-trace attack on the **`constant-time multiplication`** $x^d c_{(k)}$. We demonstrate that the proposed single-trace attack allows to extract the secret index $d = (d_{l-1}, d_{l-2}, \cdots, d_0)_2$, even when cryptosystems use ephemeral keys, or the DPA countermeasures [RHHM17, CEvMS15b] are applied. Hence, the proposed attack can make the latest countermeasures proposed for secure private syndrome computation obsolete. As done in the proposed multiple-trace attack, we divide the attack position into two parts to find d: the **`word unit rotation`** to find $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$ (Subsection 5.1), and the **`bit rotation`** to find $(d_{\log_2 W-1}, \cdots, d_1, d_0)$ (Subsection 5.2).

5.1 Single-Trace Attack on the Word Unit Rotation

We here describe the single-trace attack methodology on the **`word unit rotation`** and present the experiment results. To construct the attack methodology, we first categorize properties of the **`word unit rotation`**. We then propose how to find $(d_{l-1}, \cdots, d_{\log_2 W})$ based on these properties.

Attack Methodology. As described in Subsection 3.1, the *mask* value determined by the value d_i is used to check whether the rotated value is saved or not. Therefore, there exists a phase in extracting d_i bit from the l-bit secret index string $d = (d_{l-1}, d_{l-2}, \cdots, d_0)_2$ and saving it before performing the **`word unit rotation`**, such as in the step 3 of the Algorithm 1. Then, the values *mask* and $\neg$*mask* are computed and saved. Besides, when the steps 7 to 9 of the Algorithm 1 are executed, the values *mask* and $\neg$*mask* are loaded. Since, in software implementations, the power consumption depends on the Hamming weight of the intermediate value, it is possible to distinguish among the steps mentioned above and classify the power consumption properties of the target Algorithm 1 as follows:

s.1 $d_i \leftarrow (d \gg (l-1-i))\ \&\ 1$	▶ d_i is saved;
s.2 $mask \leftarrow 0 - d_i$	▶ *mask* is saved;
s.3 $\neg mask$ is calculated	▶ *mask* is loaded, $\neg$*mask* is saved;
s.4 $w[j] \leftarrow (v[j+us]\ \&\ mask) \oplus (v[j]\ \&\ \neg mask)$	▶ *mask* and $\neg$*mask* are loaded.

New Property 3. *The secret bit d_i is 0 or 1. Thus, if $d_i = 0$, the power consumption is associated with 0 when extracting and saving the d_i value. Likewise, if $d_i = 1$, then the power consumption is associated with 1.*

New Property 4. *The mask value is $0 - d_i$; therefore, it is* `0x00` *when $d_i = 0$, and the power consumption is related to 0. Contrariwise, when $d_i = 1$, the mask value is* `0xff` *on an 8-bit processor, and the power consumption is related to 8, which is the Hamming weight of the mask value.*

New Property 5. *The $\neg$mask value is 1's-complement of the mask value; therefore, it is the bitwise inversion value of the mask value. Consequently, in contrast to the New Property 4, it is* `0xff` *on an 8-bit processor when $d_i = 0$ and the power consumption is related to 8. Contrariwise, the power consumption is related to 0 when $d_i = 1$.*

We define d_i, *mask*, and $\neg$*mask* as the reference values for each property. Additionally, we define the New Property 3, the New Property 4, and the New Property 5 as key bit-dependent properties. Based on these key bit-dependent properties, power consumption traces can be classified into two groups, G_1 and G_2, depending on the d_i value. The

Algorithm 3 Single-Trace Attack on the Word Unit Rotation

```
  Input : A trace $T$
  Output : $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$
 1: for $i = l-1$ down to $\log_2 W$ do
 2:     Select points of interest $p_i$ of word unit rotation operation associated with $d_i$
 3: end for
 4: Classify $p_i$ into two groups, $G_1$ and $G_2$, using the $k$-means clustering algorithm
 5: Calculate the average values $AVG_1$ and $AVG_2$, respectively, of $G_1$ and $G_2$
 6: for $i = l-1$ down to $\log_2 W$ do
 7:     if $p_i \in G_1$ then              ▶ assume that $AVG_1 < AVG_2$
 8:         $d_i \leftarrow 1$             ▶ $d_i = 1$ when it follows the New Property 4
 9:     else
10:         $d_i \leftarrow 0$             ▶ $d_i = 0$ when it follows the New Property 4
11:     end if
12: end for
13: Return $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$
```

clustering algorithms, such as k-means, fuzzy k-means, or EM algorithms, then can be applied [Anz92].

After clustering, the average values AVG_1 and AVG_2 of each group G_1 and G_2, respectively, are calculated. Assuming that larger the Hamming weight requires the lower the power consumption, if AVG_1 is lower than AVG_2, d_i belonging to G_1 is 1 and that belonging to G_2 is 0, based on the New Property 3. The same results are obtained when we classify based on the New Property 4. In contrast, d_i belonging to G_1 is 0, and d_i belonging to G_2 is 1 when we classify based on the New Property 5. Hence, $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$ can be recovered by identifying the group that d_i belongs to. We further describe how to find the last $\log_2 W$-bit, i.e. $(d_{\log_2 W-1}, \cdots, d_1, d_0)$, in the following subsection.

The Algorithm 3 describes the attack flow. In the steps 1 to 3 of the Algorithm 3, we choose the PoIs associated with one of **s.1**, **s.2**, **s.3**, and **s.4**. The step 4 is the classification of the PoIs into two groups using the k-means clustering algorithm. Since G_1 and G_2 differ in the intermediate values that affect the power consumption, the distribution of G_1 is different from that of G_2. We thus can distinguish which group is associated with a certain intermediate value using the average values of each group, as shown in the steps 7 to 11 of the Algorithm 3. The reader may refer to experiment results for a detailed explanation.

Experiment results on an 8-bit processor. The experiment result demonstrate that the key bit-dependent properties are enough to extract the secret bit, d_i, using a single trace. The measurement setup for power consumption traces can be found in Section 4.1.

The PoIs can be identified by calculating the sum of squared pairwise t-differences (SOST) [GLP06] of the traces and then identifying the location of the information-leaking point, as shown in Figure 6. The SOST of two groups, G_1 and G_2, is calculated as below.

$$SOST = \left(\frac{E(G_1) - E(G_2)}{\sqrt{\frac{\sigma(G_1)^2}{\#G_1} + \frac{\sigma(G_2)^2}{\#G_2}}} \right)^2$$

$E(\cdot)$, $\sigma(\cdot)$, and # denote the mean, standard deviation, and number of elements, respectively. In Figure 6, the five points with the high SOST values are where the ***com*** operation, which yields a 1's complement to calculate the $\neg mask$ value, is performed. Figure 7 shows the distribution of points which have the highest SOST value, near 685 points of Figure 6. Two distributions are clearly distinguished: one is when $d_i = 0$, and the other is when

