Single Trace Analysis of Comparison Operation based Constant-Time CDT Sampling and Its Countermeasure

Keon Hee Choi[1][0009−0007−2557−566X], Ju-Hwan Kim[1][0000−0001−8762−5553], Jaeseung Han[1][0000−0001−7111−2315], Jae Won Huh[1][0000−0002−6452−8002], and Dong-Guk Han[1,2][0000−0003−1695−5103]

[1] Department of Financial Information Security, Kookmin University, Seoul, Republic of Korea
[2] Department of Information Security, Cryptology, and Mathematics, Kookmin University, Seoul, Republic of Korea
{dy1637,zzzz2605,gjwodnjs987,jae1115,christa}@kookmin.ac.kr

Abstract. Cumulative Distribution Table(CDT) sampling is a Gaussian sampling technique commonly used for extracting secret coefficients or core matrix values in lattice-based Post-Quantum Cryptography (PQC) algorithms like FrodoKEM and FALCON. This paper introduces a novel approach: a single trace analysis(STA) method for comparison operation based constant-time CDT sampling, as employed in SOLMAE—a candidate for Korean Post-Quantum Cryptography(KPQC) first-round digital signature Algorithm. The experiment is measuring power consumption during the execution of SOLMAE's sampling operation on an 8-bit AVR compiler microcontrollers unit(MCU) using ChipWhisperer-Lite. By utilizing STA, this paper recovered output of comparison operation based constant-time CDT sampling.The source of CDT sampling leakage is investigated through an in-depth analysis of the assembly code. The 8-bit AVR MCU conducts comparison operations on values exceeding 8 bits by dividing them into 8-bit blocks. Consequently, the execution time of a CDT sampling operation is influenced by the outcome of each block's comparison operation due to conditional branching. To address these concerns, this paper begins by summarizing trends in CDT sampling related research to design robust countermeasures against single trace analysis. Furthermore, a novel implementation method for comparison operation based constant-time CDT sampling against STA is proposed. This assembly-level implementation removes branching statements and performs comparative operations on all data words. Through experimental validation, this paper demonstrates the safety of the proposed countermeasure algorithm against STA.

Keywords: Side Channel Analysis · Single Trace Analysis · PQC · Gaussian sampling · CDT sampling · KPQC · SOLMAE · AVR

1 Introduction

The usage of public key cryptographic algorithms, such as Public-key Encryption(PKE)/Key Encapsulation Mechanism(KEM) and Digital Signature Algorithm(DSA), is widespread across various fields. However, it has been demonstrated that these algorithms will become vulnerable in the future due to the emergence of quantum computers and Shor's algorithm.[1,2] To address these security concerns, the National Institute of Standards and Technology (NIST) initiated the PQC standardization competition in 2016. The objective of this competition is to develop public-key cryptographic algorithms that can resist attacks from quantum computers. Currently, a subround is in progress following the final round of the competition. Additionally, as part of the competition, new algorithms are being proposed that build upon the shortlisted and selected algorithms. The competition was divided into two main areas for public-key cryptography, namely PKE/KEM and Digital Signature. Importantly, numerous lattice-based algorithms have been proposed in both areas. In these lattice-based cryptographic algorithms, important values are extracted from the Gaussian distribution, and the method employed to extract them using a table is known as CDT sampling. In other words, CDT sampling is a crucial role in lattice-based algorithms.

There are many ways to implement CDT sampling. The first proposed CDT sampling has been analyzed using the technique proposed by [3], resulting in the proposal of constant-time CDT sampling. This constant-time CDT sampling was implemented using subtraction in FrodoKEM and Lizard. Additionally, [4,5] proposed STA for CDT sampling. then secret value of FrodoKEM was leaked. Repeatedly, CDT sampling is very important. In this paper, we study in detail the security of side channel analysis for comparison operation based CDT sampling in MITAKA [6] and SOLMAE [7], which are a similar structure of the Falcon. Importantly, the security of side channel analysis for these comparison operation based CDT sampling techniques has not been studied before this work. This paper recovery the sampling value of CDT sampling through STA for vulnerability that is variable the operating time of CDT sampling depending on the results of comparative operations in 8-bit AVR MCU. To validate this vulnerability, the paper employs ChipWhisperer-Lite to measure power consumption during CDT sampling on the Atmel XMEGA-128, using the AVR compiler for the 8-bit processor. Additionally, using assembly code root cause analysis, the paper proposes a secure constant-time CDT sampling method using comparison operations to counter STA.

1.1 Contribution

This paper addresses the safety of comparison operation based constant-time CDT sampling from a side-channel analysis perspective, which has not been previously studied. In addition, by analyzing the power consumption traces used in SOLMAE, we identified the *basesampler* in the overall cryptographic operation

algorithm. This increases the feasibility of the STA in this paper. So, this paper describes the reason for vulnerability in comparison operation-based CDT sampling in great detail. Experiments have confirmed that CDT sampling in 8-bitAVR MCU varies in operating time depending on comparison operation results. The cause analysis was performed using an assembly code. In the 8-bit AVR MCU, during CDT sampling operations, when comparing values larger than 8 bits, the process is divided into 8-bit units. The analysis reveals that the operation concludes the moment the result is determined, resulting in a change in execution time. In essence, not all blocks undergo comparison operations, and this behavior is closely associated with the presence of branch statements.

