A Novel Leakage Model in OpenSSL's Miller-Rabin Primality Test

Xiaolin Duan[1], Fan Huang[1], Yaqi Wang[1], and Honggang Hu[1,2]

[1] School of Cyber Science and Technology
University of Science and Technology of China, Hefei 230027, China
[2] Hefei National Laboratory,
University of Science and Technology of China Hefei 230088, China
{duanxl, lanplush, yaqi127}@mail.ustc.edu.cn, hghu2005@ustc.edu.cn

Abstract. At Crypto 2009, Heninger and Shacham presented a branch-and-prune algorithm for reconstructing RSA private keys given a random fraction of its private components. This method is widely adopted in side-channel attacks, and its complexity is closely related to the specific leakage pattern encountered. In this work, we identified a novel leakage model in the Miller-Rabin primality test implemented in OpenSSL. Under certain side-channel attacks against fixed-window modular exponentiation (e.g., recovering the least significant b bits from each window), the proposed model enables staggered recovery of bits in p and q, reducing uncertainty in key reconstruction. In particular, this model includes previously undocumented scenarios where full key recovery is achievable without branching. To understand how the proposed leakage model could contribute to attacks on modular exponentiation, we investigated the global and local behavior of key reconstruction. Our evaluation demonstrates that the proposed scenarios enable more efficient key reconstruction and retain this advantage when additional erasure bits are introduced. Moreover, in specific cases, successful reconstruction remains achievable within practical time even if the bits obtained are less than 50%. Finally, we conducted a series of experiments to confirm the practicality of our assumption, successfully recovering the lower 4 bits from each 6-bit window.

Keywords: Partial key exposure attacks · primality test · modular exponentiation · side-channel attacks.

1 Introduction

RSA is a mainstream public key cryptosystem that has provided various cryptographic services for decades. Its security relies on the practical difficulty of integer factorization and has been intensively discussed by the community. One research direction pursued by cryptanalysis is to examine this assumption when side information is available, known as partial key exposure (PKE) attacks, which seek efficient key reconstruction given a fraction of the secrets. To carry out real-world attacks, a growing field of research has emerged known as side-channel attacks

(SCAs), drawing inspiration from Kocher's seminal works [30, 31]. This area leverages physical information from running cryptosystems, including power, electromagnetics, time, sound, cache footprints, etc., to extract sufficient secret bits. Some of these attacks have considerable strength, allowing them to derive the full key based solely on the physical information obtained [28, 34, 35, 39]. In practice, noise and countermeasures usually result in incomplete key acquisition, and PKE attacks are required to finish the analysis [8, 9, 47, 48].

1.1 Partial Key Exposure Attacks on RSA

Ideally, cryptosystems are expected to offer a certain security strength even if partial key bits are compromised, but it is well known that RSA does not enjoy this property. For security purposes, it is critical to know how many partial bits of the secret components suffice to break the system.

Let the public key be (N, e) and the private key be (p, q, d), with $N = pq$ and $d \equiv e^{-1} \mod (p-1)(q-1)$. n denotes the length of p and q. In 1985, Rivest and Shamir [42] indicated that given 2/3 least significant bits (LSBs) of p or q is sufficient to break RSA in polynomial-time. In Coppersmith's pioneered work [20], this result was improved to 1/2 of the most significant bits (MSBs) or LSBs of a factor by utilizing the lattice reduction technique. Later, Howgrave-Graham [27] presented an alternative method to settle this problem by introducing the dual lattice. In 1998, Boneh et al. [11] presented a polynomial-time attack when a quarter of the LSBs of d are given. As the Chinese Remainder Theorem (CRT) is often used to speed up RSA implementation, in 2003, Blömer and May [10] investigated PKE on CRT-RSA and stated that known half of the bits of $d \mod (p-1)$ could efficiently factorize N.