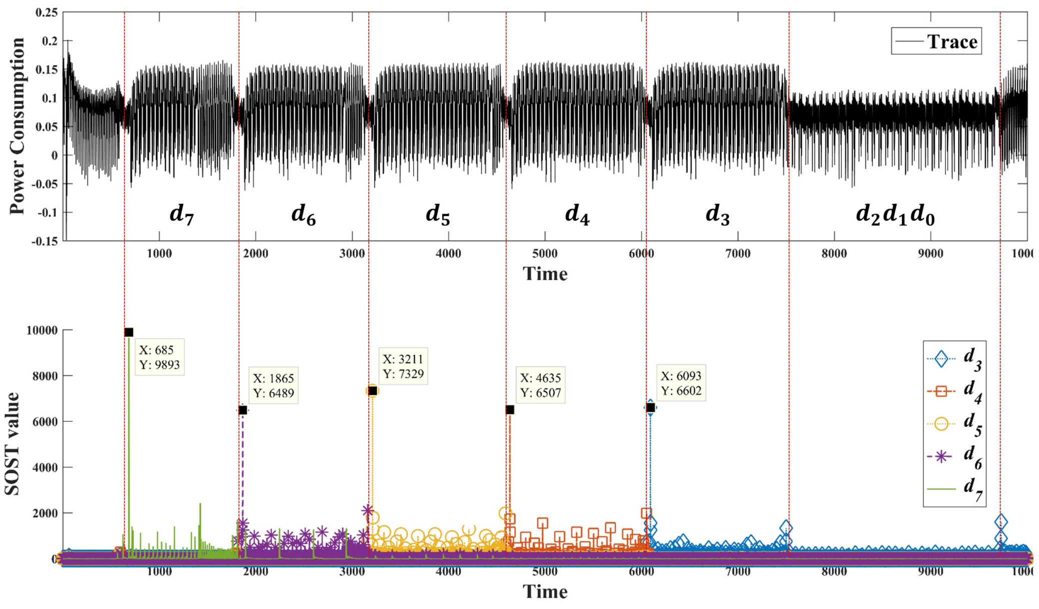


Figure 6: The power consumption trace and the SOST values between two groups, G_1 and G_2, of each d_i ($W = 8$)

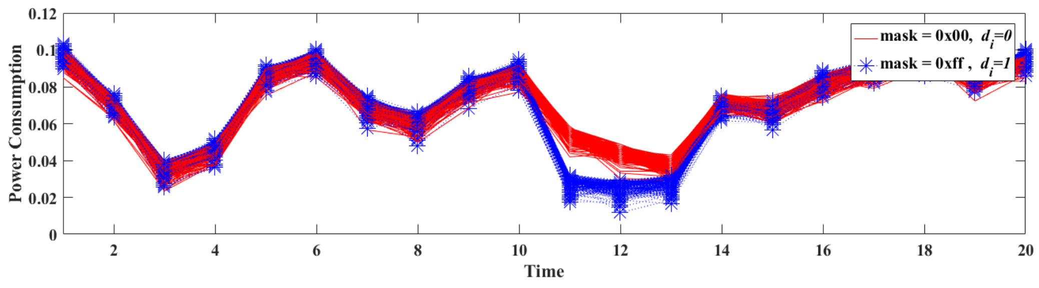


Figure 7: Distribution of PoIs which have the highest SOST value (New Property 4)

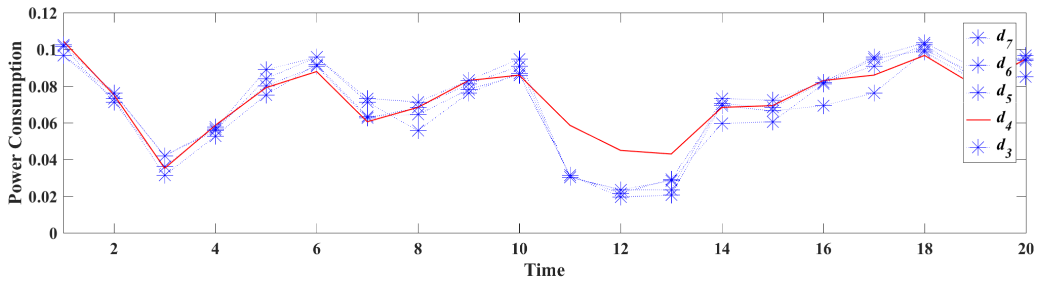


Figure 8: Finding d from d_7 to d_3 when $d = (11101010)_2$ (New Property 4)

$d_i = 1$. We use five points with the high SOST value as PoIs and select these points in the steps 1 to 3 of the Algorithm 3.

The average value for $d_i = 0$ is higher than the average value for $d_i = 1$. Therefore, if the AVG_2 is higher than the AVG_1 and d_i belongs to G_2, then d_i is 0. In contrast, if d_i belongs to G_1, it is 1. Figure 8 shows the attack results. Hence, the accurate secret bits

$(d_7, d_6, d_5, d_4, d_3)$ of indices can be found using the Algorithm 3.

Experiment results on a 32-bit processor. Since $W = 32$, the *mask* value is `0x00000000` or `0xffffffff`. One can find $(d_{12}, d_{11}, d_{10}, d_9, d_8, d_7, d_6, d_5)$ because $\log_2 W = 5$. The reader may refer to Appendix B.3 for the details.

5.2 Single-Trace Attack on the Bit Rotation

In this subsection, we describe the single-trace attack methodology on the **`bit rotation`** and discuss the results of the experiment. To recover the remaining bits, $(d_{\log_2 W-1}, \cdots, d_0)$, we apply a SPA based on the Property 1 presented in Section 2.

Attack Methodology. The most commonly used 8-bit AVR and 16-bit MSP430 processors only provide single bit shift instructions. Thus, a 1-bit right shift operation is repeated *low* times, and a 1-bit left shift operation is repeated *high* times in the steps 17 to 22 of the Algorithm 1. A SPA thus allows to identify the number of 1-bit left shift operations. Since the *low* value and the last $\log_2 W$-bit value of d are the same, the remaining bits $(d_{\log_2 W-1}, \cdots, d_1, d_0)$ can be identified. Since the most commonly used 32-bit and 64-bit processors support a barrel shifter, i.e. multiple bit shifts are performed within a single clock cycle, it is difficult to identify the last $\log_2 W$-bit of d. Thus, W candidates remain, requiring to recover accurate indices with additional algebraic computations, similar as discussed in [RHHM17]. It is still possible to extract the substantial parts of the secret indices using only a single trace.

Experiment results on an 8-bit processor. The experiment shows that the *low* value can be recovered when a processor provides 1-bit shift operations. The measurement setup for power consumption traces is as described in Section 4.1. Vertical dot lines in Figure 9[1] indicate the endpoint of each bit rotation of one word, and the endpoint is the same regardless of the index value. This is because the total number of 1-bit right and left shift operations is always the same. As a result, the **`bit rotation`** performs in constant-time. Through assembly analysis, we verified that the compiler handles the variable shift using iterative procedures with the number of repetitions as a variable when it provides a 1-bit shift operation. Thus, the 1-bit right shift operation does not occur when the last 3-bit of d is 0, and the 1-bit right shift operation is performed 7 times when the last 3-bit of d is 7 as shown in Figure 9. Hence, even if ephemeral keys are used or randomization countermeasures to DPAs [RHHM17, CEvMS15b] are applied, it is possible to recover the secret bits (d_2, d_1, d_0) of d using a single trace.

Experiment results on a 32-bit processor. Since $L = 151$, the steps 17 to 19 of the Algorithm 4 in Appendix A operate from $j = 0$ to $j = 148$. Therefore, we can identify 149 patterns indexed from 0 to 148 in the **`bit rotation`**, as shown in Figure 10[1]. Unlike the results on an 8-bit processor, it is impossible to distinguish how many single shift operations are performed. This is because our target 32-bit processor STM32F3 provides the barrel shifter. Thus, in this case, W candidates would be left, and we need to solve some linear equations to find accurate indices, similar as discussed in [RHHM17].

Remark. In our targeted platforms, the condition of constant-time was met. Furthermore, in this paper, we do not consider possible problems depending on compile options. We posted scripts for the proposed attacks online [2].