A novel STA is propose for comparison operation based CDT sampling. Additionally, a new CDT sampling implementation method is propose to resist side-channel analysis, contributing to the development of secure algorithms for CDT sampling. The practical implementation removes the branch statements from the assembly code and presents a structure where all blocks can be compared. Experimental verification demonstrates the resistance to STA through power consumption trace analysis.

1.2 Organization

The remainder of this paper introduces STA for CDT sampling through a total of five sections. In Section 2, it provides detailed explanation of lattice, LWE, and NTRU, emphasizing the significance of Gaussian sampling. This highlights the importance of CDT sampling, the two implementation methods, and the imperative need to investigate the security of comparison based CDT sampling, which has not been previously explored. Moving on to Section 3, it presents the experimental setup and target implementation. Section 4 delves into a side channel analysis of CDT sampling based on comparison operations. Here, it detailed describe the application method of STA and the cause of the vulnerability. In Section 5, we present the implementation of CDT sampling in which vulnerabilities are mitigated through an analysis of the underlying causes. To demonstrate their resistance against the attack technique proposed in this paper, it collect actual a power consumption trace. Finally, Section 6 addresses conclusions and future research directions.

2 Backgrounds

In this section, we provide an introductory overview of lattice-based cryptography[8], LWE, and NTRU encryption schemes. Following that, we delve into the Gaussian distribution and proceed to describe CDT sampling, which is of paramount importance as a module. We then elaborate on timing side-channel analysis conducted on the original CDT sampling, followed by an in-depth description of a STA of the subtraction operation based constant-time CDT sampling.

2.1 Lattice

Definition 1. Lattice: Given n linearly independent vectors $\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_n} \in \mathbb{R}^m$, the lattice $\mathcal{L}(\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_n})$ is defined as the set of all linear combinations of $\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_n}$ with integer coefficients, i.e.,

$$\mathcal{L}(\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_n}) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b_i} \mid x_i \in \mathbb{Z} \right\}.$$

We refer to $\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_n}$ as a basis of the lattice.

Equivalently, if we define $\mathbf{B}$ as the $m \times n$ matrix whose columns are $\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_n}$, then the lattice generated by $\mathbf{B}$ is

$$\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^n\}.$$

2.2 NTRU and LWE

The first public key cipher based on Lattice was proposed by A.M. in 1997, and since then, various studies have been conducted to create efficient encryption algorithms [9]. In Lattice-based encryption, efficiency primarily refers to speed and key size. NTRU, proposed by Hoffstein et al. in 1996 [10], is known for its fast encryption process. Falcon, MITAKA, and SOLMAE are examples of NTRU-based encryption algorithms [6,11,12].

Definition 2. NTRU: Let q be a positive integer, and $z(x) \in \mathbb{Z}$ be a monic polynomial. Then, a set of NTRU secrets consists of four polynomials $f, g, F, G \in R_q$ which satisfy the NTRU equation:

$$fG - gF \equiv q \mod z(x).$$

And define h as $h \leftarrow g \cdot f^{-1} \mod q$. Then, given h, find f and g.

LWE was proposed by Regev in 2005 [13]. LWE is known to be NP-hard, even when adding small values of noise. CRYSTAL-KYBER and CRYSTAL-Dilithium are examples of LWE-based cryptographic algorithms [12,14].

Definition 3. LWE: Let n and q be positive integers, and let χ be a distribution over $\mathbb{Z}$. For a vector $s \in \mathbb{Z}_q^n$, the LWE distribution $\mathcal{A}_{s,\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $a \in \mathbb{Z}_q^n$ and an integer error e from χ. The distribution returns the pair $(a, \langle a, s \rangle + e \mod q)$.

There are two important concepts of LWE.

- **Search-LWE problem:** Given m independent samples $(a_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ drawn from $\mathcal{A}_{s,\chi}$, find s.
- **Decision-LWE problem:** Given m independent samples $(a_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, distinguish whether each sample is drawn from the uniform distribution or from $\mathcal{A}_{s,\chi}$.

2.3 Discrete Gaussian Distribution

In this paper, CDT sampling is the method to extract random values from Gaussian distribution. Prior to CDT sampling, the definition of the discrete Gaussian distribution on the lattice is given as follows.

