Kalpa Publications in Computing

Volume 11, 2019, Pages 64–77

Proceedings of 8th International Workshop
on Security Proofs for Embedded Systems

kalpa Computing

Constructing Sliding Windows Leak from Noisy Cache Timing Information of OSS-RSA

Rei Ueno[1], Junko Takahashi[2], Yu-ichi Hayashi[3], and Naofumi Homma[1]

[1] Tohoku University, 2–1–1 Katahira, Aoba-ku, Sendai-shi, 980-8577, Japan
{ueno, homma}@riec.tohoku.ac.jp
[2] NTT Secure Platform Laboratories, Nippon Telegraph and Telephone Corporation, 3–9–11 Midori-cho, Musashino-shi, Tokyo, 180-8535, Japan
[3] Nara Institute of Science and Technology, 8916–5 Takayama-cyo, Ikoma-shi, Nara, 630-0192, Japan

Abstract

This paper presents a method for constructing an operation sequence of sliding window exponentiation from the noisy cache information of RSA, which can be used for a cache attack using sliding windows leak (SWL). SWL, which was reported in CHES 2017, is a kind of cache side-channel leak of a sequence of operations (i.e., multiplication and squaring) from software RSA decryption using the sliding window method for modular exponentiation. It was shown that an SWL attack can retrieve the secret keys of 1,024-bit and 2,048-bit RSA with non-negligible probability if the SWL is correctly captured. However, in practice, it is not always possible for an attacker to acquire a complete and correct operation sequence from cache information observation. In addition, no concrete method for deriving a fully correct operation sequence from a partially acquired operation sequence has been reported in literature. In this paper, we first show that the capture errors in an operation sequence can be evaluated based on the Levenshtein distance between correct and estimated sequences. The dynamic time warping (DTW) algorithm is employed for quantitative evaluation. Then, we present a method of accurately estimating a complete and correct operation sequence from noisy sequences obtained through multiple observations. The basic idea of the proposed method and DTW-based evaluation is to divide the acquired operation sequence into short subsequences referred to as "operation patterns." Furthermore, we show the effectiveness of the proposed method through a set of experiments performed using RSA software in Libgcrypt, which is one of the most common open source software in cryptography.

1 Introduction

In recent years, a new type of services that lease remote (high-end) servers to clients is being widely used, such as Amazon AWS and Google Cloud Platform. The deployment of such services makes various users share an identical computational resource (e.g., CPU and memory) as virtual machines (VMs). Consequently, cache attacks in such shared servers are attracting attention. A cache attack is a kind of side-channel attack that exploits the time differences of cache access information to estimate secret information [5, 8, 10, 11, 14, 16, 17]. Cache attacks

are possible if victim and attacker processes share a cache. These attacks can potentially occur in common cloud services where several VMs on a hypervisor share a cache. As cloud services are increasingly placing the VMs of multiple users in a server, the security evaluation of cryptographic software against cache attacks is highly required.

In CHES 2017, Bernstein et al. reported a cache attack on RSA software using (left-to-right) sliding window exponentiation [3]. Sliding window is one of the fastest methods for modular exponentiation [12], and it is widely employed in some open source software (OSS) in cryptography because of its performance. Cache attacks on the sliding window method are attracting considerable attention owing to their wide applicability to OSS such as GnuPG. While sliding window is evidently a non-constant-time algorithm, it was expected to be somewhat resistant to side-channel attacks including cache attacks and simple power analysis (SPA) [4]. This is because the sequence of operations (i.e., multiplication and squaring) is less informative compared to other non-constant-time algorithms such as the left-to-right binary method. However, a cache attack utilizes the sliding windows leak (SWL) observed through cache timing information during the modular exponentiation of CRT-RSA decryption. In [3], it was shown that a cache attack can retrieve the secret key of RSA software using sliding window with non-negligible probabilities if the operation sequence of sliding window is correctly captured. A cache attack has been demonstrated through an application to CRT-RSA software in Libgcrypt, which is one of the most common cryptographic OSS [1].

However, the feasibility of Bernsteins' attack is still unclear in a practical setting because SWL (i.e., the correct operation sequence) is not always correctly captured from cache timing information. In [3], Bernstein et al. mentioned that the correct SWL should be constructed from such noisy cache timing information obtained through multiple observations. In addition, the noise (i.e., incorrectly captured part due to miscellaneous reasons on measurement and computation environment) contains capture error, which indicates the undetection and misdetection of multiplication and squaring. Undetection is the overlooking of an operation, and misdetection is the observation of an operation that is not actually performed. Therefore, the length of the captured operation sequence is different from that of the actual sequence. As we cannot determine the position of misdetection/undetection, it is quite difficult to construct the correct SWL by only averaging a noisy operation sequence, even if an attacker can observe the cache timing information multiple times, and there is no known method for this purpose.

