IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 2, pp. 241–269. DOI:10.46586/tches.v2023.i2.241-269

How Secure is Exponent-blinded RSA–CRT with Sliding Window Exponentiation?

Rei Ueno[1] and Naofumi Homma[1]

Tohoku University, 2–1–1 Katahira, Aoba-ku, Sendai-shi, Miyagi, 980-8577, Japan
rei.ueno.a8@tohoku.ac.jp, naofumi.homma.c8@rtohoku.ac.jp

Abstract. This paper presents the first security evaluation of exponent-blinded RSA–CRT implementation with sliding window exponentiation against cache attacks. Our main contributions are threefold. (1) We demonstrate an improved cache attack using Flush+Reload on RSA–CRT to estimate the squaring–multiplication operational sequence. The proposed method can estimate a correct squaring–multiplication sequence from one Flush+Reload trace, while the existing Flush+Reload attacks always contain errors in the sequence estimation. This is mandatory for the subsequent steps in the proposed attack. (2) We present a new and first partial key exposure attack on exponent-blinded RSA–CRT with a random-bit leak. The proposed attack first estimates a random mask for blinding exponent using a modification of the Schindler–Wiemers continued fraction attack, and then recovers the secret key using an extension of the Heninger–Shacham branch-and-prune attack. We experimentally show that the proposed attack on RSA–CRT using a practical window size of 5 with 16-, 32-, and 64-bit masks is carried out with complexity of $2^{25.6}$, $2^{67.7}$, and 2^{161}, respectively. (3) We then investigate the tradeoffs between mask bit length and implementation performance. The computational cost of exponent-blinded RSA–CRT using a sliding window with a 32- and 64-bit mask are 15% and 10% faster than that with a 128-bit mask, respectively, as we confirmed that 32- and 64-bit masks are sufficient to defeat the proposed attack. Our source code used in the experiment is publicly available.

Keywords: Cache attack · Simple power analysis · Partial key exposure attack · Exponent blinding · Sliding window exponentiation · Side-channel attack · RSA–CRT

1 Introduction

1.1 Background

The left-to-right sliding window is one of the fastest modular exponentiation algorithms for implementing RSA cryptosystem. Due to its efficiency, the sliding window has been used in many RSA–CRT implementations, including one provided in major open-source cryptographic software libraries (*e.g.*, Libgcrypt). The sliding window cannot be implemented in a constant-time manner owing to its inherent features. However, the sliding window was somehow believed to be secure against a simple power analysis (SPA)-like side-channel attack [KJJ99], because the attacker cannot completely know the exponent (*i.e.*, the secret key of RSA–CRT) from a squaring–multiplication operational sequence of the sliding window obtained *via* a side-channel. In CHES 2017 [BBG+17], it was shown that this was not true—an SPA-like attack could fully recover the RSA–CRT secret key by means of a Heninger–Shacham partial key exposure attack [HS09] with an application to Libgcrypt implementation. The previous work experimentally demonstrated that, using a

Received: 2022-10-15 Accepted: 2022-12-15 Published: 2023-03-06

cache side-channel realized by Flush+Reload, their attack could perform a full key recovery of 1,024-bit and 2,048-bit RSA–CRT implemented using a left-to-right sliding window of Libgcrypt with success rates of 100% and 13%, respectively. Due to the disclosure of the attack, Libgcrypt has applied exponent blinding to the RSA–CRT implementation to prevent this attack.

Currently, no side-channel attack applicable to exponent-blinded RSA–CRT implementation with sliding window is known[1]; thus, it is considered secure against side-channel attacks with a preserved performance. Meanwhile, there is also no known method for quantitatively evaluating its security, which naturally raises an important question: *How secure is the exponent-blinded RSA–CRT with a sliding window?* The Libgcrypt RSA–CRT implementation uses a 128-bit mask for the exponent blinding, while some implementations employ (have employed) a 20-bit or 32-bit mask [FKJM+06]. However, nobody can know whether each 20-bit, 32-bit, or 128-bit mask is insufficient or sufficient to guarantee the security against a side-channel attack. If we can evaluate the security in a quantitative manner, we may use a tightly short mask for the exponent blinding, which improves the implementation performance and pushes the limits of fast RSA implementation with resistance to side-channel attacks.

1.2 Our contributions

This paper presents the first security evaluation of the exponent-blinded RSA–CRT implementation using a sliding window. We present a cache attack applicable to it, and then show that a 128-bit mask is sufficiently secure to protect RSA–CRT with a sliding window with regard to the proposed attack. Our major contributions include an improved access-driven cache attack for RSA–CRT and a new partial key exposure attack for exponent-blinded random-bit leak. The proposed attack can be performed with a significantly less complexity than an attack using a straightforward guess. However, we demonstrate that the key recovery would be still infeasible in practice, even for a 32-bit mask, even if we use the proposed sophisticated attack, which improves and combines the state-of-the-art methods for cache attacks and partial key exposure attacks on RSA–CRT. Note that, although we propose a new cache attack and partial key exposure attack on RSA–CRT, this paper aims at evaluating the security of RSA–CRT implementation with regard to mask bit length from theoretical perspectives, and we do not state that the current implementations have vulnerabilities. Actually, we show the possibility that a full key recovery of RSA–CRT sliding-window implementation with a 20-bit mask would be feasible (which is the first report on the full key recovery on such an implementation), but our analyses reveal that the proposed attack on more than 32-bit mask would be infeasible in the current situation.

The proposed attack consists of four steps: (i) improved Flush+Reload (an access-driven cache attack) on RSA–CRT decryption/signing[2], (ii) reversing partial bits of the exponent from the cache trace as well as [BBG+17], (iii) modified Schindler–Wiemers continued fraction attack to estimate the upper bits of the respective secret prime p and q, the upper bits of the blinded exponents, and the mask bits [SW17], and (iv) extended Heninger–Shacham partial key exposure attack with regard to exponent blinding. Although we use an existing method for Step (ii) as it is, we present new improvements/modifications for Steps (i), (iii), and (iv), which are essential for key-recovery attacks on exponent-blinded RSA–CRT with sliding window exponentiation.

We analyze the complexity and (in)feasibility of the proposed attack. The attack complexity depends on the mask bit length, denoted by b. For Step (iii), we require $2^{25.6}$,

[1] Although Fouque *et al.* presented a power analysis attack on exponent-blinded RSA using sliding window without CRT [FKJM+06], it remains unknown how to extend it to RSA–CRT.

[2] The proposed attack can be also mounted using power analysis (*e.g.*, SPA), if the attacker can completely distinguish squaring and multiplication from one trace without any error.

$2^{67.7}$, and 2^{161} times continued fraction expansions for $b = 16$, 32, and 64 to break the 1,024-bit RSA–CRT implementation, whereas a straightforward attack using a naïve guess of unrecovered bits after Step (ii) requires more-than 2^{500} complexity [OHK19] (as the existing Heninger–Shacham attack is inapplicable). Our evaluation results indicate that a 64-bit mask would be sufficient to protect the 1,024-bit RSA–CRT decryption/signing with a sliding window against the proposed attack, as the complexity of 2^{161} would be excessive to break 1,024-bit RSA. This leads to the conclusion that even a 32-bit mask may be sufficient for the protection against the proposed attack, because it was mentioned in [SW17] that the practically feasible number of the continued fraction expansions would be at most 2^{60}.

Reducing the mask bit length contributes to the improvement of implementation performance. We analyze the performance of exponent-blinded RSA–CRT using the sliding window with different mask bit length to counter the proposed attack. We demonstrate that exponent-blinded RSA–CRT decryption/signing with 64-bit and 32-bit masks requires approximately 15% and 10% fewer modular multiplications than that with a 128-bit mask, respectively. In addition, we experimentally confirmed the reduction of computational time; the 1,024-bit RSA–CRT decryption with 128-bit, 64-bit, and 32-bit masks was completed within 1.05 ms, 0.964 ms, and 0.869 ms on average in our environment, respectively. Though the designer should determine the mask bit length by considering all possible attacks depending on the application scenario, the obtained result would be one major factor to determine the proper mask bit length with regard to the cache/SPA-like attacks which combines the state-of-the-art.

Our source code used in the experiment is publicly available at https://github.com/ECSIS-lab/TCHES_UH23.

Remark 1. In this study, we focused on SPA-like attacks that utilize the S–M sequence in the exponentiation and/or partial bits of exponent, because this type of attack works with a cache attack. If we can obtain more side-channel information than the S–M sequence, we can adopt more sophisticated power analysis attacks. For example, there are some attacks that exploit the collision of multiplication operand(s) [Wal08, HMA+08, CFG+10, HKT12, SSS15] and/or utilize statistical tools for power traces such as correlation, clustering, and machine learning [BJ13, PITM14, PCBP21]. The proposed attack is likely to be further improved through combination with the above attacks, if detailed power traces are available for the attacker. For example, if the attacker can detect collisions of multiplication operands in a horizontal power trace, the attacker might know more bits of the exponent than the reversing algorithm used in Step (ii).

2 Preliminaries and Related Works

2.1 RSA–CRT

Let p and q be a pair of random primes used for the RSA cryptosystem. Let (e, N) be the public key, where e is usually given by $2^{16} + 1$ and $N = pq$. Let (d, p, q) be the secret key, where $ed \equiv 1 \bmod \phi(N)$ (ϕ is Euler's totient function and $\phi(N) = (p-1)(q-1)$ here). The (textbook) RSA encryption and decryption are expressed as

$$c \equiv m^e \bmod N,$$
$$m \equiv c^d \bmod N,$$

respectively, where m is the plaintext and c is the corresponding ciphertext. By contrast, the RSA signing for message m is expressed as

$$\sigma \equiv \big(H(m)\big)^d \bmod N,$$

Table 1: Modular exponentiation algorithms used in RSA software

Library	Version (as of 2021)	Algorithm
Botan	2.15.0	Fixed window
Bouncy Castle	1.8.9	Sliding window
cryptlib	3.4.5	Fixed window
Crypto++	8.3	Fixed window
GnuTLS	3.6.15	Sliding window
Libgcrypt	1.8.7	Sliding window
mbedTLS	2.22.0	Sliding window
Nettle	3.6.0	Fixed window
OpenSSL	1.1.1h	Fixed window
WolfCrypt	4.5.0	Fixed window

where H is a cryptographic hash function with padding and σ is the corresponding signature. The signature is verified by examining whether

$$H(m) \equiv \sigma^e \bmod N.$$

As a key-recovery side-channel attacker usually aims to recover the exponent d at the modular exponentiation for both cases of decryption and signing, we focus on the modular exponentiation of $c^d \bmod N$ throughout this paper; nonetheless, our analysis can be also applied to signing.

Modular exponentiation with the secret key in the RSA cryptosystem is usually performed based on Chinese Reminder Theorem (CRT) for reduced computational cost. In RSA–CRT decryption, we first compute

$$\begin{aligned} m_p &\equiv c^{d_p} \mod p, \\ m_q &\equiv c^{d_q} \mod q, \end{aligned}$$

where $d_p \equiv d \bmod \phi(p)$ and $d_q \equiv d \bmod \phi(q)$ ($\phi(p) = p - 1$ and $\phi(q) = q - 1$). We then reconstruct m using CRT. The operand and exponents in RSA–CRT are half-length of those in RSA without CRT, which yields 2–4 times faster computation.

2.2 Sliding window exponentiation

The sliding window is one of the fastest modular exponentiation algorithms using precomputation. Due to its efficiency, the sliding window method has been widely and practically employed in many RSA–CRT implementations, including one provided in Libgcrypt, which is used for the experiment in this study. Libgcrypt has been widely deployed in many real-world applications as a part of GnuPG and OpenPGP. Table 1 summarizes the exponentiation algorithms used in the RSA implementation in major open-source cryptographic libraries. Sliding and fixed window exponentiations are deployed due to its high performance and some levels of side-channel resistance. Moreover, some cryptographic libraries provide an option of blinding exponent and/or message to prevent side-channel attacks.