Definition 4. Discrete Gaussian Distribution over Lattice: Let $\forall c \in \mathbb{R}^n$, $\sigma \in \mathbb{R}^+$,

$$\forall x \in \mathbb{R}^n, \rho_{\sigma,c}(x) = exp(\frac{-\pi \| x - c \|^2}{\sigma^2}).$$

Then, for $\forall c \in \mathbb{R}^n$, $\sigma \in \mathbb{R}^+$, n-dimensional lattice $\mathcal{L}$, define the Discrete Gaussian Distribution over $\mathcal{L}$ as:

$$\forall x \in \mathcal{L}, \mathcal{D}_{L,\sigma,c}(x) = \frac{\rho_{\sigma,c}(x)}{\rho_{\sigma,c}(\mathcal{L})}.$$

2.4 CDT Sampling

Some lattice-based schemes based on LWE extract the error from a Gaussian distribution. Similarly, certain lattice-based schemes based on NTRU create essential values from a Gaussian distribution. CDT sampling is an efficient method for extracting values from these Gaussian distributions, and ensuring the security of such CDT sampling is of utmost importance. The CDT table stores specific probability values of the Gaussian distribution. CDT sampling is an algorithm that randomly generates probability values and determines the range within which the generated values fall among those stored in the table. The value to be sampled at this point corresponds to the determined index. There are several ways to implement CDT sampling, and this paper deals with the safety study of implementing CDT sampling based on comparison operations.

Algorithm 1 The CDT sampling vulnerable to timing attack

Input : CDT table Ψ, σ, τ
Output : Sampled value S
1: $rnd \leftarrow [0, \tau\sigma) \cap \mathbb{Z}$ uniformly at random
2: sign $\leftarrow [0, 1] \cap \mathbb{Z}$ uniformly at random
3: $i \leftarrow 0$
4: **while** $(rnd > \Psi[i])$ **do**
5: $\quad i++$
6: **end while**
7: $S \leftarrow ((-\text{sign}) \wedge i) + \text{sign}$
8: **return** S

The initially proposed CDT sampling Algorithm 1 was found to be vulnerable to the timing attack proposed by [3]. This vulnerability arises due to the

different timing of the *while* loop termination. As a remedy, constant-time CDT sampling utilizes for statements. There are two ways to implement this: CDT sampling based on comparison operations and CDT sampling based on subtraction operations.

Algorithm 2 The subtraction operation based CDT sampling

Input : CDT table Ψ of length ℓ, σ, τ
Output : Sampled value S
1: $rnd \leftarrow [0, \tau\sigma)$ uniformly at random
2: sign $\leftarrow [0, 1] \cap \mathbb{Z}$ uniformly at random
3: $S \leftarrow 0$
4: **for** $i = 0$ to $\ell - 1$ **do**
5: $\quad S \mathrel{+}= (\Psi[i] - rnd) \gg 63$
6: **end for**
7: $S \leftarrow ((-\text{sign}) \wedge S) + \text{sign}$
8: **return** S

Both methods are available for schemes that use CDT sampling. However, only subtraction based CDT sampling has been suggested to be vulnerable. Algorithm 2 is an example of subtraction operation based CDT sampling. LWE-based lattice-based schemes commonly employ this algorithm [15,16]. Additionally, it has been proposed to perform STA by the power differences between negative and positive numbers [4,5]. Moreover, an attack to find the secret key of a cryptographic algorithm has been proposed using this method.

Algorithm 3 The comparison operation based CDT sampling: half-Gaussian table access CDT

Input : CDT table Ψ of length ℓ, σ, τ
Output : Sampled value S
1: $rnd \leftarrow [0, \tau\sigma)$ uniformly at random
2: sign $\leftarrow [0, 1] \cap \mathbb{Z}$ uniformly at random
3: $S \leftarrow 0$
4: **for** $i = 0$ to $\ell - 1$ **do**
5: $\quad S \mathrel{+}= (rnd \geq \Psi[i])$
6: **end for**
7: $S \leftarrow ((-\text{sign}) \wedge S) + \text{sign}$
8: **return** S

On the other hand, NTRU-based lattice-based schemes often utilize CDT sampling based on comparison operations, especially in [6,7,17] which employs hybrid sampling. Algorithm 3 is the comparison based CDT sampling. Unlike

conventional methods, it performs sampling from Gaussian distribution using comparison operations.

3 Experiment Setup

In this section, the experimental environment and the CDT sampling code employed in the STA experiments are described. The C code of SOLMAE, a candidate from the KPQC Round 1 digital signature category, was implemented in the AtmelXMEGA128 environment. Power consumption traces were collected during the operation for analysis.

3.1 Implimentation of comparison operation based CDT sampling

The *BaseSampler* function implemented in SOLMAE and MITAKA employs a comparison operation based CDT sampling approach. Thus, this paper utilizes the reference code of SOLMAE, which was proposed as a candidate for KPQC Round 1 digital signature. Specifically, our focus is on the *BaseSampler* function within the code. The sampling technique in SOLMAE follows the sampling outlined in [17] and employs a table to generate values from a half-Gaussian distribution. The *BaseSampler* function is illustrated in Listing 1.1. The CDT table contains 13 values arranged in ascending order, which are sequentially compared against the randomly selected value "r" from the reference code.

```
int base_sampler()
{
  uint64_t r = get64(); //get randomly 64 bits from RNG.
  int res = 0;
  for (int i = 0; i < TABLE_SIZE; i++)
    res += (r >= CDT[i]);
  return res;
}
```

Listing 1.1: *BaseSampler* function C code

3.2 Target Device of Experiment

The board utilized in this paper consists of an AtmelXMEGA128 (8-bit processor) and Chipwhisperer-Lite. The AtmelXMEGA128 is an 8-bit AVR MCU. The *BaseSampler* function implemented in *SOLMAE* operates on the AtmelXMEGA128 board, while Chipwhisperer-Lite is employed to collect the power consumption data during the *BaseSampler* function operation Figure 1.