We present a method for constructing the correct SWL from noisy cache timing information (i.e., noisy operation sequence of sliding window) to evaluate feasibility of an SWL-based attack. The basic idea of the proposed method is to separate the noisy sequence of multiplication and squaring into "operation patterns." An operation pattern is defined as $\mathrm{S}^t\mathrm{M}$, which indicates that a multiplication is performed after t times of squaring. By transforming a noisy operation sequence to a pattern sequence, we can evaluate the capture errors in the operation sequence in a quantitative manner based on the dynamic time warping (DTW) algorithm [13]. In addition, we can specify the length of the actual operation sequence and the position of misdetection and undetection. In this paper, we demonstrate an experimental attack on CRT-RSA software in Libgcrypt 1.7.6 to evaluate the practicality and feasibility of an SWL-based attack using the proposed method in terms of the required number of observations and the remaining number of SWL candidates. As a result, we confirm that the proposed method can reduce the number of candidates for operation sequences to only 3 in our setup (this indicates that the number of candidates for RSA secret key can be reduced to at most 300,000), while a straight-forward estimation of location of capture errors is infeasible (For typical example, it requires approximately 2^{70} guesses of capture error locations).

Algorithm 1 Left-to-right sliding window exponentiation

Require: Base X, modulus N, exponent $E = (e_k e_{k-1} \dots e_1)_2$
Ensure: Modular exponentiation $R = X^E \bmod N$
1: **parameter** w; ▷ Predetermined maximum window size
2: **function** SlidingWindowExponentiation$_w(X, N, E)$
3: **int** $X_1 \leftarrow X$; **int** $X_2 \leftarrow X^2 \bmod N$;
4: **for** i from 1 to $2^w - 1$ **do** ▷ Precomputation
5: **int** $X_{2i+1} \leftarrow X_2 X_{2i-1} \bmod N$;
6: **end for**
7: **int** $R \leftarrow 1$; **int** $j \leftarrow k$;
8: **while** $j \geq 1$ **do** ▷ Main loop
9: **int** $z \leftarrow$ LeadingZerosOf$((e_j e_{j-1} \dots e_1)_2)$; ▷ Count leading zeros
10: **int** $j \leftarrow j - z$; ▷ Set j such that $e_j = 1$
11: **int** $l \leftarrow \min(w, j+1)$;
12: **int** $s \leftarrow$ TrailingZerosOf$((e_j e_{j-1} \dots e_{j-l+1})_2)$; ▷ Count trailing zeros
13: **int** $u \leftarrow (e_j e_{j-1} \dots e_{j-l+1})_2 >> s$; ▷ Remove trailing zeros
14: **for** t from 1 to $z + (l - s)$ **do**
15: $R \leftarrow R^2 \bmod N$;
16: **end for**
17: $R \leftarrow R X_u \bmod N$;
18: $j \leftarrow j - (l - s)$;
19: **end while**
20: **return** R;
21: **end function**

2 Preliminaries

2.1 Sliding window exponentiation

Sliding window exponentiation is one of the fastest modular exponentiation algorithms. It employs precomputation of the small odd powers of a base and windowing to reduce the number of multiplications. Algorithm 1 is the left-to-right sliding window method for modular exponentiation, which is target of this study. Here, the inputs X, N, and E correspond to the ciphertext, public key, and secret key of RSA decryption, respectively. Output R corresponds to plaintext. At Line 1, the maximum window size, w, is predetermined as a parameter. Basically, for a larger w, the computation time of the main loop (i.e., Lines 8–22) is faster while the time for precomputation and the required memory are larger. In Lines 3–6, we precompute the odd powers of X upto X^{2^w-1}. Note here that $X_{2i+1} = X^{2i+1}$. Lines 8–19 represent the main loop of this algorithm. In the main loop, we first count the number of leading zeros of $(e_j e_{j-1} \dots e_1)_2$ at Line 9 and set the pointer j such that $e_j = 1$ at Line 10. Line 11 is required to determine the maximum window size at the end of the main loop. At Line 12, we count the trailing zeros of the j-th–$(j - l + 1)$-th exponent bits $(e_j e_{j-1} \dots e_{j-l+1})_2$ as s, which is equivalent to $\log_2 \left(\mathrm{GCD}((e_j e_{j-1} \dots e_{j-l+1})_2, 2^l)\right)$, where GCD denotes the greatest common divisor. Here, $l - s$ is the temporal window size at the loop. At Line 13, we remove trailing zeros from $(e_j e_{j-1} \dots e_{j-l+1})_2$ by shifting it to the right by s and derive $u = (e_j e_{j-1} \dots e_{j-s+1})_2$. Note here that u is always odd because e_{j-s+1} should be one. After we perform the squaring operation $z + (l - s)$ times (i.e., the number of leading zeros + temporal window size) at Lines 14–16, we perform multiplication with precomputed X_u at Line 17. Finally, at Line 18, we update j for the next loop. Note here that $e_1 = 1$ because E should odd in the case of RSA.

Similar to [3], we denote multiplication and squaring by M and S, respectively, and denote operation sequence consisting of multiplication and squaring as the string of S and M such as SSM (which indicates that a multiplication operation is performed after two

squaring operations.) In Alg. 1, for example, if the exponent is a 16-bit value $E = (1000101011000011)_2$ and the window size is $w = 4$, the left-to-right operation sequence is given by SMSSSSSSMSSSMSSSSSSM. In the sequence, the first, second, third, and fourth multiplications are performed with X_1, X_5, X_3, and X_3, respectively. In this example, multiplication is performed only 4 times. In contrast, the left-to-right binary method and Montgomery ladder, which are the typical modular exponentiation algorithms, require 6 and 16 multiplications, respectively. Thus, the sliding window method is known to be one of the fastest modular exponentiation algorithms because of the precomputation, as analyzed in [9]. In addition, left-to-right sliding window exponentiation had been expected to be somewhat resistant to (cache) timing attacks and SPAs because the timing and SPA traces are less informative compared to the left-to-right binary method owing to the operand-selective multiplication (until the publication of the SWL-based attack [3]). Therefore, the left-to-right sliding window method has been widely deployed in numerous cryptographic OSS including GnuPG.