[1]The traces were shifted by 0.2 multiple units on the y-axis for easy comparison.
[2]`https://drive.google.com/file/d/1olIBpTXs-sZ4Beg3g69b-YVkFrRwb4lh/view?usp=sharing`

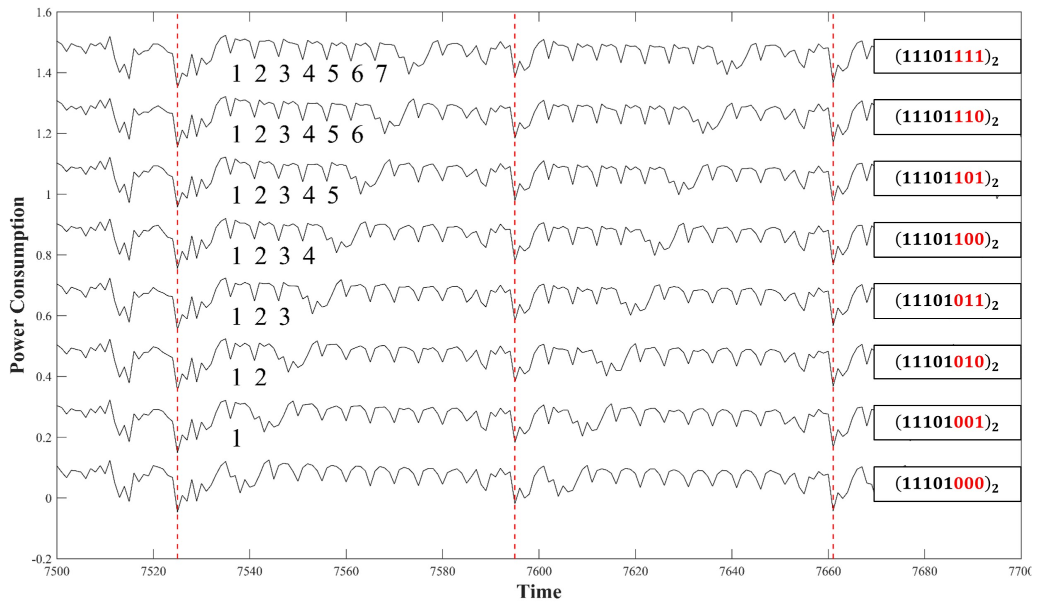


Figure 9: Simple power analysis on the bit rotation ($W = 8$)

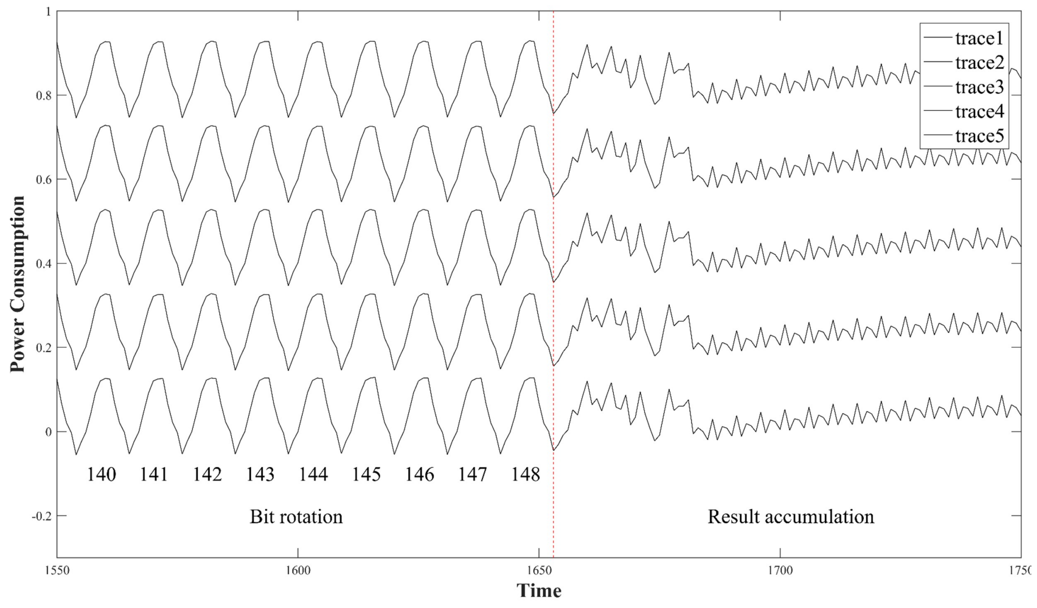


Figure 10: Simple power analysis on the bit rotation ($W = 32$)

6 Case Study: NIST Round 2 Code-Based Cryptography

We here discuss the applicability of our proposed attacks against LEDAcrypt and BIKE, based on QC-LDPC and QC-MDPC codes, respectively, which are the second-round candidates of the NIST PQC standardization [BBC$^+$b, BBC$^+$a, ABB$^+$]. Both authors of LEDAcrypt and BIKE mentioned that their primitives for indistinguishability under chosen plaintext attack (IND-CPA) use ephemeral key pairs to prevent reaction attacks.

Hence, the key generation algorithm for a public/private key pair always runs and the generated key pair is just used one-time. Conversely, regarding indistinguishability under the adaptive chosen ciphertext attack (IND-CCA) versions of LEDAcrypt and BIKE, the authors suggested the use of long-term (static) key pair, meaning that several key exchanges can take place with the same key pair.

The vulnerabilities of QC-MDPC/LDPC code-based cryptography to SCA have been studied, mainly relating to syndrome computations [vMG14b, CEvMS15a, CEvMS16]. However, the BIKE and LEDAcrypt cryptosystems do not consider the secure syndrome computations to SCAs. Thus, we assume that the countermeasures presented in [Cho16, RHHM17, CEvMS15b] are applied to remove each of TA and DPA vulnerability. We briefly describe LEDAcrypt and BIKE cryptosystems and analyze how the proposed attacks can be applied in Subsection 6.1 and Subsection 6.2.

6.1 Case Study: LEDAcrypt

LEDAcrypt consists of LEDAcrypt KEM, designed using the Niederreiter cryptosystem, and LEDAcrypt PKC, designed using the McEliece cryptosystem [BBC+a, BBC+b]. It utilizes QC-LDPC codes to provide improved decoding performance and compact key sizes. The authors of [BBC+a, BBC+b] mentioned that LEDAcrypt KEM using ephemeral and long-term key pairs provides IND-CPA security and IND-CCA2 security. Moreover, LEDAcrypt PKC using long-term key pairs provides IND-CCA2 security. Table 3 describes the key pairs and syndromes of LEDAcrypt to demonstrate the applicability of our proposed attacks.

Using Long-Term Key Pairs. LEDAcrypt KEM and LEDAcrypt PKC using long-term key pairs cannot guarantee resistance against SCAs in private syndrome computations, as shown in [vMG14b, CEvMS15a, CEvMS16]. Therefore, the application of SCA countermeasures might be considered. However, the countermeasures presented in [Cho16, RHHM17, CEvMS15b] are not secure against our multiple- and single-trace attacks. In the case of LEDAcrypt PKC, the secret indices that represent $L = HQ$ can be recovered with our proposed attacks during syndrome computation (see Table 3). Therefore, we can obtain the secret $L = HQ$; besides, it can be used to perform BF decoding to extract a secret message from a received vector over a public code. In the case of LEDAcrypt KEM, L_{n_0-1} can be recovered by our proposed attacks; consequently, it is possible to derive $L = HQ$ using the recovered L_{n_0-1} and the public key P (see Table 3). Hence, $L = HQ$ can be used to perform BF decoding to find secret information, the same as LEDAcrypt PKC.