Algorithm 1 is the left-to-right sliding window for base c, modulus N, and exponent d with a bit length l. At Lines 3–6, we first precompute up-to the $(2^w - 1)$-th odd powers of the base (*i.e.*, $c^1, c^3, \ldots, c^{2^w-1}$), where w is the maximum window size. Lines 8–19 constitute the main loop of the sliding window exponentiation. A loop consists of a sequence of squarings followed by a multiplication, where the number of squarings is dependent on the exponent (*i.e.*, secret key). At Line 9, we first count the leading zeros of the remaining exponent bits as z to determine the location of the temporal window at Line 10. Then, at Line 12, we count the trailing zeros in the maximum window size (or all remaining exponent bits when $w > i$, as in Line 11) as t to determine the temporal

Algorithm 1 Left-to-right sliding window exponentiation

Input: Base c, modulus N, and exponent $d = (d_l d_{l-1} \dots d_1)_2$
Output: Modular exponentiation result $m \equiv c^d \bmod N$

```
 1: parameter w;                                          ▷ Predetermined maximum window size
 2: Function SlidingWindow_w(c, N, d)
 3:     int c_1 ← c; int c_2 ← c^2 mod N;
 4:     for j from 1 to 2^{w-1} − 1 do                    ▷ Precomputation
 5:         int c_{2j+1} ← c_2 c_{2j-1} mod N;
 6:     end for
 7:     int m ← 1; int i ← l;
 8:     while i > 0 do                                    ▷ Main loop
 9:         int z ← LeadingZerosOf((d_i d_{i-1} ... e_1)_2);   ▷ Count leading zeros
10:         int i ← i − z;                                ▷ Set i such that d_i = 1
11:         int k ← min(w, i);
12:         int t ← TrailingZerosOf((d_i d_{i-1} ... d_{i-w+1})_2);   ▷ Count trailing zeros
13:         int u ← (d_i d_{i-1} ... d_{i-k+1})_2 ≫ t;    ▷ Remove trailing zeros
14:         for f from 1 to z + (k − t) do
15:             m ← m^2 mod N;                            ▷ Squaring
16:         end for
17:         m ← m c_u mod N;                              ▷ Multiplication
18:         i ← i − (k − t);
19:     end while
20:     return m;
21: end Function
```

window size as $w - t$ (or $i - t$). At Line 13, we determine the multiplicand according to the value of the temporal window u. At Lines 14–16, we perform $z + (k - t)$ squarings, followed by a multiplication with a precomputed value c_u at Line 17.

As in the previous studies (*e.g.*, [BBG+17, UTHH21]), the operation sequence of squarings and multiplications in sliding window exponentiation is represented using two symbols S and M, which denote squaring and multiplication, respectively. For example, a string of SSM denotes an operation sequence of two squarings followed by one multiplication. If the exponent d is given by a 20-bit value $(1101\,1010\,1000\,0110\,0111)_2$ as an example, Algorithm 1 with a maximum window size $w = 4$ computes $c^d \bmod N$ using a left-to-right operation sequence of SSSSM SSSM SSM SSSSSSM SSSSM, where the multiplicands of the first, second, third, fourth, and fifth multiplication are c^{13}, c^5, c^1, c^3, and c^7, respectively.

Although sliding window exponentiation requires precomputation, this method with an optimal maximum window size w (in terms of computational time) can achieve one of the fastest modular exponentiations, as analyzed in [Koc95]. Here, the optimal maximum window size w depends on the bit length of the exponent, because a larger w yields fewer multiplications in the main loop at the cost of precomputation. In this study, we always consider $w = 5$ unless otherwise stated, as the Libgcrypt RSA–CRT implementation with exponent blinding, which is the target of this study, uses $w = 5$.[3]

2.3 Access-driven cache attack on exponentiation

The side-channel attacker on exponentiation typically estimates the operation sequence to recover the secret exponent from the side-channel information. We describe the S–M sequence estimation using Flush+Reload, which is an access-driven cache attack used in many previous works (*e.g.*, [YF14, BBG+17]). In attacks on exponentiation, Flush+Reload is frequently used to exploit the instruction-flow dependency on the secret exponent. When using Flush+Reload, it is assumed that the attacker can run a process on a CPU core while the victim's process is running on another core with the last level cache (LLC)

[3] In fact, it is optimal for the bit lengths of the exponent for the exponent-blinded 1,024-bit RSA–CRT. For a larger bit RSA–CRT, a larger window size is optimal in terms of computational time (*e.g.*, $w = 6$ for 2,048-bit and 4,096-bit RSA–CRT); however, Libgcrypt implementation uses $w = 5$ even for RSA–CRT larger than 1,024-bit.

shared. Such an assumption holds frequently true in practice, such as for cloud services that provide server(s) and virtual machines (VMs) for various users/clients, where the attacker can perform a cross-VM cross-core attack.

The basic idea of Flush+Reload is to exploit the timing difference in loading data from LLC or main memory. Loading data from LLC is much faster than from main memory. To distinguish them, the attacker repeats the following three procedures:

1. Flush: the attacker flushes the shared cache (the cache flush can be performed using, for example, the CLFLUSH operation in x86 CPUs).
2. Wait: the attacker waits for the victim's process to perform the operation depending on secret value (*i.e.*, squaring or multiplication in the case of RSA–CRT).
3. Reload: the attacker reloads the code segments related to the victim's secret.

If the victim performs a squaring or multiplication during the time slot of Wait, the code segments for squaring or multiplication would be loaded faster at Reload because they come from LLC. Otherwise, Reload would be slower because they come from the main memory. Thus, the attacker can estimate whether the victim performs a squaring or multiplication during a time slot and can obtain the S–M sequence by repeating Flush+Reload. Note that the Libgcrypt implementation uses an identical code for both squaring and multiplication to mitigate cache attacks [YF14], but estimation of the S–M sequence is still possible by probing other secret-dependent instructions (*e.g.*, the entry of the main loop) in addition to the multiplication, as in [BBG+17].

Remark 2. There is another major type of access-driven cache attack named Prime+Probe, which has been used to break the public key cryptographic implementations including ones using sliding window [LYG+15, IGI+15, IGI+16]. However, the accuracy of Prime+Probe would be usually lower than that of Flush+Reload, as a Flush+Reload attacker only has to access the target address, which yields higher speed and lower noise measurements. In fact, the conventional Prime+Probe (and even Flush+Reload) attacks on modular exponentiations require multiple measurements of the cache trace to tolerate the measurement noise. Thus, we focus on Flush+Reload, as an attack on exponent-blinded exponentiation requires a very accurate estimation of S–M sequence with no error from only one cache trace. More precisely, techniques to tolerate noise using multiple measurements of cache traces, such as clustering, averaging, and majority voting, are unavailable for attacking exponent-blinded implementations because the S–M sequence changes in every cache trace measurement due to the random mask.

2.4 Cache attack on RSA–CRT using sliding window

The cache attack by Bernstein *et al.* in CHES 2017 [BBG+17] consists of three steps: (i) estimation of the S–M sequence from the cache trace acquired *via* Flush+Reload, (ii) reversing the partial bits of the exponent from the S–M sequence according to the window determination rule of the sliding window, and (iii) recovery of full bits of the exponent from the above partial bits using the Heninger–Shacham partial key exposure attack. Bernstein *et al.* showed in [BBG+17] showed that this attack can reduce the number of key candidates to at most 10^6 for the 1,024-bit RSA–CRT with a 100% success rate, and to 2×10^6 for 2,048-bit RSA–CRT with a 13% success rate.

The reversing algorithm in Step (ii) is one of the their major proposals in [BBG+17], and it is analyzed and improved by van Vredendaal and Breitner [vV18, Bre17]. In this study, we employ van Vredendaal's and Breitner's reversing algorithm due to its optimality[4].

[4] Although the algorithm was shown to be optimal/complete in the number of recovered bits, Oonishi and Kunihiro further improved the reversing algorithm by reducing the number of candidate values which uncertain bits can take [OHK19]. However, for the ease of calculation and evaluation, we use van

The inputs of the algorithm are an S–M sequence and maximum window size w, and the output is partial bits of the corresponding exponent. Let $d' \in \{0, 1, \mathrm{x}, \underline{\mathrm{x}}\}^l$ be the estimated bit string of exponent, where x and $\underline{\mathrm{x}}$ denote an uncertain bit and l is the bit length of the exponent. Here, the underlined symbol (*i.e.*, $\underline{\mathrm{x}}$) denotes the bit positions where a multiplication is performed. Given an S–M sequence, the algorithm first determines an initial estimation $\hat{d}$, which is a string composed of only x and $\underline{\mathrm{x}}$ (*i.e.*, $\{\mathrm{x}, \underline{\mathrm{x}}\}^l$) derived by the conversion rule of S $\to$ x and SM $\to \underline{\mathrm{x}}$ (for example, $\hat{d} = \mathrm{x}\underline{\mathrm{x}}\mathrm{x}\mathrm{x}\underline{\mathrm{x}}$ for an S–M sequence of SSM SSSM). Then, according to the reverse of the temporal window determination rule of the sliding window, the algorithm derives an estimation of d (*i.e.*, d'). Let $\hat{d}_i$ and d'_i be the i-th bit of $\hat{d}$ and d', respectively. Let $\mathcal{V}$ be an integer set defined by $\{\, v \mid \hat{d}_v = \underline{\mathrm{x}} \,\}$. Here, d'_i is determined as

$$d'_i = \begin{cases} 1 & \text{if } i \in \mathcal{V} \text{ (\textit{i.e.}, } \hat{d}_i = \underline{\mathrm{x}}), \\ 1 & \text{else if there exists } v \in \mathcal{V} \text{ such that } v + w_v^- - 1 = i = v + w_v^+ - 1, \\ \mathrm{x} & \text{else if there exists } v \in \mathcal{V} \text{ such that } v + w_v^- - 1 \le i \le v + w_v^+ - 1, \\ 0 & \text{else,} \end{cases}$$

where w_v^- and w_v^+ denote the possible minimum and maximum sizes of the temporal window that include the bit position v with $\hat{d}_v = \underline{\mathrm{x}}$, respectively. Because the temporal windows should not overlap a bit position, such hypothetical minimum and maximum sizes can be determined uniquely for a given S–M sequence. More precisely, w_v^- (resp. w_v^+) can be determined by dividing $\hat{d}$ such that the size of temporal windows becomes as small (resp. large) as possible in a right-to-left (resp. left-to-right) manner. For example, an S–M sequence of SSSSM SSSM SSM SSSSSSM SSSSM is converted to $\hat{d} = \mathrm{xxx}\underline{\mathrm{x}}\mathrm{xx}\underline{\mathrm{x}}\mathrm{xx}\underline{\mathrm{x}}\mathrm{xxxxxx}\underline{\mathrm{x}}\mathrm{xxxx}\underline{\mathrm{x}}$, and then is reversed to $d' = 1\mathrm{xx}1\ 1\mathrm{x}10\ 1000\ \mathrm{xx}10\ \mathrm{xxx}1$.

As in the example, the reversing algorithm cannot fully recover the exponent bits, and the algorithm output d' contains some uncertain bits (*i.e.*, x). This is because of the fact that the attacker cannot know which multiplicand is used in the multiplication from the cache trace. The expected number of recovered bits depends on the maximum window size w. For $w = 5$ (as targeted in this study), it was shown in [OHK19] that the algorithm can recover about 41.9% bits of exponent on average from a given S–M sequence. The remaining uncertain bits are recovered by means of the partial key exposure attack introduced in Section 2.6.

In Step (ii), it is assumed that the attacker can obtain a complete and correct S–M sequence in Step (i), although the S–M sequence estimated using Flush+Reload would contain inevitable noise. In fact, the experimental evaluation in [BBG+17] showed that it was impossible to obtain a completely correct S–M sequence, and the estimated S–M trace contains 14 errors on average, despite the use of the performance degradation attack [ABF+16] to enhance the accuracy. Note that it would be relatively difficult to correct even one error in the estimated S–M sequence [OK20, UTHH21], and errors in the S–M sequence would make the reversed bits of the exponent non-trivially incorrect. Bernstein *et al.* noted that it would be possible to obtain a correct S–M sequence from multiple measurements of cache traces with alignment and a simple majority rule, and Ueno *et al.* showed an explicit method to reconstruct the correct S–M sequence from approximately 100 measurements based on Levenshtein distance and dynamic time warping (DTW) [UTHH21]. However, these error-correction methods using multiple measurements are unavailable for attacking exponent-blinded implementation due to the random mask; we must obtain a correct S–M sequence from one measurement.

Vredandaal's and Breitner's algorithm in this study, as the difference between van Vredendaal/Breitner and Oonishi–Kunihiro has little impact on the proposed attack as discussed in Section 4.4.

2.5 Exponent blinding

Exponent blinding is one of the major countermeasures against side-channel attacks. The basic idea behind exponent blinding is to add a random value to the secret exponent such that the exponentiation result is preserved. More precisely, exponent-blinded RSA–CRT decryption is expressed by

$$m_p \equiv c^{d_p + r_p \phi(p)} \bmod p,$$
$$m_q \equiv c^{d_q + r_q \phi(q)} \bmod q,$$

where r_p and r_q are random integers. We can easily confirm $c^{d_p + r\phi(p)} \equiv c^{d_q} \bmod p$ and $c^{d_q + r\phi(q)} \equiv c^{d_q} \bmod q$ according to Euler's theorem. Even if the attacker obtains *partial* information of a blinded exponent *via* a side-channel, he/she cannot know the secret keys d_p and d_q because the exponents are masked using a random value.