2.2 Flush+Reload

Flush+Reload is one of the most popular methods for cache attacks [16]. The use of Flush+Reload enables an attacker to know that the data in the memory of a specific location are loaded (and cached) by a victim. If the data are the code segments of multiplication and squaring for (CRT-)RSA decryption, the attacker can obtain the operation sequence of modular exponentiation via Flush+Reload. Flush+Reload is available if the attacker can run a process on a CPU core where the victim's process is running on another core, and their processes share the main memory and L3 cache. Such a scenario is practical for several contemporary cloud services that lease VMs and computational resources.

The basic idea of Flush+Reload is to repeat cache reload and flush periodically to estimate whether data are loaded and used by the victim. Flush+Reload is performed in the following three steps: (i) the attacker flushes the shared cache, (ii) the attacker waits for the victim to load target data (i.e., code segments of multiplication and squaring) from the main memory (the waiting time is referred to as slot time), and (iii) the attacker loads the target data to measure the time (i.e., clock cycles) required for the loading. For example, in x86 CPUs, loading time can be measured using the RDSTC operation. Here, if the victim loads the target data (i.e., performs multiplication and/or squaring) during the slot time, the loading time at step (iii) should be short because the data should be cached. Otherwise, the loading time should be long because the attacker should load the data from the main memory. Thus, the attacker can estimate the timing of execution of multiplication and squaring and can obtain the operation sequence via cache information.

In this paper, to perform Flush+Reload, we employ an open-source toolkit for micro-architectural side-channel named Mastik, which is available in [15].

2.3 SWL-based attack

In [3], an SWL-based attack is performed in the following four steps: (i) First, the attacker estimates the execution timing of operations related to modular exponentiation (i.e., multiplication and squaring) through a cache attack such as Flush+Reload. (ii) Then, the estimated execution timing is translated to the operation sequence, which represents the order of execution of multiplication (M) and squaring (S). (iii) The key bits are partially estimated using an algorithm proposed in [3], which is based on the windowing rule of the left-to-right sliding window method. (iv) Finally, the entire key is recovered from the partial key by employing the algorithm developed by Heninger and Shacham [7]. It is shown that this attack can reduce the

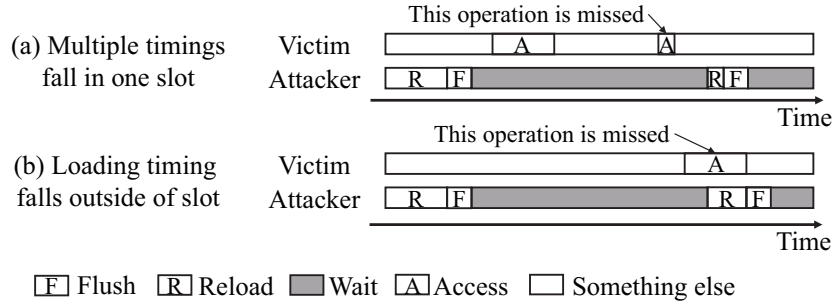


Figure 1: Example of capture errors.

space of a secret key to less-than 10^6 for all keys of 1,024-bit CRT-RSA, and to 2^6 for 13% keys of 2,048-bit CRT-RSA if the attacker can obtain a complete and correct operation sequence at Step (ii).

An SWL-based attack requires Flush+Reload to estimate the execution timing of multiplication and squaring. We describe how to apply Flush+Reload to the implementation in Libgcrypt in order to estimate the execution timing and operation sequence. As the Libgcrypt implementation employs an identical integer multiplication and modular reduction function for both multiplication and squaring[1], the execution timing of the function is not informative. Therefore, we apply probes on the pre-operations and post-operations of the function to distinguish between multiplication and squaring. The sliding window method selects a value (i.e., X_{2i+1}) from a precomputed table before performing a multiplication. In addition, there is a procedure for returning to the start of main loop after the multiplication. We also obtain the execution timings of the above two operations (i.e., operand selection and jump operation) to distinguish between squaring and multiplication.

An SWL-based attack should accurately acquire the execution timing of multiplication and squaring to obtain a correct operation sequence. However, in practice, it is quite difficult to obtain such a correct operation sequence because the execution timing estimated by cache attacks, including Flush+Reload, includes noise, which is referred to as capture error. Capture error is classified into misdetection and undetection. Misdetection indicates that the attacker observes an operation via the cache information when the operation is actually not performed. Misdetection occurs when other data in the same cache line are loaded to the cache. Undetection indicates that the attacker misses the execution of an operation. Figure 1 shows two typical examples of undetection. Undetection occurs when (a) an operation is performed multiple times during a slot time or (b) the execution timing of an operation overlaps with the attacker's reload operation. Such undetection is closely related to the slot time, which is determined by the attacker. A shorter slot time mitigates (a), while a longer time mitigates (b). As an integer multiplication and a modular reduction in Libgcrypt (i.e., the target library in this study) require 30,000 clock cycles on average, the slot time should be shorter than 30,000 clock cycles. On the contrary, Allan et al. reported that a slot time longer than 10,000 clock cycles would be effective for preventing (b) [2].