Table 3: Keys and syndromes of LEDAcrypt

	Public key	Private key	Syndrome
LEDAcrypt KEM	$P = [M \mid I_r] = L_{n_0-1}^{-1} L$	H, Q	$L_{n_0-1} c^{\intercal}$
LEDAcrypt PKC	$P = [Z \mid [M_0 \mid \cdots \mid M_{n_0-2}]^{\intercal}]$		$(HQ) c^{\intercal}$

$* \, I_r$	is an $r \times r$ identity matrix
$* \, Z$	is a diagonal block matrix with $n_0 - 1$ replicas of the block I_r
$* \, M_i$	is an $r \times r$ dense circulant matrix, $0 \le i < n_0 - 1$, $M = [M_0 \mid \cdots \mid M_{n_0-2}]$
$* \, Q$	is an $n \times n$ sparse circulant matrix composed of $n_0 \times n_0$ sparse circulant blocks
$* \, H_i$	is an $r \times r$ sparse circulant matrix, $0 \le i \le n_0 - 1$, $H = [H_0 \mid \cdots \mid H_{n_0-1}]$
$* \, L_i$	is an $r \times r$ sparse circulant matrix, $0 \le i \le n_0 - 1$, $L = HQ$
$* \, c$	is a received row vector, $c = [c_0 \mid \cdots \mid c_{n-1}]$

Using Ephemeral Key Pairs. LEDAcrypt KEM using ephemeral key pairs inherently provides resistance against multiple-trace attacks. Notwithstanding, it would be still vulnerable to TAs in private syndrome computation, as shown in [vMG14b]. Thus, adopting the `constant-time multiplication` as a countermeasure might be considered. However, this countermeasure is still vulnerable to our proposed single-trace attack. Therefore, we can also derive not only secret $L = HQ$ using our single-trace attack but also the secret message.

6.2 Case Study: BIKE

BIKE is a suite of KEM algorithms based on QC-MDPC codes [ABB+]. The authors of [ABB+] present three IND-CPA variants of BIKE, called BIKE-1, BIKE-2, and BIKE-3, which use ephemeral key pairs. BIKE-1, BIKE-2, and BIKE-3 follow the framework of the McEliece cryptosystem, Niederreiter cryptosystem, and Ouroboros, respectively [DGZ17]. They also present three indistinguishability under the chosen ciphertext attack (IND-CCA) variants of BIKE, called BIKE-1-CCA, BIKE-2-CCA, and BIKE-3-CCA, designed to use long-term key pairs. To demonstrate the applicability of our proposed attacks, we describe the key pairs and syndromes of BIKE in Table 4.

Using Long-Term Key Pairs. All the IND-CPA variants of BIKE using long-term key pairs cannot guarantee resistance against SCAs in private syndrome computations, as mentioned in Subsection 6.1. Therefore, similar to LEDAcrypt, our proposed multiple- and single-trace attacks could be applied. In the case of BIKE-1, we can find H with our proposed attacks during syndrome computation, whereas in the case of BIKE-2 and 3, we can find H_0; then it is possible to calculate H_1 using the recovered H_0 and the public key F (see Table 4). The secret message from the received vector can also be extracted using BF decoding and the recovered H.

Using Ephemeral Key Pairs. All the IND-CCA variants of BIKE using ephemeral key pairs also inherently provide resistance against multiple-trace attacks. However, as described in Subsection 6.1, our single-trace attack can be applied. Accordingly, not only H but also the secret message can be retrieved.

Table 4: Keys and syndromes of BIKE

	Public key		Private key	Syndrome
BIKE-1	$F = [F_0 \mid F_1]$	$F_0 = G \cdot H_0$ $F_1 = G \cdot H_1$	H	$Hc^\intercal$
BIKE-2	$F = [F_0 \mid F_1]$	$F_0 = I_r$ $F_1 = H_1 \cdot H_0^{-1}$		$H_0c^\intercal$
BIKE-3	$F = [F_0 \mid F_1]$	$F_0 = G \cdot H_0 + H_1$ $F_1 = G$		$c_0^\intercal + H_0c_1^\intercal$

∗ I_r is an $r \times r$ identity matrix
∗ G is an $r \times r$ dense circulant matrix
∗ H_i is an $r \times r$ sparse circulant matrix, $H = [H_0 \mid H_1]$
∗ c is a received row vector, $c = [c_0 \mid c_1]$

7 Conclusion

We proposed a multiple-trace attack which enables to completely recover accurate secret indices, and also a single-trace attack which can even work when using ephemeral keys or applying existing DPA countermeasures. We also discussed that the BIKE and LEDAcrypt become vulnerable to our proposed attacks.

The proposed multiple-trace attack can be prevented by applying randomization countermeasures, such as intermediate data masking [RHHM17, CEvMS15b], prior to syndrome computations. As for the single-trace attack, the hiding methods, such as random noise and dummy operation, can be applied to increase attack complexity. It would be one of the interesting future research topics to construct theoretically-sound countermeasure against the single-trace attack proposed in this paper.

Acknowledgments

The authors would like to thank the anonymous reviewers and the shepherds Diego F. Aranha and Mehdi Tibouchi for their useful comments that improved the quality of the paper. We also thank Dukjae Moon of Samsung SDS for his valuable feedback and many helpful discussions.

References

[ABB+] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zemor, and Valentin Vasseur. BIKE (Bit Flipping Key Encapsulation). https://bikesuite.org/.

[Age15] National Security Agency. Cryptography Today. https://www.nsa.gov/ia/programs/suiteb_cryptography/, 2015.

[Anz92] Yuichiro Anzai. *Pattern Recognition & Machine Learning*. Elsevier, 1992.

[BBC+a] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDAkem (Low dEnsity coDe-bAsed key encapsulation mechanism). https://www.ledacrypt.org/LEDAkem/.

[BBC+b] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDApkc (Low-dEnsity parity-check coDe-bAsed public-key cryptosystem). https://www.ledacrypt.org/LEDApkc/.

[BCDR17] Dominic Bucerzan, Pierre-Louis Cayrel, Vlad Dragoi, and Tania Richmond. Improved timing attacks against the secret permutation in the mceliece pkc. *International Journal of Computers Communications & Control*, 12(1):7–25, 2017.

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. Mcbits: Fast constant-time code-based cryptography. In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, pages 250–272, 2013.

[BMvT78] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Information Theory*, 24(3):384–386, 1978.

[BS08] Bhaskar Biswas and Nicolas Sendrier. The Hybrid McEliece Encryption Scheme (HyMES). https://www.rocq.inria.fr/secret/CBCrypto/index.php?pg=hymes, 2008. source code published.

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[CD10] Pierre-Louis Cayrel and Pierre Dusart. Mceliece/niederreiter pkc: Sensitivity to fault injection. In *2010 5th International Conference on Future Information Technology*, pages 1–6. IEEE, 2010.

[CEvMS15a] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Differential power analysis of a mceliece cryptosystem. In *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, pages 538–556, 2015.

[CEvMS15b] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Masking large keys in hardware: A masked implementation of mceliece. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 293–309, 2015.

[CEvMS16] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Horizontal and vertical side channel analysis of a mceliece cryptosystem. *IEEE Trans. Information Forensics and Security*, 11(6):1093–1105, 2016.

[Cho16] Tung Chou. Qcbits: Constant-time small-key code-based cryptography. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 280–300, 2016.

[Cho17] Tung Chou. Mcbits revisited. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 213–231, 2017.

[CS16] Julia Chaulet and Nicolas Sendrier. Worst case QC-MDPC decoder for mceliece cryptosystem. *CoRR*, abs/1608.06080, 2016.

[DGZ17] Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, pages 18–34, 2017.

[FGH16] Tomáš Fabšič, Ondrej Gallo, and Viliam Hromada. Simple power analysis attack on the QC-LDPC McEliece cryptosystem. *Tatra Mountains Mathematical Publications*, 67(1):85–92, sep 2016.

[FGO+11] Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high rate mceliece cryptosystems. In *2011 IEEE Information Theory Workshop, ITW 2011, Paraty, Brazil, October 16-20, 2011*, pages 282–286, 2011.

[GLP06] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, pages 15–29, 2006.

[HMP10] Stefan Heyse, Amir Moradi, and Christof Paar. Practical power analysis attacks on software implementations of mceliece. In *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*, pages 108–125, 2010.

[Inca] NewAE Techonology Inc. ChipWhisperer-Lite. https://wiki.newae.com/CW1173_ChipWhisperer-Lite.