The experimental steps conducted in this paper are as follows:

- Collection of power consumption data during the comparison operation-based CDT sampling.
- Analysis of the assembly language, considering different compiler optimization levels, to identify vulnerabilities in the comparison operations.
- Investigation of comparison operation vulnerabilities using real-world traces.
- Acquisition of output values for the newly proposed CDT sampling algorithm through STA.

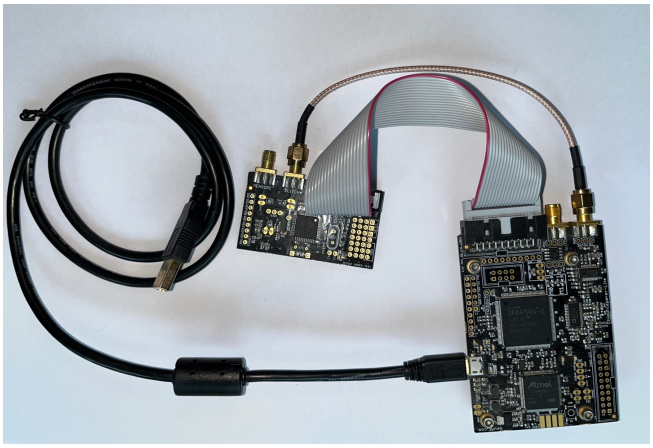

Fig. 1: AtmelXMEGA128(8-bit AVR MCU) and Chipwhisperer-Lite

This paper demonstrates that vulnerable implementations of comparison operations, which could be realistic in a commercialized environment, can expose the actual values of CDT sampling. Furthermore, a CDT sampling algorithm resistant to side-channel attacks is proposed.

4 Side Channel Analysis of Comparison Operation based Constant-Time CDT Sampling

4.1 Description of the Cause of Vulnerability

The security of comparison operations heavily depends on the specific implementation technique and enviroment like compiler. Let us consider the comparison of two multi-word numbers, denoted as A and B in Figure 2.

Various methods can be employed to compare these numbers. One common approach is to initiate the comparison with the most significant words. Compare A and B as follows:

1. Check if A_0 is greater than B_0. If so, $A > B$.
2. Check if A_0 is less than B_0. If true, $A < B$.

3. Check if A_0 and B_0 are equal. If true, continue to compare the next word until the comparison ends.

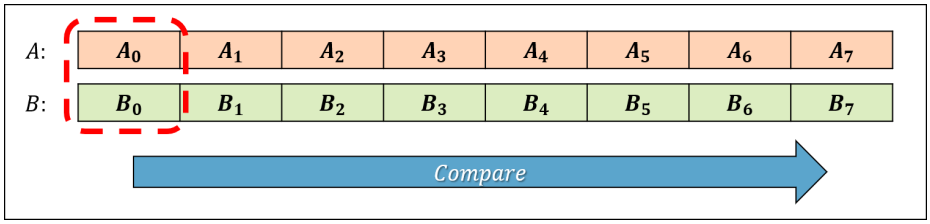

Fig. 2: The comparison of two multi-word numbers, denoted as A and B. A and B are each 8 blocks.

This implementation is vulnerable to side channel analysis. For instance, let's consider two scenarios: (1) $A_0 > B_0$ and (2) $A_0 = B_0, A_1 < B_1$. In these situations, the execution time of the comparison operations may differ. As a result, timing vulnerabilities arise, which can be exploited through STA to distinguish between the two scenarios. Therefore, a comparison algorithm resistant to STA is required.

```
<base_sampler>:
    ...
    24c:                ldi r22, 0x00
    24e:                ldi r23, 0x00
    250:                ldd r24, Z+7
    252:                cp  r25, r24
    254:                brcs  .+74        ; 0x2a0 <base_sampler+0x92>
    256:                cp  r24, r25
    258:                brne  .+66        ; 0x29c <base_sampler+0x8e>
    25a:                ldd r24, Z+6
    25c:                cp  r20, r24
    25e:                brcs  .+64        ; 0x2a0 <base_sampler+0x92>
    260:                cp  r24, r20
    262:                brne  .+56        ; 0x29c <base_sampler+0x8e>
    ...
    29c:                ldi r22, 0x01     ; 1
    29e:                ldi r23, 0x00     ; 0
    2a0:                add r18, r22
    2a2:                adc r19, r23
    ...
```

Listing 1.2: Base Sampler() assembly code

In this section, the vulnerabilities associated with various implementations of weak comparison operations are explored. The assembly code of the *BaseSampler*

function used in *SOLMAE* is examined for various optimization levels (Level: 0, 1, 2, 3, s) provided by the AtmelXMEGA128. The assembly code depicted in Listing 1.2 illustrates the part of *BaseSampler* function for the optimized s-level. It is evident that the comparisons are performed sequentially, word by word. Notably, vulnerabilities in the word based comparison method are evident. The process of performing comparison operations for each optimization level follows a similar pattern as shown in Listing 1.2. Subsequent instructions are dependent on the results of the word comparisons, leading to variations in executed operations and resulting in distinct power consumption patterns manifested as differences in power traces.