Nevertheless, it is still difficult to obtain a correct and complete operation sequence from a

[1]In this paper, "multiplication" denotes the multiplication in the sliding window method as opposed to squaring, while "integer multiplication" indicates just a code segment which is used for both multiplication and squaring.

single measurement of cache information for RSA decryption. This indicates that the length of the acquired operation sequence probably differs from that of the actual sequence. While the attacker would be able to observe the execution timing multiple times, there is no known method for constructing a correct operation sequence from such noisy execution timing in literature. Even though the authors of [3] stated that a correct operation sequence could be obtained by averaging numerous execution timings obtained through multiple observations, only averaging is not useful for correcting stochastic capture errors. This indicates that it is quite difficult to align the obtained noisy operations sequence and to find the correspondence of S and M among them.

It is also infeasible to estimate the location of capture errors. The length of an (actual) operation sequence depends on the window size, w, and the number of zeros counted as leading zeros. The ratio of key length to the number of zeros is approximately given by $1/(w+1)$. In the 1,024-bit CRT-RSA in Libgcrypt, where the length of secret keys is 512 bits and w is four, the average length of an operation sequence is 614. For example, if the length of the actual and estimated operation sequences are 610 and 600, respectively, the number of possible locations of capture error is given by $\binom{610}{10} = 1.2 \times 10^{21} \approx 2^{70}$, which is too large to estimate a correct operation sequence in an exhaustive search. A method for constructing a correct and complete operation sequence from noisy sequences is required to evaluate the feasibility of SWL-based attacks.

3 Quantitative Analysis of SWL Availability

We first show the characterization of the execution timing and estimated operation sequence observable via Flush+Reload. While this preliminary evaluation is performed in a similar manner to [3], to the best of the authors' knowledge, this is the first report on such a comprehensive characterization of the estimated operation sequence for evaluating SWL availability. In this experiment, we use an RSA software from Libgcrypt 1.7.6 compiled by gcc 4.8.5 with option `-O2`, employ a public toolkit for micro-architectural side-channel attack named Mastik [15], and run the RSA software on a Linux (CentOS 7.4) PC with an Intel Core i5-3470. We set the slot time of Flush+Reload to 10,000 clock cycles. In the reload step, we assume that the data should be loaded from the cache if the loading clock cycle is less than 100. Otherwise, the data are assumed to be loaded from the main memory. In addition, similar to [3], we employ a performance degradation attack (PDA), which increases the number of clock cycles (i.e., latency) required for executing target operations [2]. The PDA is useful for decreasing the undetection (b). Thus, as mentioned in Section 2.3, we apply four probes in total on the calls of integer multiplication and modular reduction (for each multiplication and squaring), operand selection, and the jump operation.

Figure 2 shows an example of a cache timing trace during CRT-RSA decryption, where the horizontal axis denotes the slot index and the vertical axis denotes the loading time (i.e., the number of clock cycles) for the attacker's reload operation. Here, the lines denoted by "Integer multiplication" and "Modular reduction" show the time required for reloading the code segments of integer multiplication and modular reduction. Note that a pair of integer multiplication and modular reduction is used per multiplication or squaring in the sliding window method. Additionally, we cannot distinguish whether multiplication or squaring is performed only from the traces of integer multiplication and modular reduction. The lines denoted by "Operand selection" and "Jump" enable us to perform this discrimination, that is, if the attacker observes either or both of them, a multiplication should be performed; otherwise, a squaring should be performed. For example, around a slot index of 1655, a squaring operation would be performed

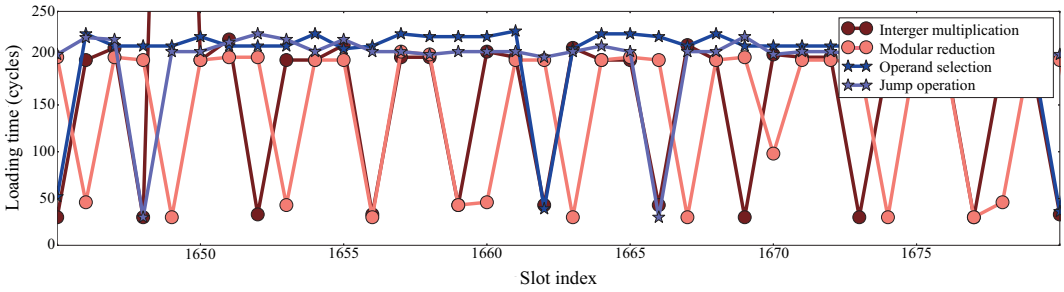


Figure 2: Example of cache timing trace.