[Incb] NewAE Techonology Inc. ChipWhisperer UFO. https://wiki.newae.com/CW308T-STM32F.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996.

[LDW94] Yuan Xing Li, Robert H Deng, and Xin Mei Wang. On the equivalence of mceliece's and niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–273, 1994.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic coding theory. *Coding Thv*, 4244:114–116, 1978.

[Mil85] Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, pages 417–426, 1985.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards.* Springer, 2007.

[MSSS11] H. Gregor Molter, Marc Stöttinger, Abdulhadi Shoufan, and Falko Strenzke. A simple power analysis attack on a mceliece cryptoprocessor. *J. Cryptographic Engineering*, 1(1):29–36, 2011.

[MTSB12] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. Mdpc-mceliece: New mceliece variants from moderate density parity-check codes. *IACR Cryptology ePrint Archive*, 2012:409, 2012.

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. Mdpc-mceliece: New mceliece variants from moderate density parity-check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 2069–2073, 2013.

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Prob. Control and Inf. Theory*, 15(2):159–166, 1986.

[NIS97] NIST. AES Competition. http://csrc.nist.gov/archive/aes/, 1997.

[NIS07] NIST. SHA-3 Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/, 2007.

[NIS16] NIST. Post-Quantum Cryptography. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography, 2016.

[NIS19] NIST. Post-Quantum Cryptography, Round 2 Submissions, NIST Computer Security Resource Center. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions, 2019.

[Pat75] Nicholas J. Patterson. The algebraic decoding of goppa codes. *IEEE Trans. Information Theory*, 21(2):203–207, 1975.

[PRD$^+$15] Martin Petrvalsky, Tania Richmond, Milos Drutarovsky, Pierre-Louis Cayrel, and Viktor Fischer. Countermeasure against the spa attack on an embedded mceliece cryptosystem. In *2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 462–466. IEEE, 2015.

[PRD$^+$16] Martin Petrvalsky, Tania Richmond, Milos Drutarovsky, Pierre-Louis Cayrel, and Viktor Fischer. Differential power analysis attack on the secure bit permutation in the mceliece cryptosystem. In *2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 132–137. IEEE, 2016.

[RHHM17] Melissa Rossi, Mike Hamburg, Michael Hutter, and Mark E. Marson. A side-channel assisted cryptanalytic attack against qcbits. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 3–23, 2017.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134, 1994.

[SS92] V. M. Sidelnikov and S. O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2(4):439–444, 1992.

[SSMS09] Abdulhadi Shoufan, Falko Strenzke, H. Gregor Molter, and Marc Stöttinger. A timing attack against patterson algorithm in the mceliece PKC. In *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, pages 161–175, 2009.

[STM$^+$08] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side channels in the mceliece PKC. In *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*, pages 216–229, 2008.

[Str10] Falko Strenzke. A timing attack against the secret permutation in the mceliece PKC. In *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*, pages 95–107, 2010.

[Str11] Falko Strenzke. Message-aimed side channel and fault attacks against public key cryptosystems with homomorphic properties. *J. Cryptographic Engineering*, 1(4):283–292, 2011.

[Str13] Falko Strenzke. Timing attacks against the syndrome inversion in code-based cryptosystems. In *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, pages 217–230, 2013.

[vMG14a] Ingo von Maurich and Tim Güneysu. Lightweight code-based cryptography: QC-MDPC mceliece encryption on reconfigurable devices. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6, 2014.

[vMG14b] Ingo von Maurich and Tim Güneysu. Towards side-channel resistant implementations of QC-MDPC mceliece encryption on constrained devices. In *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, pages 266–282, 2014.

A Constant-Time Multiplication

The Algorithm 4 is a detailed algorithm scheme for the Algorithm 1.

Algorithm 4 Constant-Time Multiplication in $\mathbb{F}_2[x]/\langle x^r - 1\rangle$ (refer to [Cho16])

```
  Input : $d = (d_{l-1}, \cdots, d_0)_2$, $0 \le d \le r-1$, $c_{(k)} = (c_{L-1}, \cdots, c_0)_{2^W}$, $L = \lceil r/W \rceil$
 Output : $x^d c_{(k)}$
 1: $v \leftarrow 0$, $w \leftarrow c_{(k)}$, $tail \leftarrow r \bmod W$
 2: for $i = l-1$ down to $\log_2 W$ do   ▶ word unit rotation is from 2 to 16
 3:     $d_i \leftarrow (d \gg (l-1-i))\ \&\ 1$
 4:     $mask \leftarrow 0 - d_i$
 5:     $us \leftarrow 1 \ll (i - \log_2 W)$
 6:     $ptr \leftarrow v$, $v \leftarrow w$, $w \leftarrow ptr$
 7:     for $j = 0$ up to $L-1-us-1$ do
 8:         $w[j] \leftarrow (v[j+us]\ \&\ mask) \oplus (v[j]\ \&\ \neg mask)$
 9:     end for
10:     $w[L-1-us] \leftarrow ((v[L-1] \mid (v[0] \ll tail))\ \&\ mask) \oplus (v[L-1-us]\ \&\ \neg mask)$
11:     for $j = 1$ up to $us - 1$ do
12:         $w[j+L-1-us] \leftarrow (((v[j] \ll tail) \mid (v[j-1] \gg (W - tail)))\ \&\ mask)$
13:                                             $\oplus (v[j+L-1-us]\ \&\ \neg mask)$
14:     end for
15:     $w[L-1] \leftarrow ((v[us-1] \gg (W - tail))\ \&\ mask) \oplus (v[L-1]\ \&\ \neg mask)$
16: end for
17: $low \leftarrow d\ \&\ ((1 \ll \log_2 W) - 1)$   ▶ bit rotation is from 17 to 30
18: $mask \leftarrow ((low - 1) \gg (W-1)) - 1$
19: $high \leftarrow W - low$
20: $tmp \leftarrow w[0]$
21: for $j = 0$ up to $L-3$ do
22:     $w[j] \leftarrow w[j] \gg low$
23:     $w[j] \leftarrow w[j] \mid ((w[j+1] \ll high)\ \&\ mask)$
24: end for
25: $w[L-2] \leftarrow w[L-2] \gg low$
26: $w[L-1] \leftarrow w[L-1] \mid (tmp \ll tail)$
27: $w[L-2] \leftarrow w[L-2] \mid ((w[L-1] \ll high)\ \&\ mask)$
28: $w[L-1] \leftarrow w[L-1] \gg low$
29: $w[L-1] \leftarrow w[L-1] \mid ((tmp \ll high)\ \&\ mask)$
30: $w[L-1] \leftarrow w[L-1]\ \&\ ((1 \ll tail) - 1)$
31: Return $w$
```

Toy example for the case $r = 40, W = 8$, a vector $c_{(k)} = (c_0, c_1, \cdots, c_{39}) \in \mathbb{F}_2^{40}$ can be represented as the polynomial $c_{(k)} = c_0 + c_1 x + c_2 x^2 + \cdots c_{39} x^{39} \in \mathbb{F}_2[x]/\langle x^{40} - 1\rangle$. Let the polynomial $c_{(k)}$ be

$$(x^{39}+x^{38}+x^{37}+x^{36})+(x^{27}+x^{26}+x^{25}+x^{24})+(x^{21}+x^{20}+x^{17}+x^{16})+(x^{14}+x^{12}+x^{10}+x^{8}),$$

which can be expressed as a 5-byte array as below:

$v[0]$	$v[1]$	$v[2]$	$v[3]$	$v[4]$
$(00001111)_2$	$(11110000)_2$	$(11001100)_2$	$(10101010)_2$	$(00000000)_2$

Let $d = 19$; then it is represented by 6-bit $(010011)_2$ because $l = \lceil \log_2(40-1) \rceil = 6$. Since $W = 8$ and $\log_2 W = 3$, it is possible to calculate the rotated intermediate values using 8-bit word unit rotation for d_i from d_5 to d_3. For the last 3-bit, $(d_2, d_1, d_0)_2 = (011)_2$, a sequence of logical instructions is used, combining the most significant 5-bit of $v[i]$ and the least significant 3-bit of $v[(i+1) \bmod l]$. Accordingly, the multiplication $x^d = x^{(010011)_2} = x^{0 \cdot 2^5} \cdot x^{1 \cdot 2^4} \cdot x^{0 \cdot 2^3} \cdot x^{(011)_2}$ and $c_{(k)}$ is given by:

$$c_{(k)} \cdot (x^{0 \cdot 2^5} \cdot x^{1 \cdot 2^4} \cdot x^{0 \cdot 2^3} \cdot x^{(011)_2}) = ((((c_{(k)} \cdot x^{0 \cdot 2^5}) \cdot x^{1 \cdot 2^4}) \cdot x^{0 \cdot 2^3}) \cdot x^{(011)_2}).$$

Firstly, the computation is started from the multiplication with x^{2^5} which can be acquired by 4-byte left rotations. However, the d_5 is 0, so the unrotated value is saved.