In more detail, the first word is compared in lines 252, 254, and the next operation varies depending on the result. First, calculate $r25 - r24$. If a carry occurred, then branch to line 2a0. This indicates that $r24$ was a greater number than $r25$. If no carry has occurred, go to lines 256, 258. Then, calculate $r24 - r25$. If the values are not the same between $r24$ and $r25$, branch to line 29c. This means that $r25$ was a greater number than $r24$. If the values were the same, compare the next two words by executing the following lines. Repeat this process until the comparison operation is finally completed. In other words, the vulnerability appears in the fact that the processing method in the branch statement varies depending on the result of the comparison operation. This is an important point to understand for design of countermeasure.

4.2 Single Trace Analysis on the Comparison Operation based Constant-time CDT Sampling

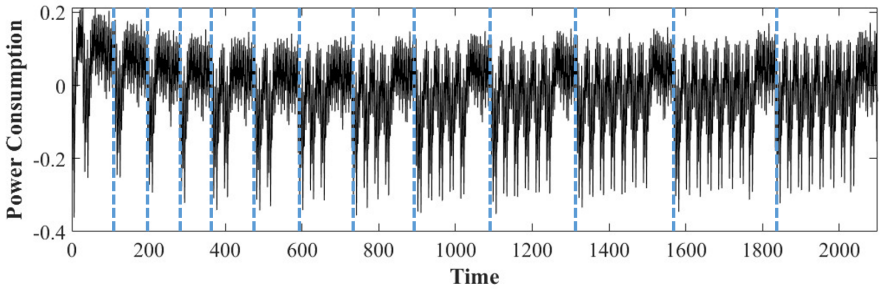

Fig. 3: The power consumption trace of maximum r on $uint64\_t$

The *BaseSampler* function utilized in SOLMAE implements CDT sampling through comparison operations, as depicted in Listing 1.1. The comparison operations are performed between two operands of the $uint64\_t$ data type: a random variable r and each the 13 values stored in the CDT table. On an 8-bit processor, these comparison operations are performed by dividing them into 8 words. The

aforementioned comparison operations have two vulnerabilities. First, the number of comparisons depends on the values being compared. Second, the value being added depends on the result of each comparison operation, i.e., an additional operation is required to add 1. Therefore, it is risky to work with data types larger than *word*.

Figure 3 shows the power consumption trace of the $CDTsampling$ when r is set to the maximum value of the $uint64\_t$ data type (i.e., $2^{64} - 1$). From the power consumption trace, it is evident that the number of comparisons with each CDT table differs, indicating variations in computation time based on the compared values.

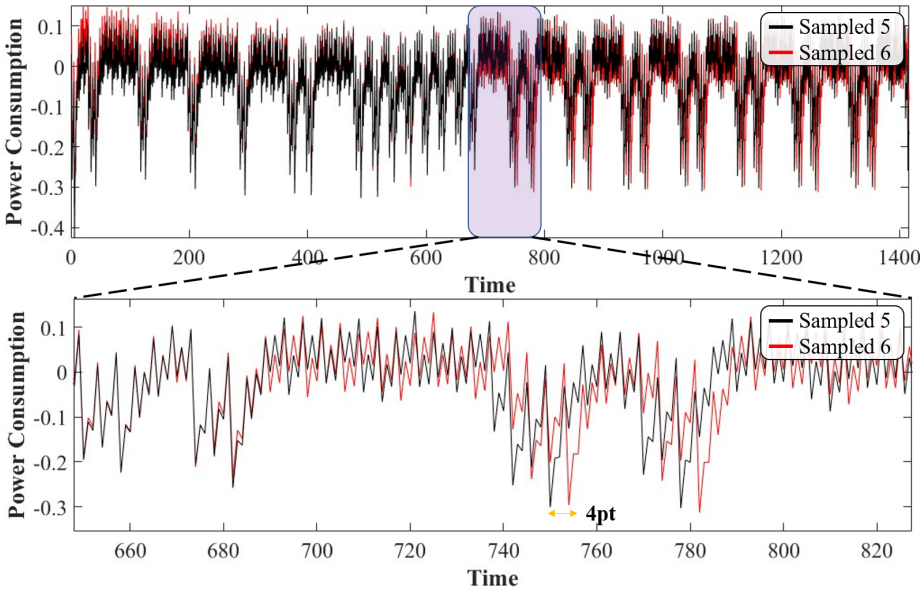

Fig. 4: Two power consumption traces differ by only one in sampling values. They differ by only one in r values

Figure 4 shows two power consumption traces with only a difference of 1 in the values of 'r.' More precisely, the return values, sampled by the difference in 'r' values, also differ by one. The noted discrepancy is a result of the optional addition operation, leading to evident distinctions between the two traces. This is also related to the data type of the resulting value returned. Since the returned data type is a unit larger than the *word*, a difference also occurs in the addition operation.These discrepancies in power consumption traces enable the visual detection of any divergence in assembly instructions.