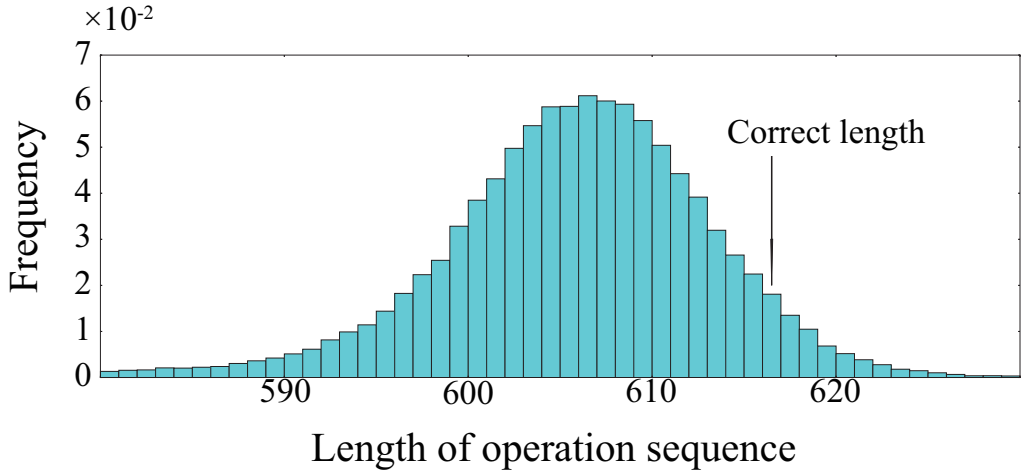


Figure 3: Histogram of length of observed operation sequences.

because integer multiplication and modular reduction are loaded from the cache but operand selection and jump operation are loaded from the main memory. In contrast, multiplication would be performed around a slot index of 1665. Thus, from Fig. 2, we can estimate that the operation sequence corresponding to the cache timing trace at slot indices 1650–1665 is SSSM (however, it may contain capture errors).

To evaluate the influence of capture errors on the feasibility of SWL-based attacks, we perform Flush+Reload on CRT-RSA decryption with a fixed key to obtain cache timing traces. Figure 3 displays the histogram of the length of the operation sequence estimated by the obtained cache timing trace, where the length of the correct operation sequence is 617. From Fig. 3, we confirm that the lengths of the estimated operation sequence vary and most of them are shorter than the length of the correct sequence. This indicates that undetection should be dominant among capture errors.

We employ Levenshtein distance (a.k.a edit distance) to evaluate the similarity between estimated operation sequences [6]. The Levenshtein distance between two strings of characters is defined as the minimum number of edit operations, that is, insert, delete, and replace. In the

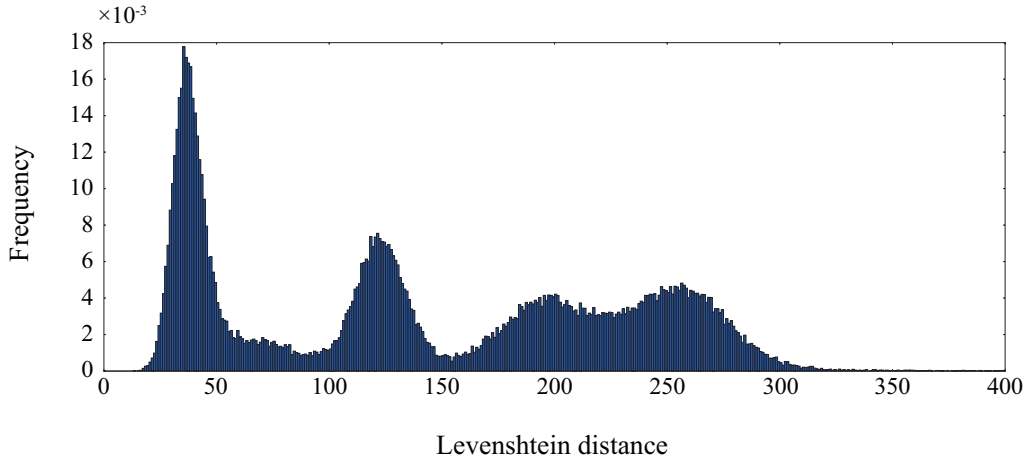
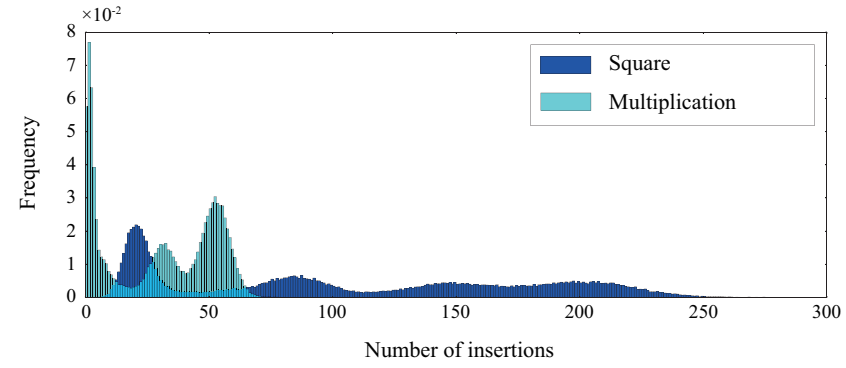


Figure 4: Histogram of Levenshtein distance.