	$v[0]$	$v[1]$	$v[2]$	$v[3]$	$v[4]$
unrotated	$(00001111)_2$	$(11110000)_2$	$(11001100)_2$	$(10101010)_2$	$(00000000)_2$
rotated	$(00000000)_2$	$(00001111)_2$	$(11110000)_2$	$(11001100)_2$	$(10101010)_2$

Secondly, the multiplication with x^{2^4} can be acquired by 2-byte left rotations. Since the d_4 is 1, the rotated value is saved.

	$v[0]$	$v[1]$	$v[2]$	$v[3]$	$v[4]$
unrotated	$(00001111)_2$	$(11110000)_2$	$(11001100)_2$	$(10101010)_2$	$(00000000)_2$
rotated	$(11001100)_2$	$(10101010)_2$	$(00000000)_2$	$(00001111)_2$	$(11110000)_2$

Thirdly, the multiplication with x^{2^3} can be obtained by 1-byte left rotations. However, the d_3 is 0, so the unrotated value is saved.

	$v[0]$	$v[1]$	$v[2]$	$v[3]$	$v[4]$
unrotated	$(11001100)_2$	$(10101010)_2$	$(00000000)_2$	$(00001111)_2$	$(11110000)_2$
rotated	$(10101010)_2$	$(00000000)_2$	$(00001111)_2$	$(11110000)_2$	$(11001100)_2$

Lastly, the multiplication with $x^{(011)_2}$ can be acquired by the sequence of logical instructions which combines the most significant 5-bit of $v[i]$ and the least significant 3-bit of $v[(i+1) \bmod l]$.

	$v[0]$	$v[1]$	$v[2]$	$v[3]$	$v[4]$
rotated	$(01011001)_2$	$(00010101)_2$	$(11100000)_2$	$(00000001)_2$	$(10011110)_2$

B Experiment Results on a 32-bit Processor

B.1 Multiple-Trace Attack on the Word Unit Rotation

We measured 500 power consumption traces at 7.38 MS/s sampling rate when the Algorithm 4 in Appendix A is operating on a ChipWhisperer UFO STM32F3 target board. Figure 11 shows one of the power consumption traces.

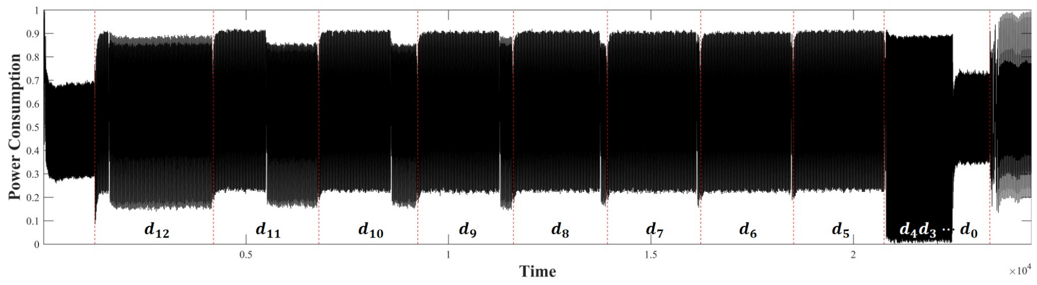
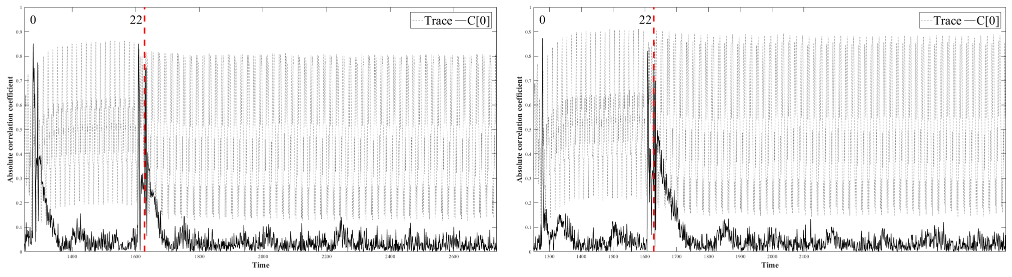


Figure 11: Power consumption trace of the constant-time multiplication ($W = 32$)

Figure 12 shows the experimental proof of the attack methodology to find the most significant bit d_{l-1} based on the New Property 1. Since $L = 151$, $l = 13$, and $\log_2 W = 5$, the steps 7 to 9 of the Algorithm 4 operate 22 times when d_{12}. Thus, the first 22 patterns in Figure 12(a) and Figure 12(b), marked from 0 to 21, are interesting domains. The power consumption with regard to C_0 occurs sequentially **twice** in the 1st iteration of the steps 7 to 9 of the Algorithm 4 when $d_{12} = 0$, as shown in Figure 12(a). Contrariwise, the power consumption with regard to C_0 occurs **once** in the 1st iteration of the steps 7 to 9 of the Algorithm 4 when $d_{12} = 1$, as shown in Figure 12(b).