An increment of 1 of the sampling result occurs when r is greater than or equal to value of table in the comparison between r and the value of table.

Furthermore, the values in the CDT table are arranged in ascending order. Consequently, once r becomes smaller than a particular value in the CDT table, the resulting value remains unchanged. This implies that if a comparison operation with a CDT table value greater than r is identified, the output of CDT sampling can be obtained. The power consumption traces of the first word in the comparison operation, as depicted in Figure 5, exhibit distinct shapes for the scenarios where r is greater than, equal to, and less than the value in the CDT table, respectively. The visual distinctiveness of these power traces facilitates the acquisition of the CDT sampling value. This vulnerability arises from the inherent characteristics of the weak comparison operation, as discussed earlier.

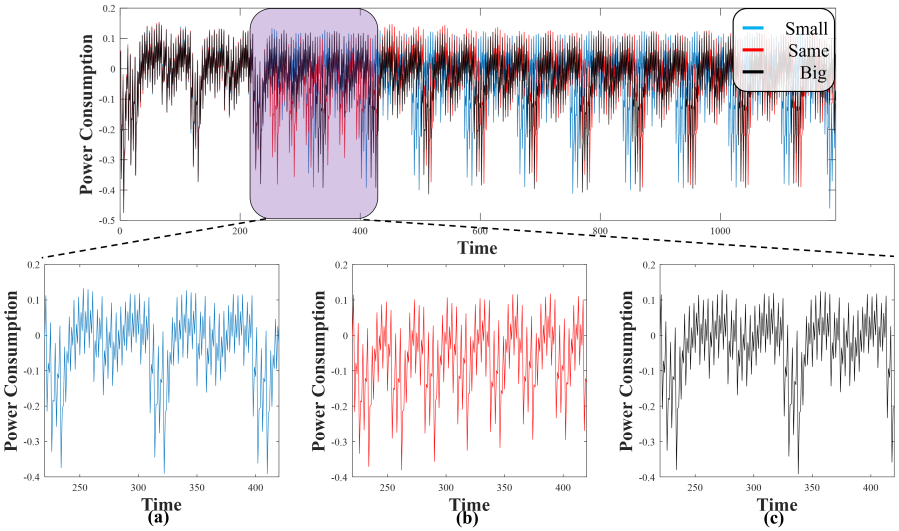

Fig. 5: The power consumption traces of CDT sampling have different shapes for each r value: (a) $A_0 < B_0$, (b) $A_0 = B_0$, and (c) $A_0 > B_0$ where A_i and B_i represent individual words.

5 Countermeasure

In the previous section, we highlighted the vulnerability of comparison operations when processing data larger than the word size of the processor. To address this issue and ensure the safety of comparison operation based constant-time CDT sampling, we propose a novel implementation method with countermeasure.

Before introducing the proposed countermeasure, we first provide an overview of trends in countermeasures related to CDT sampling. First, in [4] the CDT sampling method using Table was proposed. But it requires a large storage space.

In addition, there is also the protection of sampling through the masking method proposed by [18] and the random shuffling method proposed by [19,20]. However, But these have memory overheads and time overheads. And analysis techniques related to these are being proposed.[21]. However, since there have not been many studies related to sampling using comparison operation, a new concept of implementing CDT sampling using comparison operation has been attempted.

In previous sections, the cause of vulnerability mentioned in this paper were attributed to the varying number of clock cycles depending on the branch statement in the 8-bit AVR MCU environment. Hence, the countermeasure proposed an implementation method that eliminates the discrepancy in the number of clock cycles. The proposed secure CDT sampling algorithm in this paper is denoted as Algorithm 4. The algorithm processes the r and the CDT table in word-sized blocks, corresponding to the processing units of the processor. The values in r, CDT table that exceed the word size are divided into n word blocks. Comparison operations are performed identically each block. However, if the outcome of a comparison operation is determined in the previous block, subsequent operations are only performed, i.e., it does not affect the result. Due to the inherent nature of comparison operations, methods employing them may result in branching. Branching commands such as 'brne' and 'brcc' are commonly used. In AVR instruction sets, 'brne' and 'brcc' differ by only 1 with respect to true and false conditions, allowing for an equal adjustment in the number of clock cycles for the operation. However, this implementation approach can be considered risky. Therefore, this paper introduces an assembly code that effectively eliminates the need for branch commands while implementing Algorithm 4.