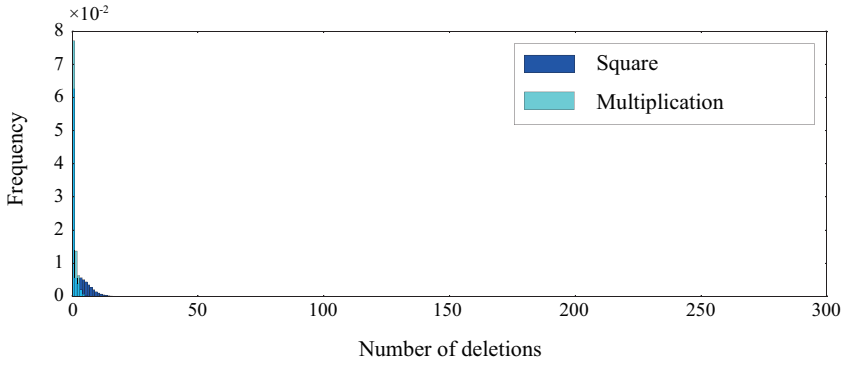
context of Flush+Reload, the characters are given by M and S, and insert, delete, and replace correspond to the undetection, misdetection, and misrecognition of M and S, respectively[2]. Figure 4 shows the histogram of the Levenshtein distance between the correct and estimated operation sequences, and Figs. 5 (a), (b), and (c) show the histograms of number of insert, delete, and replace operations, respectively. From these figures, we confirm that most capture errors are caused by the undetection of operations and the pattern of capture errors varies for each measurement. Undetection occurs in every observation, while misdetection and misrecognition occur rarely. This leads to the following important heuristic: a longer observed operation sequence contains less capture errors. However, as the locations of insert are unknown to the attacker, she cannot align noisy operation sequences by finding the correspondence between them, as mentioned in Section 2.3. Thus, it is impossible to obtain a correct operation sequence by averaging noisy and stochastic operation sequences. Moreover, the histograms of Levenshtein distance shown in Figs. 4 and 5 represent a complex and composite distribution and not a simple Gaussian distribution. This indicates that we require a dedicated strategy to evaluate the location and type of capture errors in a quantitative manner.

In this paper, we employ the DTW algorithm to quantitatively evaluate capture errors [13]. This algorithm is frequently used for the visualization of Levenshtein distance. First, we transform the operation sequence into another representation to which DTW can be applied. The operation sequence generated by the sliding window method should repeat one or more squaring operations followed by a multiplication. Therefore, the operation sequence of sliding window exponentiation is represented by $\mathrm{S}^{t_1}\mathrm{M}\ \mathrm{S}^{t_2}\mathrm{M}\ \ldots\ \mathrm{S}^{t_u}\mathrm{M}\ \ldots\ \mathrm{S}^{t_v}\mathrm{M}$. where t_u is an integer greater than one. We refer to one $\mathrm{S}^{t_u}\mathrm{M}$ as an operation pattern, and v denotes the number of operation patterns. Each operation pattern can be represented by an integer (i.e., t_u). In other words, we can transform the operation sequence to a stacked bar chart of t_u's representing the operation pattern. Figure 6 illustrates an example of the translation of an operation sequence to pattern sequence and a stacked bar chart. In the proposed method, we represent the operation sequence by a stacked bar chart.

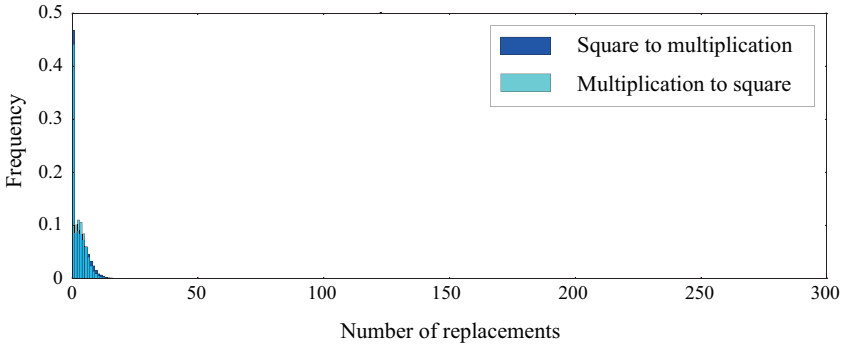
[2]We define the direction of insertion, deletion, and replacement as from the captured sequence to the correct one.



(a)



(b)



(c)

Figure 5: Histogram for edit operations: (a) insert, (b) delete, and (c) replace.

DTW is an algorithm for measuring the similarity among multiple time series with different lengths. DTW finds the correspondence of a point of a series to other points in order to align the multiple time series. Figure 7 shows an example of alignment by DTW. Here, DTW aligns two time series, $\mathbf{A} = (a_1, a_2, \ldots, a_x, \ldots, a_{10}) = (2, 6, 6, 8, 4, 9, 6, 5, 3, 4)$ and $\mathbf{B} = (b_1, b_2, \ldots, b_y, \ldots, b_8) = (2, 6, 5, 4, 8, 6, 5, 3)$. In the two-dimensional table, the

Operation sequence	SSSMSSSSSSSMSSSSSMSMSSSSSSSSSSSSSSSM				
Operation pattern	SSSM	SSSSSSSM	SSSSM	SM	SSSSSSSSSSSSSSSM
Assigned value (i.e., t_u)	3	7	4	1	15

Figure 6: Example of transformation of operation sequence to operation pattern and t_u.

		a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
		2	6	6	8	4	9	6	5	3	4
3	b_8	91	19	19	35	11	46	16	10	6	7
5	b_7	90	10	10	18	10	25	7	6	10	11
6	b_6	81	9	9	9	9	15	6	7	16	20
8	b_5	65	9	9	5	21	6	10	19	43	34
4	b_4	29	5	5	17	5	30	22	18	18	18
5	b_3	25	1	1	9	5	21	18	17	21	22
6	b_2	16	0	0	4	8	17	17	warping path		
2	b_1	0	16	32	68	72	121	137	146	147	151

