Single-Trace Vulnerability of Countermeasures against Instruction-related Timing Attack

Bo-Yeon Sim[1] and Dong-Guk Han[1,2]

[1] Department of Mathematics, Kookmin University, Seoul, Republic of Korea
{qjdusls,christa}@kookmin.ac.kr
[2] Department of Financial Information Security, Kookmin University, Seoul, Republic of Korea

Abstract. In this paper, we propose that countermeasures against instruction-related timing attack would be vulnerable to single-trace attacks, which are presented at ISPEC 2017 and CHES 2019. The countermeasures use *determiner* to make operations, which leak timing side-channel information, perform in a constant-time. Since *determiner* is divided into two groups according to secret credentials, it is possible to recover secret credentials by clustering *determiner* into two groups.

Keywords: Side-Channel Attack · Timing Attack Countermeasures · Single-Trace Attack · Clustering

1 Introduction

If large-scale quantum computers become a reality, current widely used Rivest-Shamir-Adleman (RSA) [RSA78] and elliptic curve cryptography (ECC) [Mil85, Kob87] become obsolete due to Shor's algorithm [Sho94]. Therefore, as a new alternative, research has been actively conducted on post-quantum cryptography (PQC), which is secure against quantum and classical computers [Age15, CCJ+16, NIS16]. Twenty-six candidates were announced in the second-round of NIST PQC standardization [NIS19], lattice-based and code-based cryptography are promising candidates.

Cryptographic algorithms should be implemented for use and run on a variety of equipment. Therefore, it not only consumes time and power, it also emits electromagnetic waves, while the cryptographic algorithm is running; these kinds of physical information are called as side-channel information. It is possible to extract secret credentials, such as cryptographic keys, using side-channel information, and it was first proposed by Kocher [Koc96]. We called this kind of attacks as side-channel attacks.

LAC [LLZ+18] and Hamming quasi-cyclic (HQC) [MAB+], which are lattice-based and code-based cryptography, respectively, are the second-round candidates of NIST PQC standardization. They use error-correcting code, Bose–Chaudhuri–Hocquenghem (BCH), due to non-zero decryption failure rate. However, there are timing side-channel leakage of BCH, then it is possible to reduce the security of LAC and HQC. Thus, to counter timing attack, constant-time variants of BCH were proposed [WR19, WBBG19].

Our Contributions. In this paper, we show that there are single-trace vulnerabilities of countermeasures against Instruction-related timing attack [WR19, WBBG19]. This vulnerabilities are based on the single-trace attack [SH17, SKH18, SKC+19], which exploits the characteristic that the *mask* value as determined by the secret value is used to obtain accurate results.

2 Instruction-related Timing Attacks on Decoding of BCH codes and Its Countermeasures

Decoding of BCH codes consists of three steps. Firstly, computing the syndromes from the received codeword. Secondly, computing the error-locator polynomial using the syndromes. Thirdly, computing the roots of the error locator polynomial and correcting the received codeword. The simplified inversion-less Berlekamp-Massey algorithm [LJ83], which is mainly used for computing the error-locator polynomial, is vulnerable to timing attacks [WR19, WBBG19]. Vulnerable operations to instruction-related timing attack are categorized into three types as follows.

Loops whose bound is input-dependent For instance, if a loop iterates depending on input length for efficiency reasons or there are early-termination statements such as Listing 1, it would be vulnerable to timing attack.

```
// Case 1
for (i = 0; i < secret_length; i++)
{
  ...
}
// Case 2
for (i = 0; i < max_length; i++)
{
  ...break;
}
// Case 3
for (i = 0; i < max_length; i++)
{
  ...exit;
}
```

Listing 1: Examples of loops whose bound is input-dependent

Thus, to mitigate these vulnerabilities, the iteration length must be fixed and early-termination statements should not be used. Accordingly, repeating the iteration of the loop as maximum length and using a *bound determiner*, such as Listing 2, were proposed. Since *mask* is -1, which all the bits are 1, the *mask* value is `0xffffffff` when the bit length of data types is 32. The $determiner1$ is as shown below:

$$determiner1 = \begin{cases} \texttt{0x00000001} & , \ if\ i < secret\_length; \\ \texttt{0x00000000} & , \ if\ i \geq secret\_length. \end{cases}$$

Therefore, the results of loops determines depending on $determiner1$, and dummy operations are performed when $i \geq secret\_length$.

```
for (i = 0; i < max_length; i++)
{
  // mask is generated based on data types
  determiner1 = ((i - secret_length) & mask) >> 31;
}
```