Exponent-blinded RSA–CRT is known to be insecure if the attacker can obtain complete blinded exponents, because the blinded exponent allows for a correct decryption. In addition, an attacker who knows three blinded exponents can remove the mask and recover the secret key d_p and d_q in polynomial time. Let $D_{p,1}$, $D_{p,2}$, and $D_{p,3}$ be three blinded exponents with different masks $r_{p,1}$, $r_{p,2}$, and $r_{p,3}$, respectively (*i.e.*, $D_{p,1} = d_p + r_{p,1}\phi(p)$, $D_{p,2} = d_p + r_{p,2}\phi(p)$, and $D_{p,3} = d_p + r_{p,3}\phi(p)$). If $|r_{p,1} - r_{p,2}|$ and $|r_{p,1} - r_{p,3}|$ are co-prime, an attacker who completely knows $D_{p,1}$, $D_{p,2}$, and $D_{p,3}$ can recover the secret key $\phi(p)$ using the Euclidean algorithm as

$$\phi(p) = \gcd(|D_{p,1} - D_{p,2}|, |D_{p,1} - D_{p,3}|), \tag{1}$$

where $\gcd(x, y)$ denotes the greatest common divisor of x and y. This is also true for q. Fortunately, the S–M sequence of the sliding window available for cache attackers yields an exposure of only approximately 41.9% exponent bits when $w = 5$; hence, exponent-blinded RSA–CRT with a sliding window is believed to be secure.

The bit length of the mask has a large impact on the performance. For the 1,024-bit RSA–CRT, d_p and $\phi(p)$ are given by 512 bits. Therefore, if the mask bit length is b, the bit length of the blinded exponent is $512 + b$, which degrades the implementation performance. For example, if $b = 128$ as in the Libgcrypt implementation, the exponent blinding incurs an approximately 25% penalty in the execution time (excluding the cost for random number generation). The mask bit length should be set as short as possible while maintaining sufficient security against cache attacks. However, no method for evaluating the security of exponent-blinded RSA–CRT with a sliding window is known.

2.6 Partial key exposure attack

Algorithms to recover the full secret key of RSA(–CRT) from its partial information have been developed over the last approximately two decades. They have been used for the full key recovery after estimating partial bits using the side-channel attack like cache attacks, SPA, and cold-boot attacks [HSH+09].

There are three types of models for partial key bit leaks, which are referred to as random bits (*e.g.*, Heninger–Shacham-type attack [HS09]), continuous bits (*e.g.*, Coppersmith-type attack [Cop97]), and bit flips (*e.g.*, Henecka *et al.*'s attack [HMM10]). Table 2 lists the major existing key exposure attacks for RSA(–CRT). The random-bit leak model, which is the main focus of this paper, indicates that the attacker can know bits of random positions, and the remaining bits are considered as uncertain (or erasure) bits to be recovered. The partial key exposure attack for the random-bit leak was initially presented by Heninger and Shacham [HS09]. It is known that the Heninger–Shacham algorithm completes the full-key recovery in polynomial time if more than 50% bits are exposed [HS09, PPS12]. The random-bit leak model fits the scenario of a cache attack (or SPA) on RSA–CRT with

Table 2: Major partial key exposure attacks on RSA(–CRT)

Leak model	Random bits	Continuous bits	Bit flips
No exponent blinding	Heninger–Shacham [HS09] Paterson *et al.* [PPS12] Kunihiro *et al.* [KSI13]* Oonishi–Kunihiro [OK20]**	Coppersmith [Cop97] Boneh *et al.* [BDF98] Blömer–May [BM03] Coron [Cor04, Cor07] Herrmann–May [HM08] Sarkar–Maitra [SM09] Ernst *et al.* [EJMdW05] Aono [Aon09] Takayasu–Kunihiro [TK14]	Henecka *et al.* [HMM10] Paterson *et al.* [PPS12] Kunihiro *et al.* [KSI13]* Kunihiro [Kun15] Oonishi–Kunihiro [OK19]†
Exponent blinding	Fouque *et al.* [FKJM+06]‡ **This study**	Cimato *et al.* [CMS15] Zhou *et al.* [ZvdPYS22]	Schindler–Itoh [SI11] Bauer [Bau12]† Schindler–Wiemers [SW14, SW17]

* For hybrid model of random bits leak with bit flips.
** For S–M sequence of sliding window with some flips (*i.e.*, errors) in S and M.
† For S–M sequence with some flips (*i.e.*, errors) in S and M, but not (necessarily) for sliding window.
‡ On RSA without CRT.

a sliding window, as Bernstein *et al.* adopted the Heninger–Shacham algorithm in their attack [BBG+17]. In the Heninger–Shacham attack, all exposed bits are supposed to be correct. Kunihiro *et al.* proposed an extended attack that can tolerate some errors (*i.e.*, bit flips), which are likely included in the exposed bits obtained *via* SPA [KSI13]. Moreover, Oonishi and Kunihiro presented a partial key exposure attack on non-exponent-blinded RSA–CRT with sliding window [OK20] which can tolerate some flips of S and M in the estimated S–M sequence. However, its tolerable error rate is severe for $w = 4$ (0.8%), and is not evaluated for $w = 5$.

There are a few partial key exposure attacks on exponent-blinded RSA(–CRT). Fouque *et al.* presented a power analysis attack on exponent-blinded RSA using a sliding window without CRT [FKJM+06]. Their attack exploits an approximation of $d \approx \left\lfloor \frac{1+kN}{e} \right\rfloor$ to estimate the random mask and secret key, and it is unknown how to extend it to RSA–CRT (such an approximation using e, k, and N is unavailable for d_p and d_q). Bauer's attack [Bau12], which followed Fouque *et al.*'s attack, can tolerate errors probably included in partial key bits (more precisely, S–M sequence estimated *via* power traces); still, it cannot be applied to RSA–CRT. In addition, although Bauer mentioned that his attack can be extended to fixed and sliding window exponentiations, no concrete extension method is known and its success rate and feasibility are unclear. Cimato *et al.* extended the Coppersmith-type attack to the exponent blinding [CMS15], and Schindler and Wiemers presented a continued fraction attack for the exponent blinded RSA–CRT that can tolerate some errors in the bit-flip leak. Recently, Zhou *et al.* presented yet another Coppersmith-type attack on exponent-blinded RSA–CRT, which is sufficiently feasible and is adoptable of SCAs [ZvdPYS22]. Thus, no existing partial key exposure attack is adoptable for a cache attack or SPA on exponent-blinded RSA–CRT with the sliding window exponentiation.

Hereafter, we introduce the Heninger–Shacham branch-and-prune attack [HS09] and then introduce the Schindler–Wiemers continued fraction attack [SW17], which are two bases of the proposed attack.

Heninger–Shacham branch-and-prune attack. The Heninger–Shacham attack exploits the following relation between public and secret keys of RSA–CRT:

$$\begin{aligned} pq &= N, \\ ed_p &= k_p(p-1)+1, \qquad (2) \\ ed_q &= k_q(q-1)+1, \qquad (3) \end{aligned}$$

where k_p and k_q are an integer[5]. The integers k_p and k_q are unknown for the attacker in general. However, as in previous studies, we consider k_p and k_q to be known, because (k_p, k_q) takes only at most 2^{16} patterns for the standard encryption key (*i.e.*, $e = 2^{16} + 1$), which is sufficiently small for the attacker to perform an exhaustive search.

Let $x[i]$ denote the i-th bit of x in the binary integer representation. Let $\tau(x)$ be the number of trailing zeros of x (*i.e.*, $\arg\max_{s\in\mathbb{Z}} \gcd(2^s, x)$). Let Slice$[i]$ be a tuple of secret key candidates at a bit position related to i as

$$\text{Slice}[i] = (p[i], q[i], d_p[i + \tau(k_p)], d_q[i + \tau(k_q)]),$$

and Slice$[1] = (1, 1, d_p[1 + \tau(k_p)], d_q[1 + \tau(k_q)])$ can be determined only from public information, including (k_p, k_q). The Heninger–Shacham attack reconstructs the secret key in an iterative manner from Slice$[1]$ to Slice$[l]$ (*i.e.*, derives candidates for Slice$[i]$ from Slice$[i-1]$ in an ascending manner). Although one slice Slice$[i]$ contains four binary variables, it takes only two pattern for each slice regarding the RSA–CRT constraint relation. According to Hensel's lifting lemma, the above equations are translated to the following constraint relations:

$$\begin{aligned}
p[i] + q[i] &\equiv \left(N - p^{(i)}q^{(i)}\right)[i] \bmod 2,\\
d_p[i + \tau(k_p)] + p[i] &\equiv \left(k_p(p^{(i)} - 1) + 1 - ed_p^{(i+\tau(k_p))}\right)[i + \tau(k_p)] \bmod 2,\\
d_q[i + \tau(k_q)] + q[i] &\equiv \left(k_q(q^{(i)} - 1) + 1 - ed_q^{(i+\tau(k_q))}\right)[i + \tau(k_q)] \bmod 2,
\end{aligned}$$

for any i, where $p^{(i)}$ denotes up-to the $(i-1)$-th bit value of p, (*i.e.*, $p^{(i)} = \sum_{a=0}^{i-1} 2^a p[a]$), and this is the same for other variables $q^{(i)}$, $d_p^{(i+\tau(k_p))}$, and $d_q^{(i+\tau(k_q))}$. Because the attacker already have candidates for up-to the $(i-1)$-th slices when estimating the i-th slice, the right-hand side of these equations are known to the attacker. Therefore, for one sequence of Slice$[1]$, Slice$[2], \ldots,$ Slice$[i-1]$, the simultaneous equations have two solutions for Slice$[i]$. Then, if the solution does not match the exposed bits, the slice candidate is discarded as it should be an incorrect estimation. Thus, the attacker constructs a branch-and-prune tree of slice nodes corresponding to the key candidates in accordance with the constraint relations, and prunes slice nodes with incorrect key candidates inconsistent with exposed bits. The time and memory complexities heavily depend on the ratio of exposed bits. It is known that the Heninger–Shacham algorithm runs in polynomial time if the exposed bits ratio is greater than 0.5.

Schindler–Wiemers continued fraction attack. Schindler and Wiemers presented a continued fraction attack on exponent-blinded RSA–CRT with a bit-flip leak. The continued fraction attack is based on Equation (1) and exploits the following three properties of blinded exponent in RSA–CRT to tolerate the bit flips: (a) earlier computation steps of the gcd depend on only the upper bits of $|D_{p,1} - D_{p,2}|$ and $|D_{p,1} - D_{p,3}|$, (b) $\lambda_1(D_{p,1} - D_{p,2}) + \lambda_2(D_{p,1} - D_{p,3}) = 0$ holds, where $\lambda_1 = \pm(r_{p,1} - r_{p,3})$ and $\lambda_2 = \pm(r_{p,1} - r_{p,2})$, according to extended Euclidean algorithm, and (c) $\phi(p) = \left|\frac{D_{p,1}-D_{p,3}}{\lambda_1}\right| = \left|\frac{D_{p,1}-D_{p,2}}{\lambda_2}\right|$ holds. These properties allow the attacker to feasibly approximate the upper bits of p from the estimated blinded exponents with bit flips, which can be also applied to the case of q.