Figure 7: Example of DTW.

value in the cell at (x, y), which indicates the DTW value between a_x and b_y, is given by $(a_x - b_y)^2 + \min(p(x-1, y-1), p(x-1, y), p(x, y-1))$, where $p(x, y)$ denotes the value of the cell at (x, y). Note here that $p(0, 0) = 0$ and $p(0, y) = p(x, 0) = \infty$ for any x and y satisfying $x \neq y$. After calculating this table from $p(1, 1)$ to $p(10, 8)$, we determine a path from $(1, 1)$ to $(10, 8)$ such that the sum of the values in the path is minimized. The path is referred to as the warping path, and it represents correspondence (i.e., alignment). In Fig. 7, the cells highlighted in orange color show the warping path, which indicates $(a_1) \sim (b_1)$, $(a_2, a_3, a_4) \sim (b_2)$, $(a_5) \sim (b_3, b_4)$, $(a_6) \sim (b_5)$, $(a_7) \sim (b_6)$, $(a_8) \sim (b_7)$, and $(a_9, a_{10}) \sim (b_8)$, where $(A) \sim (B)$ denotes that A corresponds to B. The computational complexity of DTW is given as the product of lengths of the time series. In the case of an SWL-based attack, the complexity is given by $\mathcal{O}(v^2)$, which is sufficiently feasible.

As a preliminary evaluation, we apply DTW to the correct operation pattern and a noisy pattern obtained by applying Flush+Reload to CRT-RSA decryption. Figure 8 shows the result of DTW, where the upper and lower stacked bar charts are the estimated and correct patterns, respectively. Here, the lengths of correct and observed pattern sequences are 104 and 102, respectively. Furthermore, the lengths of correct and observed pattern sequences are 616 and 611, respectively. The dotted line represents the correspondence between them. The bars denoted by red and different colors represent that the lengths of the operation patterns (i.e., t_u's) is equal and not equal to each other, respectively. The inequality of the length of an operation pattern is basically caused by the undetection of M and misdetection of S in the observed pattern. However, DTW successfully aligns the operation patterns while considering

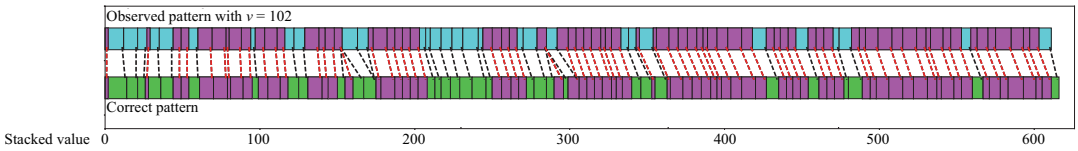


Figure 8: A result of DTW, which shows correspondence between observed and correct operation patterns.

such capture errors. Importantly, DTW appears to work well even if the number of operation patterns (i.e., v) is different from each other, that is, there is an undetection of M in the observed pattern. Thus, we can obtain the correspondence between two operation patterns and specify the location and type of capture errors for the quantitative evaluation.

4 Construct Correct Operation Sequence

4.1 Proposed method

This section presents a method for constructing the correct operation sequence based on the discussion in Section 3. We assume that the attacker can obtain numerous noisy operation sequences for a fixed key through CRT-RSA decryption.

In Section 3, we confirmed the usefulness of representing the operation sequence as a pattern sequence (i.e., a stacked bar chart given by $(t_1, t_2, \ldots, t_v)$), and that most lengths of the operation patterns obtained from Flush+Reload are basically equal to the length of the corresponding correct pattern. On the other hand, it is difficult to align the pattern sequences if there are many undetection of M's. Therefore, the proposed method employs the transformation of the operation sequence to an operation pattern, classify the pattern sequences by its length, and performs operation-pattern-wise majority voting for each classes.

More precisely, we first transform all obtained operation sequences into the corresponding pattern sequences, and we classify the pattern sequences by their length. Here, the longer pattern sequences are meaningful for us because such longer pattern sequences would have less capture errors of misdection of M's, as discussed in Section 3. This is because most of the capture errors are caused by deletion (i.e., undetection of operations). Considering only the pattern sequences with an identical length v, we determine the u-th operation pattern by the majority vote of the u-th operation patterns observed through Flush+Reload. Thus, we can obtain a likely candidate of the operation sequence and apply the algorithm of Heninger and Shacham to reduce the key space, similar to a previous work [3]. This procedure should be performed from the largest v, and should be repeated with decreasing v. The proposed method can reduce the key space of 1,024-bit CRT-RSA to $r \times 10^6$ from the noisy operation sequences observed through cache attacks, where r is the number of repetitions.