Listing 2: Examples of constant-execution loops

Branches whose condition is input-dependent As shown in Listing 3, branches execute differently depending on condition, thus, operating pattern is irregular. This irregularity induces the possibility of timing attack.

```
// Case 1
if (i == 1)
{
  a = b;
}
// Case 2
if (j != 0)
{
  v = x;
}
else
{
  v = v;
}
```

Listing 3: Examples of branches whose condition is input-dependent

To make it perform in constant-time, regular operations always carried out independent of secret credentials i and j, as shown in Listing 4. When the bit length of data types is 32, the $determiner2$ and $determiner3$ are as shown below:

$$determiner2 = \begin{cases} \texttt{0x00000001} & , \textit{if } i = 0; \\ \texttt{0x00000000} & , \textit{if } i = 1. \end{cases} \quad determiner3 = \begin{cases} \texttt{0x00000000} & , \textit{if } i = 0; \\ \texttt{0xffffffff} & , \textit{if } i = 1. \end{cases}$$

Therefore, the results determines depending on $determiner2$ and $determiner3$.

```
// mask is generated based on data types
determiner2 = !(((i - 1) & mask) >> 31);
a = b * determiner2;
determiner3 = -((uint32_t)- j >> 31);
v = x & determiner3 + v & !determiner3;
```

Listing 4: Examples of constant-execution branches

Input-dependent memory access Even though arrays are stored in contiguous memory blocks, accessing arrays depends on specific secret data, such as Listing 5, would be vulnerable to timing attack.

```
// Case 1
for (i = 0; i < length; i++)
{
  if(i == index)
  {
    a[i]=b[2i+1];
  }
}
```

Listing 5: Examples of input-dependent memory access

To make uniform array accessing, *blinded array access*, which accesses all elements such as Listing 6, was proposed. If the bit length of data types is 32, the *determiner*4 is as below:

$$determiner4 = \begin{cases} \texttt{0x00000000} & , \; if \; i \neq index; \\ \texttt{0xffffffff} & , \; if \; i = index. \end{cases}$$

Therefore, the results determines depending on *determiner*4.

```
for (i = 0; i < length; i++)
{
  // mask is generated based on data types
  xorVal = i ^ index;
  // anyOnes = 0 if i = index, 1 otherwise
  anyOnes = set_bit(xorVal)
  determiner4 = (anyOnes & 1) - 1;
  out = b[i] & determiner4;
}
```

Listing 6: Examples of uniform array access

To mitigate instruction-relate timing side-channel vulnerabilities, correspondence techniques such as Listing 2, 4, and 6 were applied. As a result, constant-time simplified inversion-less Berlekamp-Massey algorithms are proposed such as Algorithm 1 and 2.

Algorithm 1 Constant-time simplified inversion-less Berlekamp-Massey algorithm [WR19]

```
    Input : $S[2t] = S[1], S[2], \cdots, S[2t]$      ▶ calculated syndromes
    Input : $t$                                   ▶ error-correcting capability of the code
   Output : $C^{t+1}(x)$                          ▶ error-location polynomial
 1: /* Arrays */
 2: $C^{[t+2]}[t+1](x), D[t+1], L[t+1] = \{0\}, UP[t+1]$
 3: /* Initialization */
 4: $C^0[0] = 1, C^1[0] = 1, D[0] = 1, D[1] = S[1], L[0] = 0, L[1] = 0, UP[0] = -1$
 5: $upMax = -1, p = -1, pVal = 0$
 6: /* Main algorithm */
 7: for $i = 1$ up to $t$ do
 8:     $UP[i] = 2 \cdot (i-1) - L[i]$
 9:     $flag1 = !(((0 - D[i]) \;\&\; mask) \gg 31)$      ▶ $flag1 = 1$ if $D[i] = 0$
10:     $C^{i+1}(x) = C^i(x) \cdot flag1$
11:     $L[i+1] = L[i] \cdot flag1$
12:     /* Find another row $p$, prior to ith row such that $D[p]! = 0$ and $UP[p]$ has largest value */
13:     for $j = 0$ up to $i - 1$ do
14:         $flag2 = ((0 - D[i]) \;\&\; mask) \gg 31$      ▶ $flag2 = 1$ if $D[i] \neq 0$
15:         $flag2 = flag2 \cdot ((upMax - UP[j]) \;\&\; mask) \gg 31$
16:                                                ▶ $flag2 = 1$ if $D[i] \neq 0$ and $UP[j] > upMax$
17:         $flag2 = flag2 \cdot !flag1$
18:         $upMax = UP[j] \cdot flag2 + upMax \cdot !flag2$
19:         $p = j \cdot flag2 + p \cdot !flag2$
20:     end for
21:     $flag2 = !(((0 - p) \;\&\; mask) \gg 31)$      ▶ $flag2 = 1$ if $p = 0$
22:     $pVal = 1/2 \cdot flag2 + p \cdot !flag2$
23:     $C^{i+1}(x) = (C^i(x) + (D[i] \cdot (blinded(p, t+1, D))^{-1} \cdot x^{2 \cdot ((i-1)-(pVal-1))} \cdot C^p(x))) \cdot !flag1$
24:     $L[i+1] = (max(L[i], blinded(p, t+1, L) + 2 \cdot (i-1) - (pVal - 1))) \cdot !flag1$
25:     $flag2 = (((i - t) \;\&\; mask) \gg 31)$      ▶ $flag2 = 1$ if $i \neq t$
```

```
26:     $D[i+1] = (blinded(2 \cdot (i \cdot flag2) + 1 - j, t, S)$
27:                 $+ \sum_{j=1}^{L[i+1]} (blinded(j, t+1, C^{i+1}) \cdot blinded(2 \cdot i + 1 - j, t, S))) \cdot flag2$
28: end for
29: Return $C^{t+1}(x)$
```