The upper bits of p can be approximated using a continued fraction. Let x and y be

[5]Originally, another equation $ed = k(N - p - q + 1) + 1$ (where k is an integer related to k_p and k_q) is included in the equations; but, in this study, we omit it and its corresponding constraint relation, because we can perform the branch-and-prune without them.

integers. The continued fraction of a rational number x/y is expressed by

$$\frac{x}{y} = x_0 + \cfrac{1}{x_1 + \cfrac{1}{x_2 + \cfrac{1}{\ddots\, x_{\rho-1} + \cfrac{1}{x_\rho}}}},$$

where the expansion is completed with ρ times. Here, $x_0, x_1, \ldots, x_\rho$ (and the corresponding $y_0, y_1, \ldots, y_{\rho-1}$) are integers determined in accordance with extended Euclidean algorithm such that

$$\begin{aligned} x &= x_0 y + y_0, \\ y &= x_1 y_0 + y_1, \\ y_0 &= x_2 y_1 + y_2, \\ &\vdots \\ y_{\rho-3} &= x_{\rho-1} y_{\rho-2} + y_{\rho-1}, \\ y_{\rho-2} &= x_\rho y_{\rho-1}. \end{aligned}$$

Here, x/y can be approximated as x'/y' by terminating the continued fraction expansion at the θ-th step ($\theta < \rho$). Schindler and Wiemers showed that $\phi(p)$ (and p) can be feasibly approximated using the continued fraction expansion with $x = |D_{p,1} - D_{p,2}|$ and $y = |D_{p,1} - D_{p,3}|$. It is sufficient to terminate the continued fraction expansion when $x_\theta \leq 2^b$ or $y_\theta \leq 2^b$, where b is the mask bit length, as it holds $|r_{p,1} - r_{p,2}| < 2^b$ and $|r_{p,1} - r_{p,3}| < 2^b$. Thus, the result of the continued fraction expansion is given by a pair of b-bit integers (x', y') such that $\frac{x'}{y'} \approx \left|\frac{D_{p,1}-D_{p,3}}{D_{p,1}-D_{p,2}}\right|$ (according to Property (b)). Then, Property (c) is used to estimate p (or q) from the results of the continued fraction as $\left\lfloor \left| \frac{\tilde{D}'_{p,\mu_1} - \tilde{D}'_{p,\mu_3}}{x'} \right| \right\rfloor$ or $\left\lfloor \left| \frac{\tilde{D}'_{p,\mu_1} - \tilde{D}'_{p,\mu_2}}{y'} \right| \right\rfloor$ (we use x' or y', whichever is greater). In addition, owing to Property (a), the continued fraction (according to extended Euclidean algorithm) can be calculated with some θ if the lower bits of the exponent contain some errors. Therefore, the continued fraction attack successfully works using an exhaustive guesses of bit flips in the upper b bits of exponent (not full bits), even if the estimated blinded exponent contains some noise. Yet, its extension to random-bit leak of this attack has not been discovered.

3 Proposed Attack

3.1 Overview

The proposed attack consists of four steps: (i) S–M sequence estimation *via* a side-channel, (ii) reversing partial bits of the exponent using van Vredendaal's and Breitner's algorithm [vV18, Bre17], (iii) estimation of the random mask using a modified Schindler–Wiemers continued fraction attack, and (iv) full-key recovery using an extended Heninger–Shacham partial key exposure attack.

3.2 Step (i): Estimation of S–M sequence using Flush+Reload

In this study, we employ a cache attack, namely Flush+Reload, to estimate the S–M sequence, although we can also employ SPA. In attacking exponent-blinded RSA–CRT, we

must estimate a correct S–M sequence from one measurement of the cache trace, because error-correction techniques using multiple measurements for an exponent (*e.g.*, majority vote or [UTHH21]) are unavailable due to the random mask. Since existing Flush+Reload attacks on RSA–CRT always incur error in the S–M sequences estimated from one trace, it is mandatory for an attack on exponent-blinded RSA–CRT to improve the accuracy of the Flush+Reload trace measurement.

Flush+Reload trace measurement. To acquire the S–M sequence during sliding window exponentiation, we set Flush+Reload probes at four positions: (A) integer multiplication[6] in Lines 15 and 17 of Algorithm 1, (B) modular reduction in Lines 15 and 17, (C) determination of temporal window position corresponding to Lines 9 and 10, and (D) determination of the temporal window size corresponding to $k - t$ at Line 14. The Libgcrypt implementation performs modular multiplication using a multi-precision integer multiplication followed by a modular reduction. To detect the timing of multiplication and squaring, we set probes on code segments for (A) integer multiplication and (B) modular reduction. We guess that the victim performs either squaring or multiplication during the time slot if we find either of following two patterns: (1) Probe (A) detects the victim's loading and Probe (B) also detects it less-than two slots after the detection of Probe (A) or (2) either Probe (A) or (B) detect it and another modular squaring/multiplication is not detected within the nearest two slots. Note that, if Probes (A) and/or (B) detect the loading two times within four successive time slots, we consider it as one detection, as either is likely to detect it due to speculative execution.

The attacker cannot distinguish squaring and multiplication from Probes (A) and (B), as the Libgcrypt implementation employs identical code segments for both squaring and multiplication to mitigate cache attacks [YF14]. Therefore, we should know the timing of the entry of each loop in the sliding window in distinguish squaring and multiplication. For this purpose, we set two probes (C) and (D) on code segments to determine the temporal window, which is executed at the entry of each loop of the sliding window, only after modular multiplication (not squaring). We guess that a new loop has started if we find either of the following two patterns: (1) Probe (C) detects the victim's loading and Probe (D) also detects the same less-than two slots after the detection of Probe (C) or (2) Probe (D) detects the victim's loading but Probe (C) does not, and another loop entry is not detected within the nearest two slots. This is similar to the case of Probes (A) and (B), except when only Probe (C) detects the loading but Probe (D) does not. This is because we experimentally found that this is the best in accordance with our error-correction strategy described below, where we use an S–M sequence estimation method that employs the information from Probes (C) and (D). Thus, we use two probes for detecting each procedure (four in total) to reliably estimate the S–M sequence with reduced error probability. Such probe doubling is especially effective to prevent a misdetection, as most capture errors in Flush+Reload trace are from misdetection [UTHH21].

To evaluate and validate the S–M sequence estimation, we performed an experimental cross-core Flush+Reload attack on Libgcrypt RSA–CRT running on an Intel i5-3470 CPU with 6 GB memory. The operating system was CentOS7, and the target software was from Libgcrypt 1.7.8 [Lib17][7]. We used an open-source toolkit for microarchitectural attack provided by Yarom, namely Mastik [Yar18], and we employed the performance degradation

[6] We use the terms "integer multiplication" and "modular reduction" for the code segments used for modular multiplication and modular squaring in the sliding window. Modular multiplication and modular squaring are performed using the same codes in Libgcrypt.

[7] This is old version for 2022. Let us use this version for a validity evaluation of the countermeasure which was implemented after the publication of Bernstein *et al.*'s attack targeting Libgcrypt 1.7.6, and this also allows us to compare our attack to Bernstein *et al.*'s attack in a relatively fair manner. However, this version of RSA–CRT software is almost same to the up-to-date version. Note that the purpose of this paper is not to state the existence of vulnerabilities in current implementations, but to evaluate the security of sliding-window exponentiation in RSA–CRT.

Table 3: Parameters for Flush+Reload attack using Mastik

Observed operations and Flush+Reload probe position	
Probed operation	Probe position
(A) Multi-precision integer multiplication	`_gcry_mpih_mul+21`
(B) Multi-precision modular reduction	`_gcry_mpih_mod+23`
(C) Temporal window position determination	`_gcry_mpi_powm+1743`
(D) Temporal window size determination	`_gcry_mpi_powm+1868`
Positions of performance degradation	
Compromised operation	Applied position
Multi-precision modular reduction	`_gcry_mpih_submul_1+24`
	`_gcry_mpih_submul_1+47`
	`_gcry_mpih_submul_1+60`
Multiplicand selection	`_gcry_mpi_set_cond+32`
	`_gcry_mpi_set_cond+46`
	`_gcry_mpi_set_cond+60`
Parameter settings for `FR-trace`*	
Parameter	Value
`slot`	10,000
`maxsample`	100,000
`threshold`	100
`idle`	500
`pdacount`	2

* Program for Flush+Reload in Mastik.

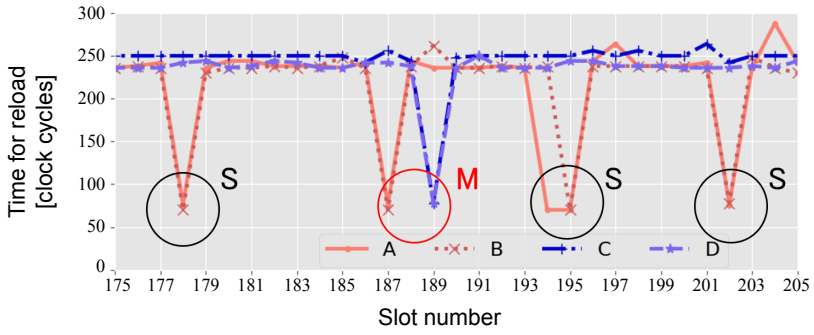

Figure 1: Example Flush+Reload trace.

attack for improved accuracy, as done in the many related studies [ABF+16]. Table 3 summarizes the parameters for the Flush+Reload attack.

Figure 1 shows a part of an example trace obtained *via* the proposed Flush+Reload attack, where the horizontal axis denotes the Flush+Reload time slot number, and the vertical axis denotes the loading time at Reload for each probe ((A)–(D)). As the loading from the main memory and LLC requires 250 and 100 cycles on average, respectively, we guessed that the victim accessed the probed code segment if Reload took less-than 195 cycles. At the 188th, 187th, 194th, 195th, and 202nd slots, Probes (A) and/or (B) detected the victim's loading of modular multiplication/squaring. Here, as the 194th slot detection may be an error due to the speculative execution, we ignore the 194th slot detection, as mentioned above. Moreover, at the 189th slot, Probes (C) and (D) detected the victim's temporal window determination (*i.e.*, the entry of a loop), which is performed only after modular multiplication (not squaring). Thus, we could estimate the S–M sequence performed by the victim as SSMS from this Flush+Reload trace.

Error-correction strategy by exploiting characteristics of sliding-window S–M sequence. We further introduce some heuristics for translating the cache traces to reliable S–M

sequence (*i.e.*, correcting errors in an estimated S–M sequence), according to the features of the sliding window exponentiation. Roughly speaking, our translation strategy is detecting modular multiplication and squaring such that the squaring is likely to be detected over multiplication (this is represented by the asymmetry between the detections of Probes (A) and (B) and Probes (C) and (D) as mentioned above), correcting inconsistency with the rule of temporal window determination, and replacing S to M in the order of likelihood such that the number of S is equivalent to the expected number. This is due to the fact that the number of squarings is actually greater than that of multiplications.

First, we must detect the timing of the entry of the first loop after the precomputation. The number of multiplications in the precomputation is fixed for the maximum window size w and is known to the attacker. As the misdetection of the victim's execution of modular multiplication frequently occurs, the attacker cannot always detect all modular multiplications. Therefore, we detect the timing of the first loop start if Probe (C) and/or (D) detects the entry of a loop after detection of more-than $0.8 \times (2^{w-1} - 1)$ successive modular multiplications (12 for $w = 5$).

Then, after translating the acquired Flush+Reload trace to the S–M sequence in the aforementioned manner, we replace an S at the tail with an M, as the least significant bit of the exponent is always one (*i.e.*, the exponents D_p and D_q are always odd). In addition, in the sliding window exponentiation, two consecutive modular multiplications are never performed (except in precomputation). Therefore, if we detect two consecutive M's, we replace the latter M with S.

We further correct errors according to the number of squarings in the exponentiation. Although the number of multiplications is variable, the number of squarings is fixed and equal to the bit length of the exponent ($512 + b$ for exponent-blinded 1,024-bit RSA–CRT, where b is the mask bit length). In the proposed method, we count the number of S in the S–M sequence derived through the above procedure. If the number of S's is greater than expected, some M's may have been misrecognized as S's due to the misdetection(s) of the entry of the loop (*i.e.*, Probes (C) and (D)). Therefore, the proposed method corrects the error by repeating the following procedures until the number of S's is equal to the expected number (this is the aforementioned error-correction method): We search for the time slots where Probe (C) does but Probe (D) does not detect the loop entry one or two slot(s) after an S; we count the number of S's between two M's around the detection of Probe (C); and, if the counted number is greater than two, we replace the S detected at the time slot with Probe (C) to an M.

Experimental evaluation. We generated 100 random RSA–CRT secret keys, and performed an experimental Flush+Reload attack 1,000 times for each key. The experimental conditions were the same as the above. Libgcrypt 1.7.8 RSA–CRT utilizes a 128-bit mask, and the mask was randomly generated for each trial. We then translated the acquired Flush+Reload traces into an S–M sequence using the proposed method. Figure 2 shows a histogram of the number of errors in the estimated S–M sequence. The number of errors was calculated as the Levenshtein distance between the true and estimated S–M sequences, as proposed in [UTHH21]. For comparison, Figure 2 also shows the result of Bernstein *et al.*'s Flush+Reload attack on Libgcrypt 1.7.6 in [BBG+17][8]. In the experiment, we could obtain a completely correct S–M sequence 10% of the time, whereas the conventional method never obtained correct one. Thus, we could confirm the improvement and effectiveness of the proposed method, which enables us to use a correct S–M sequence essential for the following steps, even with the exponent blinded.

In the following of this section, we assume that the S–M sequence estimated using

[8]We derived the values for Bernstein *et al.* by visually reading the histogram of Fig. 7 in their literature [BBG+17]. Note that Libgcrypt 1.7.6 does not employ the exponent blinding; therefore, the exponent is given by 512 bits, shorter than that in Libgcrypt 1.7.8.