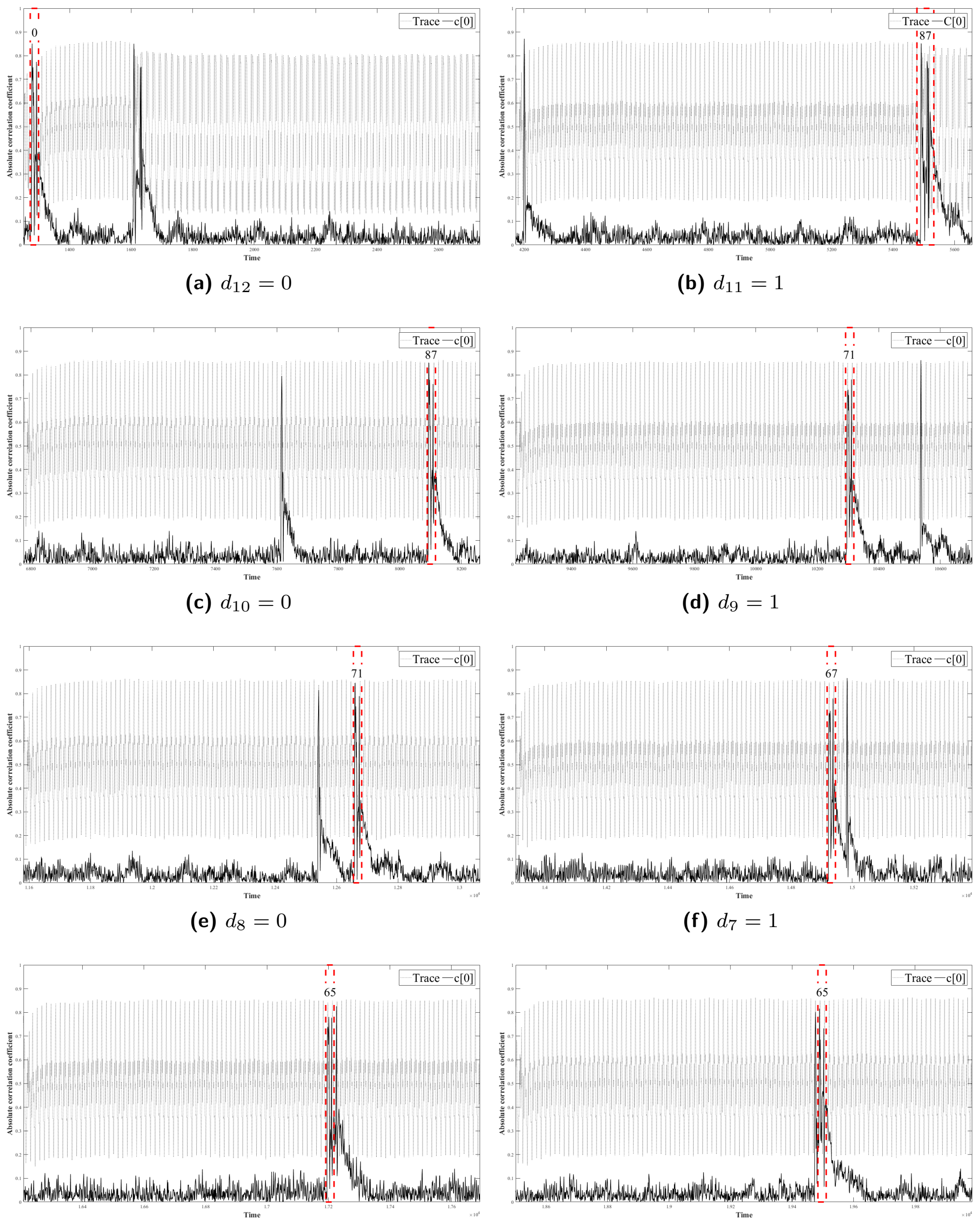
(a) $d_{12} = 0$ **(b)** $d_{12} = 1$

Figure 12: Comparison of correlation coefficient values based on d_{12} (New Property 1)

Figure 13 shows the attack results of $d = (0101011001101)_2$. Each of the figures is a magnification of the computational portion of the corresponding bit of Figure 11. Figure 13(a) shows that the high correlation occurs sequentially **twice** in the steps 7 to 9 of the Algorithm 4 since d_{12} is 0. Figure 13(b) shows that the high correlation occurs sequentially **twice** at a different position[3] with d_{12}, because d_{11} is 1. The same results can be observed in Figure 13(d), Figure 13(f), and Figure 13(g). In contrast, Figure 4(c) shows that the high correlation occurs sequentially **twice** in the same position with d_{11}, because d_{10} is 0. The same results can be observed in Figure 13(e) and Figure 13(h). Further,

[3]Two peaks will occur at $2^{i-\log_2 W}$ words left-rotated position, i.e. $((j-2^{i-\log_2 W}) \bmod L + \bigvee_{k=i+1}^{l-1} d_k)$-th iteration when W cannot divide r.

one can find the accurate secret bits $(d_{12}, d_{11}, d_{10}, d_9, d_8, d_7, d_6, d_5)$ using the Algorithm 2. The attack methodology for finding the remaining bits $(d_4, d_3, d_2, d_1, d_0)$ is described in Subsection 4.2.



(a) $d_{12} = 0$ **(b)** $d_{11} = 1$

(c) $d_{10} = 0$ **(d)** $d_9 = 1$

(e) $d_8 = 0$ **(f)** $d_7 = 1$

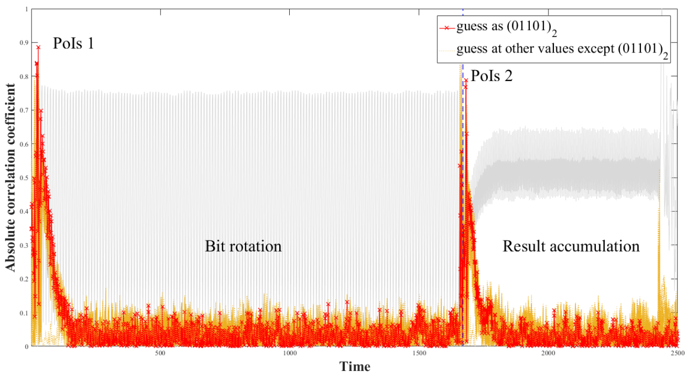
(g) $d_6 = 1$ **(h)** $d_5 = 0$

Figure 13: Finding d from d_{12} to d_5 when $d = (0101011001101)_2$ (New Property 2)

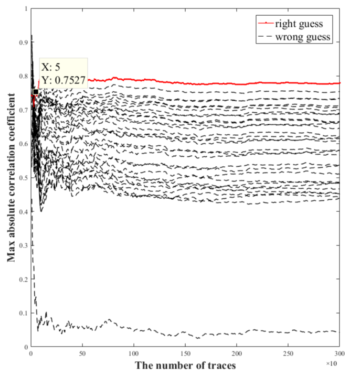
B.2 Multiple-Trace Attack on the Bit Rotation

Here, we demonstrate that the last 5-bit of d can be found by a 32-bit CPA. Using the 32-bit CPA implies that we only use 32-bit of intermediate data. The attack complexity is 2^5 when we target a 32-bit processor since we only need to find the last $\log_2 32 = 5$-bit of

d. Thus, the attack is feasible. Two PoIs can be identified, as shown in Figure 14(a). Even if the CPA uses one of these PoIs, the last 5-bit $(d_4, d_3, d_2, d_1, d_0)$ of d can accurately be derived. Figure 14(b) confirms that 50 traces are sufficient for the attack.



(a) Points of interest



(b) 32-bit CPA results

Figure 14: Correlation power analysis results when $d = (0101011001101)_2$

B.3 Single-Trace Attack on the Word Unit Rotation

The PoIs can be identified by calculating the SOST of the traces and then identifying the location of the information-leaking point, as shown in Figure 15. In the figure, the eight points with the high SOST values are where the $\neg mask$ is loaded in the step 10 of the Algorithm 4. Since $L = 151$, $l = 13$, and $\log_2 W = 5$, the steps 7 to 9 of the Algorithm 4 operate 22 times when $d_i = d_{12}$. Thus, we deduce that the point with the high SOST value is the step 10 of the Algorithm 4, since it appears after the first 22 patterns which are the steps 7 to 9 of the Algorithm 4 as shown in Figure 16.

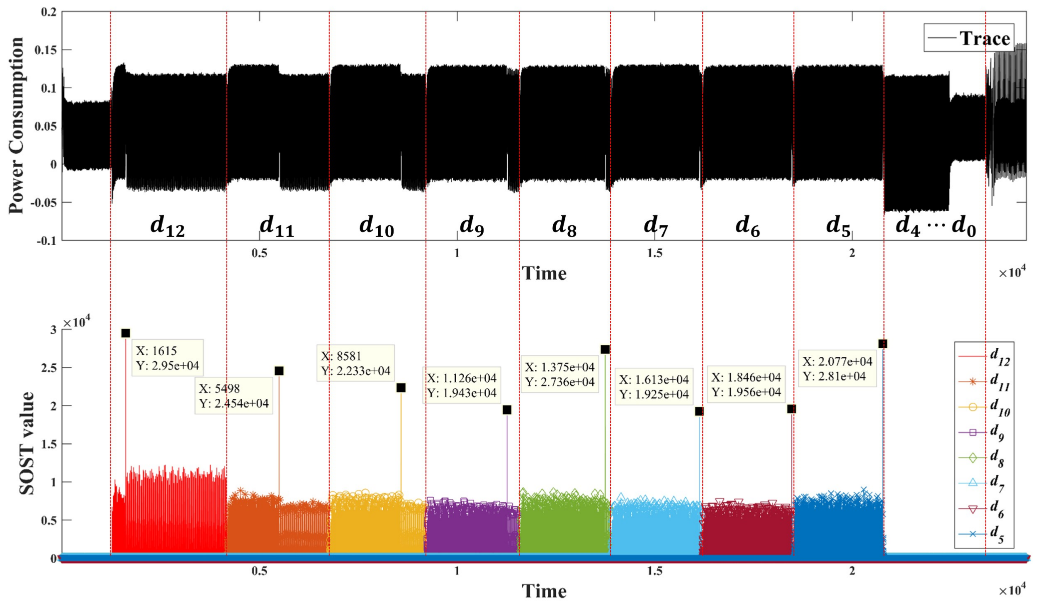


Figure 15: The power consumption trace and the SOST values between two groups G_1 and G_2 of each d_i $(W = 32)$