Algorithm 2 Constant-time simplified inversion-less Berlekamp-Massey algorithm [WBBG19]

```
    Input : $syndromes$                    ▶ calculated syndromes
    Input : $t$                            ▶ error-correcting capability of the code
    Output : $sigma$                       ▶ error-location polynomial
1:  /* Arrays */
2:  $sigma[2 \cdot (t+1)], sigma\_copy[t-1], X\_sigma\_p[t+1]$
3:  /* Initialization */
4:  $sigma[0] = 1, X\_sigma\_p[1] = 1, d = syndromes[0]$
5:  $deg\_sigma = 0, deg\_sigma\_p = 0, deg\_sigma\_copy = 0, pp = -1, d\_p = 1$
6:  /* Main algorithm */
7:  for $i = 0$ up to $t-1$ do
8:      $memcpy(sigma\_copy, sigma, 2 \cdot (t-1))$
9:      $deg\_sigma\_copy = deg\_sigma$
10:     $dd = gf\_mul(d, gf\_inverse(d\_p))$
11:     for $j = 1$ up to $(j \leq 2 \cdot i + 1)$ && $(j \leq t)$ do
12:         $sigma[j] = sigma[j] \oplus gf\_mul(dd, X\_sigma\_p[j])$
13:     end for
14:     $deg\_X = 2 \cdot i - pp$
15:     $deg\_X\_sigma\_p = deg\_X + deg\_sigma\_p$
16:     $mask1 = -(((uint16\_t) - d) \gg 15)$        ▶ $mask1 =$ 0xffff if $d \neq 0$
17:     $mask2 = -(((uint16\_t)(deg\_sigma - deg\_X\_sigma\_p) \gg 15)$
18:                                  ▶ $mask2 =$ 0xffff if $deg\_X\_sigma\_p > deg\_sigma$
19:     $mask12 = mask1$ & $mask2$
20:     $deg\_sigma = (mask12$ & $deg\_X\_sigma\_p) \oplus (!mask12$ & $deg\_sigma)$
21:     if $i == t-1$ then
22:         break;
23:     end if
24:     $pp = (mask12$ & $(2 \cdot i)) \oplus (!mask12$ & $pp)$
25:     $d\_p = (mask12$ & $d) \oplus (!mask12$ & $d\_p)$
26:     for $j = t-1$ down to 0 do
27:         $X\_sigma\_p[j+1] = (mask12$ & $sigma\_copy[j-1])$
28:                                              $\oplus(!mask12$ & $X\_sigma\_p[j-1])$
29:     end for
30:     $X\_sigma\_p[1] = 0$
31:     $X\_sigma\_p[0] = 0$
32:     $deg\_sigma\_p = (mask12$ & $deg\_sigma\_copy) \oplus (!mask12$ & $deg\_sigma\_p)$
33:     $d = syndromes[2 \cdot i + 2]$
34:     for $j = 1$ up to 0 do
35:         $d = d \oplus gf\_mul(sigma[j], syndromes[2 \cdot i + 2 - j])$
36:     end for
37: end for
38: Return $sigma$
```

3 Single-Trace Vulnerabilities of Countermeasures against Instruction-related Timing Attacks

In this section, we demonstrate that there are single-trace vulnerabilities on Algorithm 1 and 2, which are countermeasures against instruction-related timing attack. We consider single-trace attacks proposed in [SH17, SKH18, SKC$^+$19]. Key bit-dependent properties were categorized and used to classify the power consumption traces into two groups according to key bit value. In this paper, we consider software implementations and Hamming weight power consumption model. Our object is clustering the power consumption traces depending on *determiner*, thus, the key bit-dependent properties are written according to *determiner* as follows.