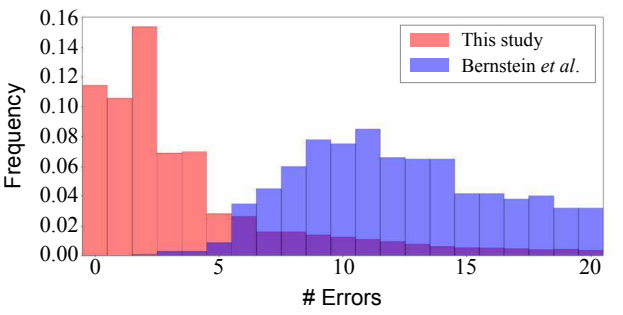

Figure 2: Histogram of number of errors in estimated S–M sequence.

Flush+Reload is completely correct as we can obtain the correct S–M sequence due to the improved Flush+Reload attack. See Section 4.2.3 and Section 4.4.2 for discussion on the impact of using real traces to the success rate.

3.3 Step (ii): Reversing partial bits of blinded exponents

In this step, we reverse the S–M sequence to the partial bits of the exponent using van Vredendaal's and Breitner's algorithm. At this step, we can obtain approximately 41.9% bits of the exponent on average for 1,024-bit RSA–CRT using the sliding window with $w = 5$.[9] The result of this step can be treated as random-bit leak with exponent blinding as the S–M sequences are (supposed to be) correct. See [vV18, Bre17] or Section 2.4 for the reversing algorithm.

3.4 Step (iii): Estimation of random masks using modified Schindler–Wiemers continued fraction attack

As described in Section 2.6, the Schindler–Wiemers continued fraction attack was originally presented for RSA–CRT key recovery from the bit-flip leak of blinded exponents. We modify and employ it to estimate the random masks used for the exponent blinding in addition to the upper bits of p, q, D_p, and D_q from the random-bit leak.

Algorithm 2 shows the proposed method to estimate the random mask and upper bits of p using the continued fraction attack. Given ν blinded exponents with some uncertain bits (*i.e.*, random-bit leak) obtained by Steps (i) and (ii), Algorithm 2 returns an estimated random mask $r_{p,\mu}$ used for blinding $D_{p,\mu}$ and the upper $2b-2$ bits of p as the best approximation. Because this attack utilizes an approximation, the estimation results are not always correct. Therefore, we use ν blinded exponents and choose the most likely result. The success rate is evaluated in Section 4.2. Here, we describe the estimation of p and its mask, but Algorithm 2 can be also used for the estimation of q and its mask. The proposed method combines the Schindler–Wiemers continued fraction approximation with an exhaustive guess of uncertain bits in the most significant bits. At Line 6, given a triple of blinded exponents with uncertain bits $(D'_{p,\mu_1}, D'_{p,\mu_2}, D'_{p,\mu_3})$, we generate a set $\mathcal{G}_{\mu_1,\mu_2,\mu_3}$ that contains candidates for $(D'_{p,\mu_1}, D'_{p,\mu_2}, D'_{p,\mu_3})$ with an exhaustive guess of uncertain bits in the upper $2b-2$ bits, whereas all the remaining lower uncertain bits are substituted with zero. This is because the lower bits have less impact on the approximation by the continued fraction, and ignoring the uncertain lower bits yields a reduced computational complexity. Note that the use of upper $2b-2$ bits is necessary for a good estimation of

[9] In [BBG+17], the libgcrypt RSA–CRT implementation used $w = 4$. The difference is because of the exponent blinding. The Libgcrypt implementation uses a window size of $w = 5$ for a 630-bit exponent (*i.e.*, 512-bit RSA–CRT secret key blinded with 128-bit mask, whereas it uses $w = 4$ for a 512-bit exponent.

Algorithm 2 Estimation of upper $2b-2$ bits of p

Input: $D'_{p,1}, D'_{p,2}, \ldots, D'_{p,\mu}, \ldots, D'_{p,\nu}$ (ν blinded exponents with random-bit leaks)
Output: $\tilde{p}$ (Estimation of p, upper $2b-2$ bits of which is likely correct)
1: **parameter** ν; ▷ Number of estimated blind exponents with random-bit leak
2: **parameter** b; ▷ Mask bit length
3: **Function** ModifiedContinuedFractionAttack$_{\nu,b}(D'_{p,1}, D'_{p,2}, \ldots, D'_{p,\nu})$
4: **set** $\mathcal{S} \leftarrow \{\}$;
5: **for each** $(\mu_1, \mu_2, \mu_3) \in \mathbb{Z}^3$ such that $1 \le \mu_1 < \mu_2 < \mu_3 \le \nu$ **do**
6: **set** $\mathcal{G}_{\mu_1,\mu_2,\mu_3} \leftarrow$ GuessingUpperUncertainBits$_{2b-2}(D'_{p,\mu_1}, D'_{p,\mu_2}, D'_{p,\mu_3})$;
7: **for each** $(\tilde{D}_{p,\mu_1}, \tilde{D}_{p,\mu_2}, \tilde{D}_{p,\mu_3}) \in \mathcal{G}_{\mu_1,\mu_2,\mu_3}$ **do** ▷ Continued fraction expansion
8: Expand $\left|\frac{\tilde{D}_{p,\mu_1}-\tilde{D}_{p,\mu_3}}{\tilde{D}_{p,\mu_1}-\tilde{D}_{p,\mu_2}}\right|$ to an approximate continued fraction as $\frac{x'}{y'}$ while $x' \le 2^{b-1}$ and $y' \le 2^{b-1}$;
9: **if** $x' \le y'$ **then**
10: **int** $p' \leftarrow \left\lfloor \left| \frac{\tilde{D}_{p,\mu_1}-\tilde{D}_{p,\mu_3}}{x'} \right| \right\rfloor$;
11: **else**
12: **int** $p' \leftarrow \left\lfloor \left| \frac{\tilde{D}_{p,\mu_1}-\tilde{D}_{p,\mu_2}}{y'} \right| \right\rfloor$;
13: **end if**
14: **if** $2^{511} < p' < 2^{512}$ **then**
15: $\mathcal{S} \leftarrow \mathcal{S} \cup \{p'\}$;
16: **end if**
17: **end for**
18: **end for**
19: **int** $\tilde{p} \leftarrow \arg\max_{p' \in \mathcal{S}} \mathrm{Val}_{\mathcal{S}}(p')$;
20: **return** upper $2b-2$ bits of $\tilde{p}$;
21: **end Function**

the b-bit random mask. Then, at Lines 8–13, we approximate p as p' using the continued fraction expansion for all candidates in $\mathcal{G}_{\mu_1,\mu_2,\mu_3}$. Because p should be in the range of $(2^{511}, 2^{512})$ for 1,024-bit RSA–CRT, we consider the continued fraction result as a good approximation if $2^{511} < p' < 2^{512}$, and preserve it in a set of candidates $\mathcal{S}$.

After examining the continued fraction approximation for all candidates in $\mathcal{G}_{\mu_1,\mu_2,\mu_3}$ for any (μ_1, μ_2, μ_3), we determine the most likely approximation according to the valuation function $\mathrm{Val}_{\mathcal{S}}$ at Line 19. Here, we utilize the valuation function derived by Schindler and Wiemers. We first sort the candidates in increasing order. Let p'_h be the h-th candidate such that $p'_1 < p'_2 < \cdots < p'_h < \cdots < p'_{|\mathcal{S}|}$. The candidates would be dense around the correct p, whereas they would be sparse if they are far from p. The valuation function for p'_h is calculated using its eight neighborhoods[10] and is defined as

$$\mathrm{Val}_{\mathcal{S}}(p_h) \coloneqq \sum_{\omega \in \{\pm 1, \pm 2, \pm 3, \pm 4\}} f\left(\left|\frac{p'_h - p'_{h+\omega}}{2^{512-2b}}\right|\right), \tag{4}$$

where f denotes a probability density function for a random variable representing the fraction in the attack (see [SW17, Equation (44)]) and is given by

$$f(\alpha) = \begin{cases} \frac{11}{12} - \frac{31\alpha}{60} & \text{if } \alpha < 1, \\ -\frac{1}{12} + \frac{\alpha}{60} + \frac{2}{3\alpha^2} - \frac{1}{5\alpha^4} & \text{if } 1 \le \alpha < 2, \\ \frac{4}{3\alpha^3} - \frac{1}{\alpha^4} & \text{if } 2 \le \alpha. \end{cases}$$

Thus, at Line 19, we determine the best approximation as the candidate with the maximum valuation, the upper $2b-2$ bits of which are supposed to be equivalent to p. However, in practice, we cannot always obtain the best approximation with the maximum valuation in Algorithm 2, which may be due to the influence of lower bits on the approximation.

[10] The value of eight was determined according to our experiment.

Therefore, we may have several candidates for p with a relatively large valuation, as in the experiment in Section 4.2.

Hereafter, we estimate the upper $2b-2$ bits of $D_{p,\mu}$ and its random mask from the estimated $\tilde{p}$. We choose one $\tilde{D}_{p,\mu}$ such that the exposed bits ratio exceeds 50% if we completely estimate its upper $2b-2$ bits (this is necessary for the feasible computation of Step (iv)). Then, we also choose two blinded exponents $\tilde{D}_{p,\mu_1}$ and $\tilde{D}_{p,\mu_2}$ such that the number of exposed bits in the upper $2b-2$ bits is maximized. Then, we perform a continued fraction attack using the above three blinded exponents with an exhaustive guess of uncertain bits as well as Lines 6–16 in Algorithm 2, and we obtain a set of candidates $\mathcal{S}$. We pick one p' from $\mathcal{S}$ such that the upper $2b-2$ bits of p' match those of $\tilde{p}$ as much as possible. Here, we consider $\tilde{D}_{p,\mu}$ corresponding to p' as the estimation of $D_{p,\mu}$ (*i.e.*, the upper $2b-2$ bits of $\tilde{D}_{p,\mu}$ are equivalent to $D_{p,\mu}$). The random mask $r_{p,\mu}$ is estimated by

$$r_{p,\mu} = \left\lfloor \frac{\tilde{D}_{p,\mu}}{\tilde{p}} \right\rfloor,$$

according to the approximate equation of [SW17, Equation (54)]. For a reliable random mask estimation, we can perform this procedure using several pairs of $\tilde{D}_{p,\mu_1}$ and $\tilde{D}_{p,\mu_2}$ for different μ_1 and μ_2 that have many exposed bits in the upper $2b-2$ bits.

3.5 Step (iv): Recovery of exponent using extended Heninger–Shacham partial key exposure attack

In this step, we extend the Heninger–Shacham key exposure attack such that we can derive the RSA–CRT secret key using public information, $r_{p,\mu}$, $r_{q,\mu'}$, $\tilde{p}$, $\tilde{q}$, $\tilde{D}_{p,\mu}$, and $\tilde{D}_{p,\mu'}$ estimated in Step (iii) using a branch-and-prune method (note that it is not necessary for $\mu \neq \mu'$ to hold).

Recall that the blinded exponents in RSA–CRT are given by $D_p = d_p + r_p(p-1)$ and $D_q = d_q + r_q(q-1)$. Substituting $d_p = D_p - r_p(p-1)$ and $d_q = D_q - r_q(q-1)$ into Eqs. (2) and (3), respectively, we can derive the exponent-blinded version of the Heninger–Shacham constraint equations as

$$\begin{aligned} pq &= N, \\ eD_p &= (er_p + k_p)(p-1) + 1, \\ eD_q &= (er_q + k_q)(q-1) + 1, \end{aligned}$$

where the terms $er_p + k_p$ and $er_q + k_q$ are known to the attacker, as e is the public key, k_p and k_q are exhaustively searchable for the standard e, and r_p and r_q are estimated as $r_{p,\mu}$ and $r_{q,\mu'}$ at Step (iii), respectively. Let $\gamma_{p,\mu} = er_{p,\mu} + k_p$ and $\gamma_{q,\mu'} = er_{q,\mu'} + k_q$. As with the non-exponent-blinded version, the above equations are translated to the constraint relation for the exponent-blinded RSA–CRT as

$$\begin{aligned} p[i] + q[i] &\equiv \left(N - p^{(i)}q^{(i)}\right)[i] \bmod 2, \\ D_p[i+\tau(\gamma_{p,\mu})] + p[i] &\equiv \left(\gamma_{p,\mu}(p^{(i)}-1) + 1 - eD_p^{(i+\tau(\gamma_{p,\mu}))}\right)[i+\tau(\gamma_{p,\mu})] \bmod 2, \\ D_q[i+\tau(\gamma_{q,\mu'})] + q[i] &\equiv \left(\gamma_{q,\mu'}(q^{(i)}-1) + 1 - eD_q^{(i+\tau(\gamma_{q,\mu'}))}\right)[i+\tau(\gamma_{q,\mu'})] \bmod 2. \end{aligned}$$

Using these relations, we can derive a set of reduced key candidates using a branch-and-prune in the same manner as the Heninger–Shacham attack; that is, we construct a branch tree of slice nodes (in which $\tau(k_p)$ and $\tau(k_q)$ are replaced with $\tau(\gamma_{p,\mu})$ and $\tau(\gamma_{q,\mu'})$, respectively) and prune slice nodes that are inconsistent with the exposed bits of p, q, $\tilde{D}_{p,\mu}$, and $\tilde{D}_{q,\mu'}$. Here, their upper bits are also estimated in Step (iii) in addition to the exposure *via* side-channel. Thanks to this, the overall exposed bits ratio exceeds 50%, which allows for a sufficiently feasible branch-and-prune.