Algorithm 4 STA-Resistant CDT sampling

Input : -
Output : Sampled value z
1: $z \leftarrow 0$
2: $r_i \xleftarrow{\$} [0, 2^{word\ size})$ uniformly random with $i = 0$ to n
3: **for** $i = 0$ to $Table\_size - 1$ **do**
4: $gt \leftarrow 0, lt \leftarrow 0$
5: **for** $j = 0$ to $n - 1$ **do**
6: $gt \mid= (\neg(gt \mid lt))\&(r_j > CDT_{i,j})$
7: $lt \mid= (\neg(gt \mid lt))\&(r_j < CDT_{i,j})$
8: **end for**
9: $z \mathrel{+}= 1 \oplus lt$
10: **end for**
11: **return** z

```
<STA-Resistant CDT sampling>:
    ...
    278:            ldi  r18, 0x00  ; 0
    27a:            cp   r22, r23
    27c:            adc  r18, r18
    27e:            and  r24, r18
    280:            or   r19, r24
    282:            mov  r24, r19
    284:            or   r24, r25
    286:            com  r24
    288:            ldi  r18, 0x00  ; 0
    28a:            cp   r23, r22
    28c:            adc  r18, r18
    ...
```

Listing 1.3: The comparison operation of assembly implementation code of countermeasure

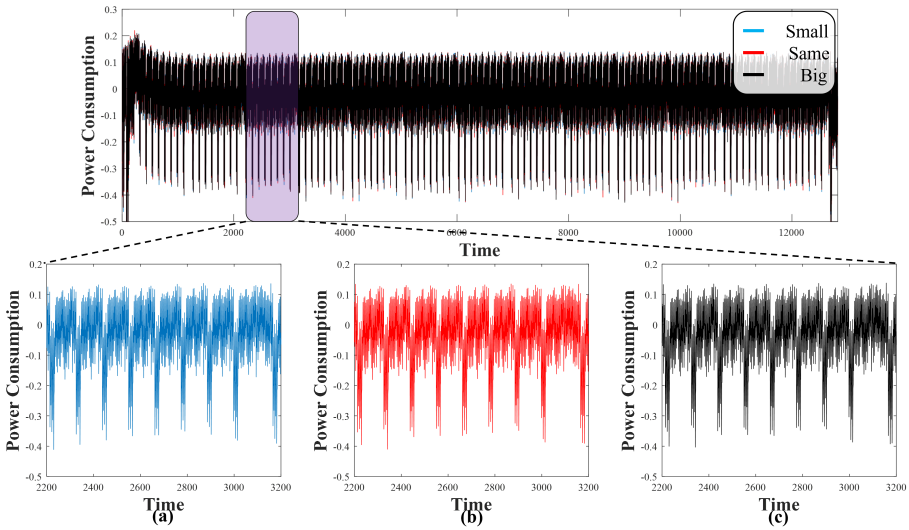

Fig. 6: The traces that overlap all three types of STA-Resident CDT sampling. And (a) $A_0 < B_0$, (b) $A_0 = B_0$, and (c) $A_0 > B_0$ where A_i and B_i represent individual words.

Listing 1.3 is a parts of the assembly code, representing the comparison operation in the proposed countermeasure. The blue and red lines in Listing 1.3 correspond to the comparison operations in Algorithm 4. Lines 278 and 288 initialize the value of register r18, where the result of the comparison operation

will be stored, to zero. Lines 27a and 28a perform comparisons between registers r22 and r23 using 'cp' commands, respectively, and store the results in the carry flag. Lines 27c and 28c execute an addition operation on the initialized r18 using the 'adc' (add with carry) instruction. During this operation, the stored carry values are combined, resulting in the storage of the comparison operation's result within r18. This approach allowed me to eliminate the need for branching instructions, thus removing the vulnerabilities previously mentioned.

Figure 6 illustrates the power consumption traces of 3 different types of the Listing 1.3 operating in the 8-bit AVR MCU. The power consumption traces (a), (b), and (c), which are fully examined by overlapping with a, b, and c, represent the corresponding power consumption traces. Similar to Figure 5, (a), (b), and (c) signify whether the most significant block of 'r' is greater than, equal to, or less than the value in the CDT table. The trace reveals that there are no discernible variations in the comparison time across different values. This serves as compelling evidence that CDT sampling demonstrates resistance against STA.

6 Conclusion and Futurework

This paper introduces a secure implementation of CDT sampling for Gaussian sampling techniques. CDT sampling is used by many algorithms to generate important values. And this paper presents an analysis of a previously unexplored vulnerability that STA in comparison operation-based CDT sampling. This paper identifies a vulnerability in which the operation time varies depending on the results of the comparison operation in 8-bit AVR MCU. The cause of the vulnerability was demonstrated through different of the number of instruction at the assembly stage. It was investigated that it was a vulnerability due to the difference in the number of clocks.

The feasibility of extracting CDT sampling outputs in real-world environments, such as AtmelXMEGA128, is demonstrated. AtmelXMEGA128 is an 8-bit AVR MCU and is used in various environments. We also employed different compiler options (0, 1, 2, 3, s) provided by Chipwhisperer in the AtmelXMEGA128 environment and verified the presence of the vulnerability across all of them. In this paper, we utilized the example of compiler option level 's,' which is set as the default among several available options. In this paper, we did not show power consumption traces for other options, as we observed that all options exhibited the same or even greater leakage. In addition, this paper deals with vulnerabilities that depend on the processor's word size and compiler. During our investigation, we observed that the number of clock cycles varied depending on the branch instruction employed. It also showed the impact of the attack by recovering the sampling value. This finding sheds light on the potential risks associated with future cryptographic algorithms that employ CDT sampling with vulnerable comparison operations, using SOLMAE as a case study. We conducted an analysis of power consumption traces to pinpoint the sections of the SOLMAE algorithm utilizing CDT sampling. This demonstrated the practical applicability of STA.