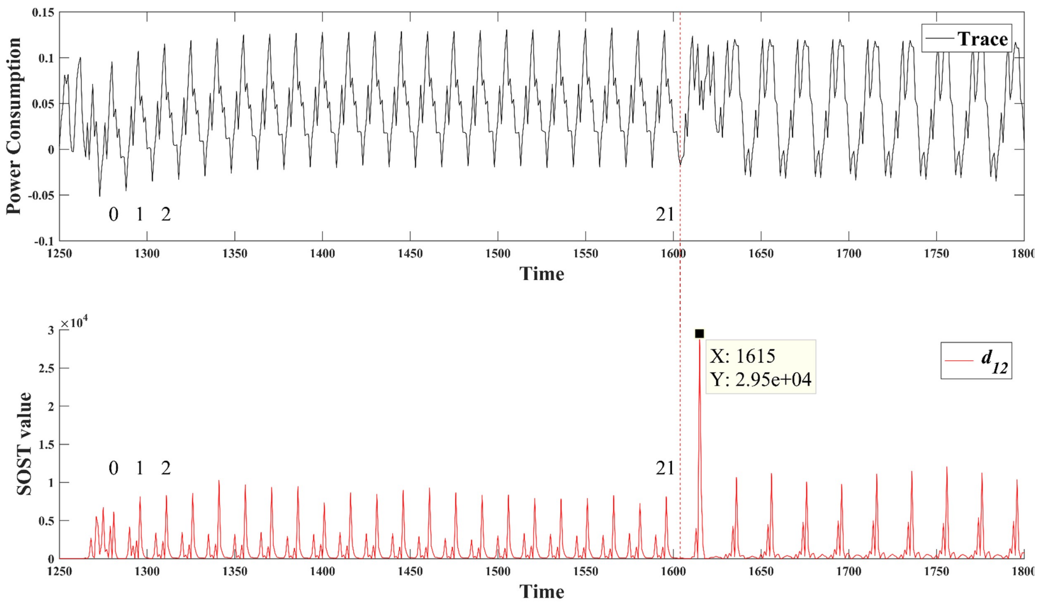


Figure 16: The power consumption trace and the SOST values between two groups, G_1 and G_2, of each d_{12} ($W = 32$)

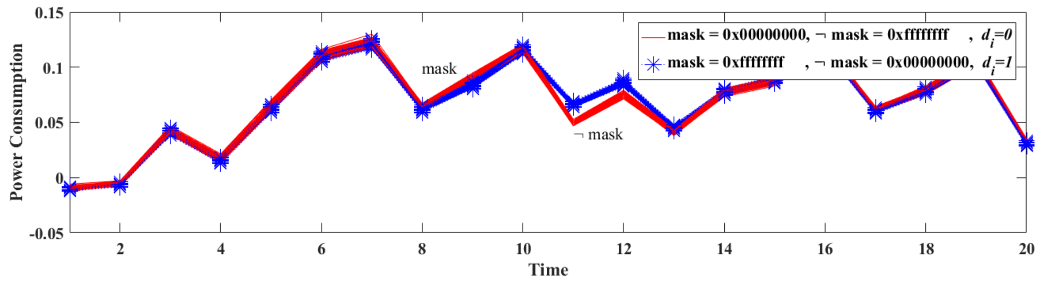


Figure 17: Distribution of PoIs which have the highest SOST value (New Property 4 and New Property 5)

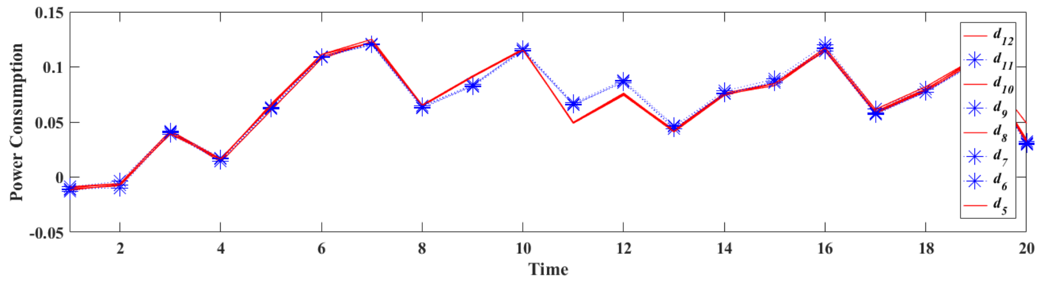


Figure 18: Finding d from d_{12} to d_5 when $d = (0101011000000)_2$ (New Property 4 and New Property 5)

Figure 17 shows the distribution of points which have the highest SOST value, near 1615 points of Figure 15. Two distributions are clearly distinguished: one is when $d_i = 0$, and the other is when $d_i = 1$. We use five points with the high SOST value as PoIs and select these points in the steps 1 to 3 of the Algorithm 3. Since we target the points where the $\neg mask$ is loaded, the average value when $d_i = 0$ is less that when $d_i = 1$ (see Figure 17). Therefore, if the AVG_2 is less than the AVG_1 and d_i belongs to G_2, then d_i is 0. In contrast, if d_i belongs to G_1, it is 1. Figure 8 shows the attack results. Hence, the accurate secret bits $(d_{12}, d_{11}, d_{10}, d_9, d_8, d_7, d_6, d_5)$ of indices can be found.

C Multiple-Trace Attack on the Word Unit Rotation Algorithm Version 2

We can construct an attack methodology such as the Algorithm 5 because the **`word unit rotation`** is not performed while $\sum_i^{l-1} d_i = 0$, where $\log_2 W \leq i \leq l-1$.

Algorithm 5 Multiple-Trace Attack on the Word Unit Rotation (version 2)

```
 Input : a trace set $T = \{T^1, \cdots, T^N\}$ and an input value set $C = \{c^1, c^2, \cdots, c^N\}$
Output : $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$
 1: Calculate the correlation coefficient between $T$ and $C_0 = \{c^1[0], c^2[0], \cdots, c^N[0]\}$
 2: $i \leftarrow l$
 3: do                                              ▶ finding $(d_{l-1}, \cdots, d_i)$
 4:   $i \leftarrow i - 1$
 5:   if the high correlation occurs twice at the 1st iteration then
 6:     $d_i \leftarrow 0$
 7:   else
 8:     $d_i \leftarrow 1$
 9:   end if
10: while $d_i = 0$
11: for $i = i-1$ down to $\log_2 W$ do               ▶ finding $(d_{i-1}, \cdots, d_{\log_2 W})$
12:   if the high correlation occurs twice at the same position (iteration) with $d_{i+1}$ then
13:     $d_i \leftarrow 0$
14:   else
15:     $d_i \leftarrow 1$
16:   end if
17: end for
18: Return $(d_{l-1}, d_{l-2}, \cdots, d_{\log_2 W})$
```