Remark 3. Strictly, the 1,024-bit RSA–CRT secret keys are given in the range of $(2^{511.5}, 2^{512})$ (which is similar for larger key sizes). This fact can be used for a further reduction of key candidates and/or computational complexity in principle. However, the proposed attack do not exploit this fact for the reduction. This is because this fact is basically related to a constraint of MSBs of secret primes, D_p, and D_q. The upper-bits of D_p and D_q estimated using Flush+Reload include less errors and are more reliable than lower-bits, as the target implementation here is the left-to-right sliding window that scans the exponent from the MSB. In addition, at Step (iii), the constraint can be used to estimate the value of erasure bits, which results in a little reduction of computational cost, as the constraint is almost solely related to some MSBs. It is a future work to develop a strategy to efficiently incorporate the constraint in the attack.

4 Evaluation

4.1 Attack complexity

The computational bottleneck of the proposed attack is the continued fraction expansion with an exhaustive guess of uncertain bits in Step (iii). The number of continued fraction expansions depends on the uncertain bits in the upper $2b-2$ bits of the blinded exponents. Let ϵ_b be the expected number of uncertain bits in the upper $2b-2$ bits of blinded exponents. For ν blinded exponents in a random-bit leak, The number of continued fraction expansions to be performed is

$$2^{3\epsilon_b}\binom{\nu}{3} \approx \frac{(2^{\epsilon_b}\nu)^3}{6},$$

because we are expected to guess $2^{3\epsilon_b}$ bits for three blinded exponents at Line 6 in Algorithm 2 and we should repeat this for 3-out-of-ν combinations of exponents. Since about 41.9% bits are exposed by van Vredendaal's and Breitner's reversing algorithm on average when $w=5$, ϵ_b is approximately given by $(1-0.419)\times(2b-2)$. For example, if $\nu=10$, the expected number of continued fraction expansion is approximately $2^{59.7}$, $2^{115.4}$, and $2^{227.0}$ for $b=16$, 32, and 64, respectively ($\nu=10$ would be sufficient for a successful attack as evaluated in Section 4.2). Note that, if we do not estimate the random mask nor reduce its candidates at Step (iii), the branch-and-prune at Step (iv) is critically infeasible.

Moreover, this complexity can be improved by selecting blinded exponents with fewer uncertain bits in the upper $2b-2$ bits from a lot of blinded exponents. We performed 20,000 exponent-blinded RSA–CRT decryptions for a secret key, and counted the numbers of uncertain bits obtained by van Vredendaal's and Breitner's reversing algorithm from its S–M sequence. Then, we selected ten blinded exponents such that the uncertain bits in the upper $2b-2$ bits are minimized, which indicates that we set $\nu=10$ for 20,000 blinded exponents in random-bit leak. We repeated this procedure for 100 RSA–CRT secret keys independently and randomly generated. Table 4 lists the averaged numbers of uncertain bits and the resulting complexities for $b=16$, 32, and 64, where complexity indicates the number of required continued fraction expansions. From Table 4, we confirm that the proposed attack can be carried out with significantly less complexity by selecting good 10-out-of-20,000 blinded exponents compared to the above straightforward manner (*i.e.*, by just using ten exponents). However, the attack is still infeasible even on a 64-bit mask, which requires far greater complexity than breaking 1,024-bit RSA. Moreover, an attack on a 32-bit mask would also be infeasible as Schindler and Wiemers mentioned that the feasible number of continued fraction expansions would be at most 2^{60} [SW17]. Thus, we confirm that a 32-bit mask would be sufficient to defeat the proposed attack.

Table 4: Complexity of proposed attack using 10-out-of-20,000 cache traces

Mask bit length b	Averaged number of uncertain bits in the upper $2b-2$ bits	Complexity
16	6.08	$2^{25.6}$
32	20.1	$2^{67.7}$
64	51.3	2^{161}

4.2 Experimental validation and success rate evaluation

4.2.1 1,024-bit RSA–CRT

We then experimentally validated the proposed attack. We here demonstrate the experimental attack using a 16-bit blinding mask (*i.e.*, $b = 16$) for the feasibility. Here, we particularly evaluate the success rate of Step (iii) in addition to its computational time, assuming that the attacker obtains complete S–M sequences in Step (i).

We performed Z exponent-blinded 1,024-bit RSA–CRT decryptions for a secret key, obtained the exposed bits of the blinded exponent from its S–M sequence according to van Vredendaal's and Breitner's reversing algorithm (*i.e.*, Step (ii)), and then execute Step (iii). We considered Step (iii) to be succeeded if we correctly reconstructed $r_{p,\mu}$ and $r_{q,\mu'}$ and the valuation ranks of correct $\tilde{p}$ (or $\tilde{q}$) were less than about 1,500.[11] We evaluated the success rate for 100 secret keys independently and randomly generated. As a result, when $\nu = 10$, we confirm that the success rates were 36/100, 72/100, and 81/100 for $Z = 10{,}000$, 20,000, and 30,000, respectively, which implies the feasibility of the proposed attack with a meaningful success rate.

Finally, we actually executed the computation of Steps (ii)–(iv) given the correct S–M sequences. We used an Intel Xeon Gold6144 with a 384 GB memory. The execution times for Steps (ii), (iii), and (iv) were 12 m, 2 h, and 3 h, respectively[12]. Consequently, we confirm that the proposed attack (given the correct S–M sequence) can recover the secret key of 1,024-bit RSA–CRT with a sliding window using a 16-bit mask within a practical time.

4.2.2 2,048-bit RSA–CRT

We then apply the proposed attack on 2,048-bit RSA–CRT implementation with a 20-bit mask. The sliding window exponentiation for a 1,044-bit exponent (*i.e.*, a 1,024-bit secret key for RSA-CRT with a 20-bit mask) in Libgcrypt uses a window size of $w = 5$. As the window size is identical to that of 1,024-bit RSA–CRT, Steps (i), (ii), and (iv) are carried out in the same manner. The feasibility of the extended Heninger–Shacham attack is also the same because the ratio of uncertain bits are the same. The computational complexity order of Step (iii) only depends on the mask bit length (ignoring the difference in the cost of continued fraction expansion between 512 and 1,024 bits), because the complexity is determined by the number of uncertain bits in the upper 20 bits of three blinded exponents. Therefore, the proposed attack can be applicable to 2,048-bit RSA–CRT with almost the same computational complexity in principle. We experimentally evaluated the success rate (*i.e.*, the probability of the correct random mask haven a rank better than 1,500) of

[11] $\tilde{p}$ is correct if its upper $2b-2$ bits are equivalent to p. The rank of $\tilde{p}$ indicates the position of $\tilde{p}$ if we sort all candidates in decreasing order of their valuations. Note here that the rank of 1,500 does NOT mean that we require a full computation of the partial key exposure attack for 2,000,000 ($\approx 1{,}500^2$) candidates for $\tilde{p}$ and $\tilde{q}$ in Step (iv). The branch-and-prune algorithm would (empirically) terminate immediately with no solution if we use a pair of wrong key and mask candidates; hence, we can easily distinguish the wrong key candidate without an intensive computation.

[12] For the ease of computation in Step (iii), we used only the upper $3b-2$ bits of p'_h to calculate the valuation in Equation (4), because the lower bits have little influence on the result. Note also that "2 h" in Step (iii) is the execution time if we have the correct S–M sequence only for Step (ii), and "3 h" in Step (iv) is the execution time for the pair of correct key and mask candidates.

Step (iii) for 2,048-bit RSA–CRT using 200 randomly generated RSA–CRT secret keys in the same manner as Section 4.2.1 with $Z = 2^{15} = 32768$. As a result, we confirmed that the success rate of random mask estimation was about 7%, which was lower than that of 1,024-bit RSA–CRT. This would be because the ratio of MSB length utilized for the continued fraction approximation for 2,048-bit RSA–CRT is almost the half of that for 1,024-bit RSA-CRT, which makes it more difficult to perform a reliable approximation and to estimate the random mask accurately in the case of 2,048-bit RSA–CRT, even though the mask bit length is identical.

Though the success rate of 7% is not very high, the proposed attack may still be practical if the attack can obtain a large number of traces and lead to one success. With regard to Flush+Reload trace acquisition, the attacker cannot achieve a very high accuracy in the S–M sequence estimation. However, as discussed in Section 4.2.3, the attacker can utilize S–M sequences even with errors for the random mask estimation, and repeat the extended Heninger–Shacham algorithm many times until a correct sequence is obtained. The adoption of advanced partial key exposure attacks for random-bit and bit-flip hybrid leakage model would be another possible option to make the attack practical. In contrast, if we consider an SPA-like attack which estimate the S–M sequence power traces, the accuracy in estimating S–M sequence is sufficiently high. In fact, a previous study have demonstrated a complete estimation with 80–100% accuracy from only one power trace (*e.g.*, [SIUH22]), which make our attack practical.

4.2.3 On attack using real Flush+Reload traces

In a real attack, we should consider S–M sequences that may include errors, as evaluated in Section 3.2. However, the random mask estimation using the continued fraction expansion in Step (iii) is still available and valid even if the S–M sequences include errors. We conducted an experimental continued fraction expansion using Flush+Reload traces including errors, which were generated to simulate real Flush+Reload traces. As a result, we confirmed that its success rate was comparable with that in Section 4.1. This is because only a few MSBs are utilized for the continued fraction expansion, and most other lower bits have little impact on the approximation result of the continued fraction expansion. In the continued fraction expansion, we always guess uncertain bits in the lower bits (*i.e.*, bits excluding the upper $2b-2$ bits) as zero and do not utilize the lower bits; therefore, the errors in the lower bits are rather trivial for the continued fraction expansion approximation. Meanwhile, the MSBs of estimated exponents are more reliable than other lower bits because of the nature of the left-to-right scanning. Thus, we can use S–M sequences with errors acquired as real Flush+Reload traces.

In contrast, we require a correct S–M sequence for the extended Heninger–Shacham attack in Step (iv). To this end, we repeat Step (iv) until we obtain a correct S–M sequence to recover a correct key. We confirmed from our evaluation result in Section 3.2 that the value of ν was not large, and would be sufficiently practical. Here, some attacks for random-bit and bit-flip leaks have been studied [KSI13], and the extension and use of such attacks for our situation may improve the success rate of the proposed attack (in fact, our extension of Heninger–Shacham attack in Section 3.5 would be applicable to other related attacks with the similar ideas). Although the S–M sequence errors do not result in uniform bit errors assumed in conventional attacks, these attacks would be available if the number of errors is sufficiently small. Thus, the success rate of our attack is comparable with the evaluation in Section 4.2.1 and Section 4.2.2. See Section 4.4.2 for an estimation of success rate of the proposed attack using real traces.

4.3 Relation between implementation performance and mask bit length against proposed attack

Practical RSA–CRT implementations in open-source libraries commonly employ longer-than 128-bit masks. In this section, we investigate the relation between implementation performance and mask bit length against the above-mentioned attack. The mask bit length requirement considered here is against only the proposed attack using Flush+Reload[13] In this section, we refer to "modular multiplication" as both squaring and multiplication without distinction, whereas we simply say "squaring" and "multiplication" to refer to them individually.

We first estimated the performance improvement through analysis of the number of modular multiplication in the exponentiation, and then measured the computational time of RSA–CRT with different mask bit lengths in our environment. In the blinded exponentiation, the number of squarings is fixed as $l + b$, where l is the bit length of exponent (*e.g.*, $l = 512$ for each subkey d_p and d_q in the 1,024-bit RSA–CRT decryption) and b is the mask bit length[14]. The number of multiplications in the precomputation was also fixed as $2^{w-1} - 1$, where w is the maximum window size. In contrast, the number of multiplications in the main loop varies depending on the exponent, and its analytical evaluation would be difficult. Therefore, we experimentally evaluated it by generating 100 random RSA–CRT secret keys, performing 1,000 blinded exponentiations for each generated key (that is, performing 100,000 exponentiations in total), and counting the number of multiplications. Figure 3 shows a histogram of the number of multiplications in the main loop for a 512-bit exponent blinded by a mask with lengths of $b = 32$, 64, and 128. The modes were 92, 97, and 108 for $b = 32$, 64, and 128, respectively, which are approximately equal to the averages. This indicates that the overall numbers of modular multiplications in the exponent-blinded sliding window is given by 652, 688, and 764 on average for $b = 32$, 64, and 128, respectively. Thus, a sliding window with 32-bit and 64-bit mask requires approximately 15% and 10% fewer modular multiplications (*i.e.*, computational cost), respectively, than that with a 128-bit mask, and even the security against side-channel attacks would be preserved.