To address these concerns, a robust CDT sampling design is proposed, ensuring security against STA in real-world. To address these issues, our proposed countermeasure for CDT sampling in this paper aims to stabilize the number of clock cycles, irrespective of the branch statement used. So, First we delved into the countermeasure algorithms for CDT sampling that were previously explored. Our investigation revealed the existence of algorithms employing table-based comparison operations, masking methods, and shuffling techniques. And we present a method for implementing comparison operation based constant-time CDT sampling, designed to mitigate the security risks associated with the previously proposed STA. The algorithm is crafted to segment and store data in units processed by the processor, facilitating comparisons across all blocks. This design allows for sampling without reliance on the results of comparison operations.

In real-world implementations, caution is warranted branch statements. Branch statements, such as 'brne' and 'brcc' commands in 8-bit AVR MCU, introduce variability in clock cycles depending on the outcome of comparison operations. If the result of the branch leads to a distant address, the number of clock cycles will vary based on the outcome. In essence, it is the need for caution in employing branch statements. To address this variability, we propose a comparison operation based constant-time CDT sampling implementation method at the actual assembly code level. Instead of using branch statements, the results of comparison operations are stored in the result register using instructions that 'cp' and 'adc'. This approach ensures uniform operation time without relying on the specific outcome of the comparison operation. Additionally, this paper showed the power consumption traces using Chipwhisperer-Lite when operating proposed countermeasure algorithm in AtmelXMEGA128(8-bit AVR MCU) to demonstrate safety against STA.

The experimental environment of this paper is 8-bit AVR MCU. In the future, we plan to investigate the possibility of STA for comparison operation based constant-time CDT sampling in various environments.

References

1. P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.
2. M. Mosca, "Cybersecurity in an era with quantum computers: Will we be ready?," *IEEE Security & Privacy*, vol. 16, no. 5, pp. 38–41, 2018.
3. P. 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 16*, pp. 104–113, Springer, 1996.
4. S. Kim and S. Hong, "Single trace analysis on constant time cdt sampler and its countermeasure," *Applied Sciences*, vol. 8, no. 10, p. 1809, 2018.
5. S. Marzougui, I. Kabin, J. Krämer, T. Aulbach, and J.-P. Seifert, "On the feasibility of single-trace attacks on the gaussian sampler using a cdt," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 149–169, Springer, 2023.

6. T. Espitau, P.-A. Fouque, F. Gérard, M. Rossi, A. Takahashi, M. Tibouchi, A. Wallet, and Y. Yu, "Mitaka: a simpler, parallelizable, maskable variant of falcon," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 222–253, Springer, 2022.
7. K. Kim, M. Tibouchi, A. Wallet, T. Espitau, A. Takahashi, Y. Yu, and S. Guilley, "Solmae algorithm specifications." `https://kpqc.or.kr/1`, 2020.
8. O. Regev, "Lecture notes of lattices in computer science, taught at the computer science tel aviv university," 2009.
9. M. Ajtai and C. Dwork, "A public-key cryptosystem with worst-case/average-case equivalence," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 284–293, 1997.
10. J. Hoffstein, J. Pipher, and J. H. Silverman, "Ntru: A ring-based public key cryptosystem," in *International algorithmic number theory symposium*, pp. 267–288, Springer, 1998.
11. P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang, *et al.*, "Falcon: Fast-fourier lattice-based compact signatures over ntru," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, no. 5, pp. 1–75, 2018.
12. S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Silder, and D. Stehlé, "Crystals-dilithium: Algorithm specifications and supporting documentation," 2020.
13. O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.
14. R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber algorithm specifications and supporting documentation," *NIST PQC Round*, vol. 2, no. 4, pp. 1–43, 2019.
15. J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila, "Frodo: Take off the ring! practical, quantum-secure key exchange from lwe," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 1006–1018, 2016.
16. J. H. Cheon, D. Kim, J. Lee, and Y. Song, "Lizard: Cut off the tail! a practical post-quantum public-key encryption from lwe and lwr," in *International Conference on Security and Cryptography for Networks*, pp. 160–177, Springer, 2018.
17. J. Howe, T. Prest, T. Ricosset, and M. Rossi, "Isochronous gaussian sampling: From inception to implementation: With applications to the falcon signature scheme," in *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*, pp. 53–71, Springer, 2020.
18. T. Schneider, C. Paglialonga, T. Oder, and T. Güneysu, "Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto," in *Public-Key Cryptography–PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II 22*, pp. 534–564, Springer, 2019.
19. D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.
20. R. A. Fisher and F. Yates, *Statistical tables for biological, agricultural and medical research*. Hafner Publishing Company, 1953.
21. K. Ngo, E. Dubrova, Q. Guo, and T. Johansson, "A side-channel attack on a masked ind-cca secure saber kem implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 676–707, 2021.