4.2 Experimental validation

This section demonstrates the validity of the proposed method through an experimental attack. We use the same experimental setup as that described in Section 3. We obtain 100,000 cache timing traces for a fixed key via Flush+Reload, in which the length of the correct pattern

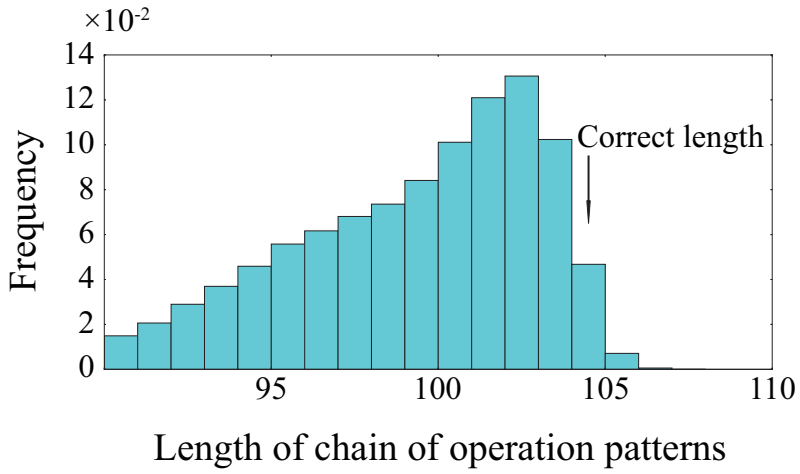


Figure 9: Histogram of length of observed pattern sequences.

sequence is 104. As a result, we confirm that we successfully construct the correct operation pattern (and sequence) from noisy observed operation sequences when v is equal to the correct length.

Figure 9 shows the histogram of the length of the observed pattern sequences, where the maximum observed length is 106 and the mode is 102. Given that the correct length is 104, Fig. 9 validates our heuristic—"a longer observed pattern sequence would have less capture errors and would be close to the correct pattern." At least, the length of the correct pattern sequence should be longer than the mode. Thus, the proposed method performs the majority voting from the largest v, which would allow for efficient estimation. In this case, as the maximum length in Fig. 9 is 106, we find the correct key using the proposed method at the third execution of Heninger-Shacham algorithm following from operation-pattern-wise majority vote. As a successful SWL-based attack can reduce key space to at most 10^6 as described in [3], we can retrieve the secret key with an exhaustive search of at most 3×10^6, which is sufficiently feasible. Thus, we confirm the effectiveness of the proposed method and the practicality of SWL-based attacks.

5 Conclusion

This paper presents a quantitative analysis of feasibility of SWL-based attacks in terms of the availability of cache timing leaks and a method for constructing a correct operation sequence from noisy and stochastic operation sequences. We employ a representation of operation sequences, named operation patterns, to which the DTW algorithm can be applied for evaluation. This leads to a heuristic that indicates that the length of the chain of observed operation patterns would have less capture errors. The proposed method for constructing correct operation patterns also employs the transformation to operation patterns. The result of an experimental attack on OSS-RSA in Libgcrypt shows that the proposed method accurately estimates the correct operation sequence from noisy observed sequences and it can retrieve the secret key of CRT-RSA within practical complexity.

A quantitative analysis of the number of cache timing traces required for a successful attack is part of future work. In addition, while DTW and Levenshtein distance are used only for evaluating the operation patterns but not used for the reconstruction method in this paper, it would be interesting to develop a method for estimating the correct operation pattern by combining all pattern sequences even with different lengths based on DTW, for improving the efficiency of estimation.

Acknowledgments

We are grateful to Mr. Hayato Mori for his cooperation. This research has been supported by JSPS KAKENHI Grants No. 17H00729 and No. 19H21526.

References

[1] GNU Privacy Guard. https://www.gnupg.org, 2017.

[2] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop van de Pol, and Yuval Yarom. Amplifying side channels through performance degradation. In *ACM Annual Conference on Computer Security Applications (ACSAC)*, pages 422–435, 2016.

[3] Daniel J. Bernstein, Joachim Breitner, Daniel Genkin, Leon Groot Bruinderink, Nadia Heninger, Tanja Lange, Christine van Vredendaal, and Yuval Yarom. Sliding right into disaster: Left-to-right sliding windows leak. In *International Conference on Cryptographic Hardware and Embedded Systems*, volume 10529 of *Lecture Notes in Computer Science*. Springer, 2017.

[4] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[5] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering*, 8(1):1–27, 2018.

[6] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[7] Nadia Heninger and Hovov Schacham. Reconstructing RSA private keys from random key bits. In *Advances in Cryptology—CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Scienece*, pages 1–17. Springer, 2009.

[8] Mehmet Sinan Inci, Berk Gülmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Cache attacks enable bulk key recovery on the cloud. In *International Conference on Cryptographic Hardware and Embedded Systems*, volume 9813 of *Lecture Notes in Computer Science*, pages 368–388. Springer, 2016.

[9] Cetin Kaya Koç. Analysis of sliding window techniques for exponentiation. *Computers & Mathematics with Applications*, 30(10):17–24, 1995.

[10] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamgurg, Moritz Lipp, Stefan Mangard, Thomas Presher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[11] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[12] Alfred j. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[13] H. Sakoe and S. Chiba. Dynamic programming alogrithm optimization for spoken word recognition. *IEEE Transasctions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.

[14] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[15] Yuval Yarom. Mastik: A micro-architectural side-channel toolkit. https://cs.adelaide.edu.au/~yval/Mastik/, Oct 2018.

[16] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium*, 2014.

[17] Yuval Yarom, Daniel Genkin, and Nadia Heninger. CacheBleed: A timing attack on OpenSSL constant-time RSA. *Journal of Cryptographic Engineering*, 8(1):1–27, 2018.