We also experimentally evaluated the RSA–CRT decryption execution time using 32-bit, 64-bit, and 128-bit masks. We performed 1,000 RSA–CRT decryptions for 100 RSA–CRT secret keys (100,000 decryptions in total), as above, and measured the execution time. We used an Intel i5-3470 CPU with a 6 GB memory and CentOS7 operating system (as same as the environment in Section 3.2). The RSA–CRT software was derived from Libgcrypt 1.7.8 [Lib17]. It originally uses a 128-bit mask, and we modified it for 32-bit and 64-bit mask implementations. As a result, the average execution times were 0.869 ms, 0.964 ms, and 1.05 ms for $b = 32$, 64, and 128, respectively. We confirm that the execution times are consistent with the number of modular multiplications, as discussed above; that is, the RSA–CRT software using 32-bit and 64-bit masks required approximately 15% and 10% less execution time, respectively, than that using 128-bit mask.

Comparison with fixed window exponentiation. Fixed window is the fastest constant-time exponentiation, and is employed in many open-source software libraries. A fixed window uses $2^w - 1$ multiplications in the precomputation (where w is the window size) and l squarings and $\lceil l/w \rceil$ multiplications in the main loop (where l is the bit length of the exponent). In total, the number of modular multiplications in a fixed window is $(2^w - 1) + l + \lceil l/w \rceil$ unless the exponent blinding is present. As the fixed window is a

[13]Note that all possible attacks should be considered to determine the mask length depending on the implementation and usage.

[14]Some sliding (or fixed) window implementations omit the leading squarings before the first multiplication. In this study, for the simplicity, we ignored it and supposed that the squarings are not omitted.

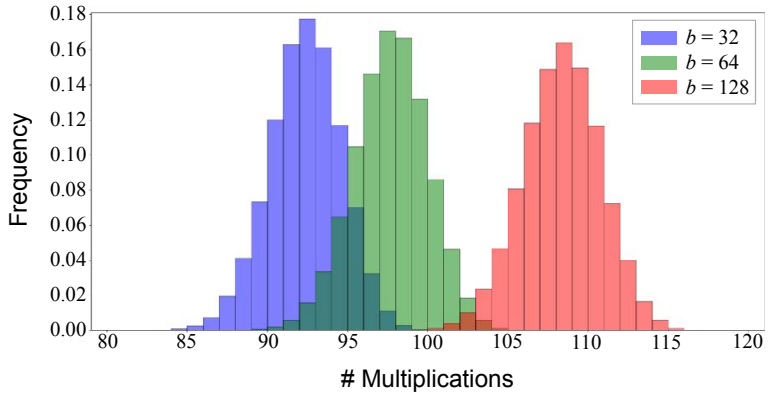

Figure 3: Histogram of number of multiplications in sliding window main loop for 512-bit blinded exponent.

deterministic algorithm that is secure against SPA-like attacks even without exponent blinding, we compared it with the exponent-blinded sliding window. When $l = 512$ and $w = 5$, the fixed window exponentiation requires 645 modular multiplications. Therefore, a fixed window without exponent blinding and the sliding window with a 32-bit mask (which requires 652 modular multiplications) would have comparable execution times[15].

Note that it should be better to adopt exponent blinding if we should consider attacks other than SPA-like ones (*e.g.*, timing attacks, collision-based/horizontal power analysis attacks, power analysis attacks using statistical tools, and Prime+Probe attacks [HMA+08, HKT12, Sch15, IGI+16, YGH18, PCBP21]). In this case, a sliding window would be superior to a fixed window for any mask bit length, as a sliding window is inherently faster than fixed window. In fact, exponent blinding would be effective in mitigating or preventing timing attacks, some sophisticated power analysis attacks, and Prime+Probe attacks.

4.4 Discussion

4.4.1 Comparison to state-of-the-art attacks

An important discussion point is how optimal our analysis is with respect to existing attacks. The strategy of the proposed attack has a common point with Fouque *et al.*'s attack [FKJM+06], namely, guessing random masks before recovering the secret key. As Fouque *et al.*'s attack and a follow-up study by Bauer [Bau12] are only attacks applicable to the exponent-blinded RSA (without CRT) with random-bit leak to the best of authors' knowledge, this strategy would be representative of the partial key exposure attack on an exponent-blinded RSA with a random bits leak in the present.

Van Vredendaal's and Breitner's recovering algorithm is optimal in the number of reversed bits, although it further improved by Oonishi *et al.* [OHK19]. It was experimentally shown that Oonishi *et al.*'s attack can reduce the entropy by 0.90 bits uncertain when $w = 5$ after van Vredendaal's attack[16]. Therefore, the improved reversing algorithm can reduce the entropy of the upper $2b$ bits in Step (iii) using 10-out-of-20,000 traces from 6.08, 20.1, and 51.3 bits (in Table 4) to 5.50, 19.2, and 46.3 bits for $b = 16$, 32, and 64, followed

[15] As the fixed window is a deterministic exponentiation, its S–M sequence leaks no information about secret key. Therefore, the fixed window may adopt a dedicated squaring algorithm, which is 1.5–2.0 times faster than a multiplication. If so, a fixed window would be superior to an exponent-blinded sliding window. However, in practice, many fixed window implementations do not employ such dedicated squaring, but instead use an identical modular multiplication code for both squaring and multiplication.

[16] The value of 0.904 bits is calculated as 534/591, because it was shown that their attack can reduce the entropy of uncertain bits from 591 to 534 when the exponent is 1,024 bits and $w = 5$.

by $2^{23.8}$, $2^{64.9}$, and 2^{146} continued fraction expansion, respectively, which indicates that the attack on a 64-bit mask is still infeasible (note that these complexities were calculated assuming that the attacker always obtained only correct S–M sequences *via* a side-channel).

Schindler–Wiemers continued fraction attack is the state-of-the-art method which can be used for estimating the random mask and the upper bits of blinded exponents. As the key-recovery capability/limitation of the (extended) Heninger–Shacham branch-and-prune attack has been analyzed [PPS12], the proposed attack would be meaningfully optimal with respect to the existing attack. Although there is a Coppersmith-type partial key exposure attack (*i.e.*, attack with continuous bit leak using LLL reduction) in the presence of exponent blinding [CMS15], the Coppersmith-type attack cannot work with a bit flips leak that suits to the sliding windows leak, and there is no known way to combine Coppersmith-type and Heninger–Shacham attacks even for non-exponent-blinded cases. Moreover, the complexity of the proposed attack was evaluated for the case that the attacker can obtain many traces as in Table 4. Thus, we believe that it is non-trivial to improve the proposed attack using existing techniques, which indicates that protecting the sliding window exponentiation using a 32-bit mask would be valid and sufficient in our attack situation, unless yet another new partial key exposure attack that significantly improves the capability of key recovery were to be found.

4.4.2 Estimation of real attack costs

In the above experiment, we employed simulated Flush+Reload traces while we evaluated them in a real setting in Section 3.2. In addition, our evaluation for Step (iv) assumed that the guessed random masks at Step (iii) and S–M sequence were completely correct (although it validated the soundness of our extended Heninger–Shacham algorithm). Thus, our experimental attack is different from real ones as follows: (1) there would be a difference between simulated and real traces, and (2) the extended Heninger–Shacham algorithm should be repeated until correct S–M sequences and guessed random masks are selected. We here discuss about the cost (*i.e.*, the number of Flush+Reload traces and time duration) for a successful real attack, considering the differences.

The difference (1) has an influence on both Steps (iii) and (iv). However, the influence on the random mask guess in Step (iii) would be negligible because the continued fraction expansion attack used in Step (iii) utilizes some upper bits of an estimated secret exponent as mentioned in Section 4.2.3. The errors included in the other lower bits are ignored due to the approximation by the continued fraction expansion. In addition, the upper bits estimated by Flush+Reload (followed by Step (ii)) are more reliable than the other lower bits due to the nature of the exponent bit scanning in the left-to-right manner. In fact, we confirmed by simulation that the random mask could be successfully guessed even using S–M sequences including errors (that imitates real errors evaluated in Section 3.2 and discussed in [UTHH21]) with a high success rate comparable to that using correct S–M sequences.

As for the difference (2), the Heninger–Shcham algorithm requires a pair of fully correct partial key information for d_p and d_q (D_p and D_q in the case of the proposed attack). This indicates that at least one correct S–M sequence for each D_p and D_q should be included in the results of Step (i). In addition, at Step (iii), we must succeed in guessing a random mask for the correct S–M sequence, although the success rate (which is defined as the probability of the correct mask having a rank better than about 1,500 in this paper) is not 100% in our experiment. Therefore, there is a tradeoff between the success rate and the number of traces/computational cost. More precisely, in order to achieve a sufficient success rate for key recovery, we must use the S–M sequences at Step (iii) as times as it is guaranteed with a meaningful probability that a correct S–M sequence is included in the inputs to Step (iii) and a random mask is guessed successfully for at least one correct S–M sequence. Note here that we can use D_p and D_q for different exponentiations, and do not

have to simultaneously succeed in the correct guesses.

Recall that the frequency of a completely correct S–M sequence acquisition at Step (i) was about 10% in our experiment. Assume that the success of random mask guess is independent of whether the S–M sequence is completely correct or not. For 1,024-bit RSA–CRT, the success rate at Step (iii) was 36%, 72%, and 82% for $Z = 10{,}000$, 20,000, and 30,000 in our experiment, respectively. This yields a success probability for secret key recovery of 3.6%, 7.2%, and 8.2% for one trial of the proposed attack. As well, for 2,048-bit RSA–CRT, the success rate at Step (iii) was 7% for $Z = 32{,}768$ in our experiment, which yields a success probability of 0.7%. Here, an attacker can improve the success probability by repeating the trial of Step (iii) for different input datasets (*i.e.*, S–M sequences), as $Z = 30{,}000$ or so would be feasible for some applications, and the attacker may use more traces. In other words, the repeated attempt improves the overall probability of success in recovering the correct secret key at the expense of computational cost.

As mentioned above, we may not always require a full computation of the extended Heninger–Shacham algorithm for wrong candidates because the Heninger–Shacham algorithm empirically terminated immediately and did not take much time for wrong candidates; Step (iv) would not be the computational bottleneck, compared to Step (iii). Thus, the proposed attack would be sufficiently feasible in the real world with a non-negligible success rate.

4.4.3 Relation between attack feasibility and window size

The computational bottleneck of the proposed attack is the number of continued fraction expansions at Step (iii) that increases by the number of uncertain bits at the upper bits of the secret exponent. The number of uncertain bits fully depends on the window size w. In this paper, we evaluated $w = 5$ which is employed exponent-blinded RSA–CRT in Libgcrypt. In contrast, if we want to reduce the memory complexity for storing the pre-computation table, we can choose a smaller window size. In [OHK19], Oonishi *et al.* showed that the expected ratios of uncertain bits after Step (ii) were 50.19% and 58.08% for $w = 4$ and $w = 5$, respectively. This indicates that the sliding window leakage for $w = 4$ contains approximately 15% less uncertain bits than that for $w = 5$, which makes the proposed attack more feasible. Assume here that the distribution of the number of uncertain bits in sliding window leakage is approximately identical for practical window sizes except for the mean. When $w = 4$, the expected numbers of uncertain bits included in the upper $2b - 2$ bits corresponding to Table 4 would be 5.17, 17.1, and 43.6 for $b = 16$, 32, and 64, respectively. These numbers correspond to the complexities of $2^{22.9}$, $2^{58.7}$, and 2^{138}, respectively. Assuming that 2^{60} continued fraction expansions are feasible as mentioned above, 1,024-bit RSA–CRT with $b = 32$ and $w = 4$ would be vulnerable to the proposed attack. Moreover, the success rate of Step (iii) would be improved for larger mask bit lengths, as discussed in Section 4.2.2. Thus, the proposed attack could be more feasible and successful for a smaller window size.