Property 1. Assume that *determiner* is 0 or 1. Thus, if $determiner = 0$, the power consumption is associated with 0 when saving and loading the *determiner* value. In contrast, if $determiner = 1$, then the power consumption is associated with 1.

Property 2. Assume that *determiner* is `0x00000000` or `0xffffffff` on a 32-bit processor; therefore, if *determiner* is `0x00000000`, the power consumption is related to 0. In contrast, when *determiner* is `0xffffffff`, the power consumption is related to 32, which is the Hamming weight of the *determiner* value.

See the example in Section 2, $determiner1$ and $determiner2$ can be divided into two groups based on Property 1. Likely, $determiner3$ and $determiner4$ can be divided into two groups based on Property 2. According to the results of the experiments in [SH17, SKH18, SKC$^+$19], it is possible to classify using the clustering algorithm, such as k-means, fuzzy k-means, or EM algorithms). Single trace is sufficient and profiling is not needed. Therefore, distinguishing the determination conditions of *determiner* is possible, and it makes countermeasures against instruction-related timing attack obsolete.

Constant-time simplified inversion-less Berlekamp-Massey algorithm As mentioned in the paragraph above, there are single-trace vulnerabilities on Algorithm 1 and 2. The countermeasures removed input-dependent loops, branches, and array accesses, however, they use input-dependent *determiner* to acquire correct results. In case of Algorithm 1, it is possible to cluster $flag1$ and $flag2$ into two groups, respectively. Similarly, $mask1$ and $mask2$ can be clustered into two groups, respectively, depending on whether it is `0xffff` or `0x0000` on 16-bit processors. Finding the determination conditions is easy since one of groups indicates leakage of zero, and the other group indicates leakage of non-zero. Consequently, the error-locator polynomial can be known, and correcting the error and recovering the message become easy.

4 Conclusion

In this paper, we discussed that constant-time simplified inversion-less Berlekamp-Massey algorithms [WR19, WBBG19] are vulnerable to single-trace attacks presented in [SH17, SKH18, SKC$^+$19]. This results implies that the use of input-dependent *determiner* can counter against timing attack, but it cannot counter against clustering-based single-trace attacks. Therefore, research on constructing countermeasure is required, and the hiding methods, such as dummy operation and random noise, can be applied to decrease the accuracy of classification.

References

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

[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.

[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.

[LJ83] Shu Lin and Daniel J. Costello Jr. *Error control coding - fundamentals and applications*. Prentice Hall computer applications in electrical engineering series. Prentice Hall, 1983.

[LLZ+18] Xianhui Lu, Yamin Liu, Zhenfei Zhang, Dingding Jia, Haiyang Xue, Jingnan He, and Bao Li. LAC: practical ring-lwe based public-key encryption with byte-level modulus. *IACR Cryptology ePrint Archive*, 2018:1009, 2018.

[MAB+] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. HQC (Hamming Quasi-Cyclic). http://pqc-hqc.org/.

[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.

[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.

[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.

[SH17] Bo-Yeon Sim and Dong-Guk Han. Key bit-dependent attack on protected PKC using a single trace. In *Information Security Practice and Experience - 13th International Conference, ISPEC 2017, Melbourne, VIC, Australia, December 13-15, 2017, Proceedings*, pages 168–185, 2017.

[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.

[SKC+19] Bo-Yeon Sim, Jihoon Kwon, Kyu Young Choi, Jihoon Cho, Aesun Park, and Dong-Guk Han. Novel side-channel attacks on quasi-cyclic code-based cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):180–212, 2019.

[SKH18] Bo-Yeon Sim, Junki Kang, and Dong-Guk Han. Key bit-dependent side-channel attacks on protected binary scalar multiplication †. *Applied Sciences*, 8(11):2168, nov 2018.

[WBBG19] Guillaume Wafo-Tapa, Slim Bettaieb, Loïc Bidoux, and Philippe Gaborit. A practicable timing attack against HQC and its countermeasure. *IACR Cryptology ePrint Archive*, 2019:909, 2019.

[WR19] Matthew Walters and Sujoy Sinha Roy. Constant-time BCH error-correcting code. *IACR Cryptology ePrint Archive*, 2019:155, 2019.