In contrast, the number of uncertain bits in the sliding window leakage increases by the window size. For example, according to Oonishi *et al.* [OHK19], its expected ratio for $w = 6$ was 63.91%, which is approximately 10% larger than that for $w = 5$. With regard to Table 4 as well, when $w = 6$, the expected numbers of uncertain bits included in the upper $2b - 2$ bits would be 6.69, 22.1, and 54.4 for $b = 16$, 32, and 64, respectively. These numbers correspond to the complexities of $2^{27.5}$, $2^{73.7}$, and 2^{170}, respectively. Although the proposed attack would be still feasible for $b = 16$ and $w = 6$, a larger window size yields more computational complexities in Step (iii) and Step (iv), and may make the success rate worse. Note that a larger window value are optimal for a larger bit RSA–CRT [Koc95] (although Libgcrypt implementation employs $w = 5$ even for 2,048-bit and 4,096-bit RSA–CRT). The above discussion indicates a further difficulty in applying the proposed attack to larger-bit RSA–CRTs.

4.4.4 Attack applicability

The proposed attack is applicable to exponent-blinded RSA–CRT with sliding window, not limited to Libgcrypt, as summarized in Table 1. Some of them provide an option to adopt the (message/exponent) blinding for a (more) secure exponentiation. For example, the mbedTLS RSA–CRT with the sliding window has an option to adopt an exponent blinding with a 28-byte mask. The proposed attack is applicable to such implementation in the manner similar to Libgcrypt. As another attack direction/context, we can obtain S–M sequences through an SPA, instead of Flush+Reload. As mentioned above, several studies have demonstrated a complete estimation with about 80% accuracy from only one power trace (*e.g.*, [SIUH22]). Therefore, if a physical side-channel is available, the attacker can correct S–M sequences more accurately than Flush+Reload, followed by Steps (ii), (iii), and (iv) of the proposed attack, which would yield a practical key recovery.

5 Conclusion

This paper shows the fist security evaluation of exponent-blinded RSA–CRT implementations with sliding window exponentiation. We presented an improved a Flush+Reload attack that accurately estimates the S–M sequence in the exponentiation, and new partial key exposure attack on RSA–CRT applicable to the sliding window leakage. Combining them, we showed the possibility of the full key recovery of 1,024-bit and 2,048 RSA–CRT sliding window implementations with a 20-bit mask. Meanwhile, we also showed that the proposed attack was not feasible against 32-bit or more masks. Accordingly, we investigated the relation between implementation performance and mask bit length against the proposed attack. Although we should consider all possible attacks to design a sufficient countermeasure, our analyses result would help to determine a proper mask length against Flush+Reload and SPA-like attacks.

The success rate of the proposed attack would be still not very high, although this paper is the first report that demonstrated the possibility of exponent-blinded CRT–RSA implementation using sliding window (even with a short mask). Our answer to the question "How secure is exponent-blinded RSA–CRT with sliding window exponentiation?" would be sufficiently secure in the current situation (at least against the proposed attack and known state-of-the-art attacks), as the current major implementations adopt a sufficient mask bit length. For further validation of our augmentation, it would be an important future work to derive a formal security bound of the key exposure attack in the presence of exponent blinding.

Acknowledgment

We owe our deepest gratitude to Mr. Soki Osawa for his valuable cooperation. Dr. Sylvain Guilley gave us insightful comments and suggestions. We want to thank Dr. Diego F. Aranha for the shepherding care. This study has been supported by JST CREST (Grant No. JPMJCR19K5).

References

[ABF+16] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop van de Pol, and Yuval Yarom. Amplifying side channels through performance degradation. In *Annual Computer Security Applications Conference*, pages 422–435, 2016.

[Aon09] Yoshinori Aono. A new lattice construction for partial key exposure attack for RSA. In *Public Key Cryptography—PKC 2000*, volume 5443 of *Lecture Notes in Computer Science*, pages 34–53, 2009.

[Bau12] Sven Bauer. Attacking exponent blinding in RSA without CRT. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 7275 of *Lecture Notes in Computer Science*, pages 82–88, 2012.

[BBG+17] 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.

[BDF98] Dan Boneh, Glenn Durfee, and Yair Frankel. An attack on RSA given a small fraction of the private key bits. In *Advances in Cryptology—ASIACRYPT 1997*, volume 1514 of *Lecture Notes in Computer Seience*, pages 25–34, 1998.

[BJ13] Aurélie Bauer and Éliane Jaulmes. Correlation analysis against protected SFM implementations of RSA. In *Progress in Cryptology—INDOCRYPT 2013*, volume 8250 of *Lecture Notes in Computer Science*, pages 98–115, 2013.

[BM03] Johannes Blömer and Alexander May. New partial key exposure attacks on RSA. In *Advances in Cryptology—CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Seience*, pages 27–43, 2003.

[Bre17] Joachim Breitner. More on dliding right. Cryptology ePrint Archive, Report 2018/1163, 2017. http://eprint.iacr.org/2018/.

[CFG+10] Christophe Clavier, Benoit Fiex, Georges Gegnerot, Myléne Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In *International Conference on Information and Communications Security*, volume 6476 of *Lecture Notes in Computer Science*, pages 46–61, 2010.

[CMS15] Stelvio Cimato, Silvia Mella, and Ruggero Susella. New results for partial key exposure on RSA with exponent blinding. In *International Joint Conference on e-Business and Telecommunications*. IEEE, 2015.

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10:233–260, 1997.

[Cor04] Jean-Sébastien Coron. Finding small roots of bivariate integer polynomial equations revisited. In *Advances in Cryptology—Eurocrypt 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 492–505, 2004.

[Cor07] Jean-Sébastien Coron. Finding small roots of bivariate integer polynomial equations: A direct approach. In *Advances in Cryptology—CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Seience*, pages 379–394, 2007.

[EJMdW05] Matthias Ernst, Ellen Jochemsz, Alexander May, and Benne de Weger. Partial key exposure attack on RSA up to full size exponents. In *Advances in Cryptology—Eurocrypt 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 371–386, 2005.

[FKJM+06] Pierre-Alain Fouque, Sébastien Kunz-Jacques, Gwenaëlle Martinet, Frédéric Muller, and Frédéric Valette. Power attack on small RSA public exponent. In *International Workshop on Cryptographic Hardware and Embedded Systems*, volume 4249 of *Lecture Notes in Computer Science*, pages 339–353. Springer, 2006.

[HKT12] Neil Hanley, HeeSeok Kim, and Michael Tunstall. Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. IACR ePrint archive: Report 2012/485, 2012. https://eprint.iacr.org/2012/485.

[HM08] Mathias Herrmann and Alexander May. Solving linear equations modulo divisors: On factoring given any bits. In *Advances in Cryptology—ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Seience*, pages 406–424, 2008.

[HMA+08] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. Collision-based power analysis of modular exponentiation using chosen-message pairs. In *International Workshop on Cryptographic Hardware and Embedded Systems*, volume 5154 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2008.

[HMM10] Wilko Henecka, Alexander May, and Alexander Meurer. Correcting errors in RSA private keys. In *Advances in Cryptology—CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Seience*, pages 351–369, 2010.

[HS09] Nadia Heninger and Hovov Shacham. 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.

[HSH+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Parl, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Communication of the ACM*, 52:91–98, 2009.

[IGI+15] Mehmet Sinan Inci, Berk Galmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Seriouly, get off my cloud! Cross-VM RSA key recovery in a public cloud. IACR ePrint archive: Report 2015/898, 2015. https://eprint.iacr.org/2015/898.

[IGI+16] 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.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc95] Cetin Kaya Kocc. Analysis of sliding window techniques for exponentiation. *Computers & Mathematics with Applications*, 30(10):17–24, 1995.

[KSI13] Noboru Kunihiro, Naoyuki Shinohara, and Tetsuya Izu. Recovering RSA secret keys from noisy key bits with erasures and errors. In *Public Key Cryptography—PKC 2013*, volume 7778 of *Lecture Notes in Computer Science*, pages 180–197, 2013.

[Kun15] Noboru Kunihiro. An improved attack for recovering noisy RSA secret keys and its countermeasure. In *Provable Security*, volume 9451 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2015.

[Lib17] GNU Privacy Guard. https://www.gnupg.org, 2017.

[LYG+15] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy*, pages 244–256. IEEE, 2015.

[OHK19] Kento Oonishi, Xiaoxuan Huang, and Noboru Kunihiro. Improved CRT–RSA secret key recovery method from sliding window leakage. In *International Conference on Information and Communications Security*, volume 11975 of *Lecture Notes in Computer Science*, pages 278–296, 2019.

[OK19] Kento Oonishi and Noboru Kunihiro. Attacking noisy secret CRT–RSA exponents in binary method. In *International Conference on Information and Communications Security*, volume 11396 of *Lecture Notes in Computer Science*, pages 37–54, 2019.

[OK20] Kento Oonishi and Noboru Kunihiro. Recovering CRT–RSA secret keys from noisy square-and-multiply sequences in the sliding window method. In *Australasian Conference on Information Security and Privacy*, volume 12248 of *Lecture Notes in Computer Science*, pages 642–652. Springer, 2020.

[PCBP21] Guilherme Perin, Łukasz Chmielewski, Lejla Batina, and Stjepan Picek. Keep it unsupervised: Horizontal attacks meet deep learning. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, pages 343–372, 2021.

[PITM14] Guilherme Perin, Laurent Imbert, Lionel Torres, and Philippe Maurine. Attacking randomized exponentiations using unsupervised learning. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 8622 of *Lecture Notes in Computer Science*, pages 144–160, 2014.

[PPS12] Kenneth G. Paterson, Antigoni Polychroniadou, and Dale L. Sibborn. A coding-theoretic approach to recovering noisy RSA keys. In *Advances in Cryptology—ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Seience*, pages 386–403, 2012.

[Sch15] Werner Schindler. Exclusive exponent blinding may not suffice to prevent timing attacks on RSA. In *International Workshop on Cryptographic Hardware and Embedded Systems*, volume 9293 of *Lecture Notes in Computer Science*, pages 229–247. Springer, 2015.

[SI11] Werner Schindler and Koichi Itoh. Exponent blinding does not always lift (partial) Spa resistance to higher-level security. In *International Conference on Applied Cryptography and Network Security (ACNS)*, volume 6715 of *Lecture Notes in Computer Science*, pages 73–90. Springer, 2011.

[SIUH22] Kotaro Saito, Akira Ito, Rei Ueno, and Naofumi Homma. One truth prevails: A deep-learning based single-trace power analysis on RSA–CRT with windowed exponentiation. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, pages 490–526, 2022.

[SM09] Santanu Sarkar and Subhamoy Maitra. Partial key exposure attack on CRT–RSA. In *International Conference on Applied Cryptography and Network Security (ACNS)*, volume 5536 of *Lecture Notes in Computer Science*, pages 473–484. Springer, 2009.

[SSS15] Takeshi Sugawara, Daisuke Suzuki, and Minoru Saeki. Two operands of multipliers in side-channel attack. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 9064 of *Lecture Notes in Computer Science*, pages 64–78, 2015.

[SW14] Werner Schindler and Andreas Wiemers. Power attacks in the presence of exponent blinding. *Journal of Cryptographic Engineering*, 4:213–236, 2014.

[SW17] Werner Schindler and Andreas Wiemers. Generic power attacks on RSA with CRT and exponent blinding: new results. *Journal of Cryptographic Engineering*, 7:255–272, 2017.

[TK14] Atsushi Takayasu and Noboru Kunihiro. Partial key exposure attack on RSA: Achieving the Boneh–Durfee bound. In *Selected Areas in Cryptography—SAC 2014*, volume 8781 of *Lecture Notes in Computer Science*, pages 345–362, 2014.

[UTHH21] Rei Ueno, Junko Takahashi, Yu-ichi Hayashi, and Naofumi Homma. A method for constructing sliding windows leak from noisy cache timing information. *Journal of Cryptographic Engineering*, 11:161–170, 2021.

[vV18] Christine van Vredendaal. *Exploiting mathematical structure in cryptography*. PhD thesis, Eindhoven University of Technology, 2018.

[Wal08] C.D. Walter. Sliding windows succumbs to big mac attack. In *International Workshop on Cryptographic Hardware and Embedded Systems*, volume 2162 of *Lecture Notes in Computer Science*, pages 286–299. Springer, 2008.

[Yar18] Yuval Yarom. Mastik: A micro-architectural side-channel toolkit. https://cs.adelaide.edu.au/~yval/Mastik/, Oct 2018.

[YF14] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium*, 2014.

[YGH18] 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.

[ZvdPYS22] Yuanyuan Zhou, Joop van de Pol, Yu Yu, and François-Xavier Standaert. A third is all you need: Extended partial key exposure attack on CRT–RSA with additive exponent blinding. Cryptology ePrint Archive, Paper 2022/1163, 2022. https://eprint.iacr.org/2022/1163.