In 2008, Herrmann and May [26] demonstrated that the problem of factorizing N could be solved in polynomial-time when given at most $\mathcal{O}(\log \log N)$ blocks of bits. Concurrently, Halderman et al. [23] proposed cold boot attacks and introduced a new leakage model that reveals a random fraction of secret bits. According to the bound presented in [26], lattice-based techniques are not directly applicable to this model. In 2009, Heninger and Shacham [25] observed substantial redundancy in RSA key storage. Based on cold boot attacks, they developed a branch-and-prune algorithm that could efficiently recover the private key given a random fraction of p, q, d, d_p, d_q.

Heninger and Shacham considered an idealized scenario in which all given information is error-free, later termed the erasure model. Under this assumption, their algorithm reconstructs the key deterministically. However, when the given bits contain errors, this algorithm fails as the correct solution is inevitably pruned during the process. In 2010, Henecka et al. [24] proposed a probabilistic algorithm capable of correcting a secret key corrupted by random bit-flip errors. This work was subsequently refined by Paterson et al. [40] with their asymmetric bit-flip model. Later, Kunihiro et al. [32] analyzed the key reconstruction in the presence of both errors and erasures. Recently, Saito et al. [44] adapted Henecka's algorithm to efficiently handle leakage in windowed exponentiation.

1.2 Side-Channel Attacks on RSA Modular Exponentiation

Numerous attacks have been proposed to exploit vulnerabilities in RSA implementations, including but not limited to modular exponentiation [1, 2, 4, 5, 9, 12, 15, 16, 21, 22, 28, 29, 33–36, 38, 41, 43–45, 47–49], binary GCD computation [8], modular inversion [3, 6, 39], key validation [46] and pseudorandom number generators [18, 19].

Modular exponentiation is a core operation in the entire life cycle of RSA, spanning key generation, encryption, and decryption. Its implementation in cryptographic libraries has undergone rigorous scrutiny and several significant iterations to enhance security and reliability. The square-and-multiply method, which exploits the binary representation of the exponent, is less efficient and has been proven to be insecure in Kocher's timing attack [30]. To improve performance, modern open-source cryptographic libraries employ window exponentiation techniques, including fixed-window exponentiation (as in OpenSSL) and sliding-window exponentiation (as in Libgcrypt).

However, windowed exponentiation alone cannot guarantee constant-time execution, as loading precomputed multipliers remains vulnerable to cache-based attacks [9, 41, 48]. To mitigate this issue, a dummy load technique has become a widely adopted countermeasure. This method works by loading all precomputed operands and then selecting the desired value through bitwise masking operations, thereby eliminating observable timing differences.

Recent studies [5, 29, 44] have demonstrated successful single-trace attacks against this countermeasure, achieving high-accuracy secret key recovery. Specifically, Alam et al. [5] presented an EM-based attack that targets the exponent extraction mechanism in OpenSSL's modular exponentiation, successfully circumventing this countermeasure. Saito et al. [44] developed a deep learning (DL)-based classifier, leveraging power analysis, to reliably distinguish secret-dependent memory accesses from dummy operations. Hu et al. [29] showed that the dummy load technique introduces measurable power differences during the computation of masks, enabling efficient key extraction through simple power analysis. Interestingly, none of these three studies targeted modular exponentiation in key generation. Our findings reveal this to be a fundamentally different scenario.

1.3 Motivation

Lattice-based RSA key reconstruction requires knowledge of partial consecutive bits, whereas Heninger and Shacham's branch-and-prune algorithm excels at handling dispersed known bits, making it more suitable for SCAs. Previous SCAs often yield sufficient information to enable successful key reconstruction via the branch-and-prune method, rarely approaching the algorithm's theoretical limits.

The complexity of the branch-and-prune algorithm varies depending on the specific leakage pattern. For instance, consider an adversary who obtains 50% of the bits of p and q. As demonstrated by [25] and [40], the key can be reconstructed using the branch-and-prune algorithm when these bits are randomly

distributed. If the 50% known bits are consecutive least significant bits, the algorithm faces significant computational challenges, requiring examination of up to $2^{n/2}$ candidates to recover the remaining bits, rendering the problem computationally infeasible in practice. However, when the known bits follow an alternating pattern, the key can be reconstructed without generating false candidates, demonstrating that the algorithm can handle erasure rates exceeding 50% in this scenario.

An intriguing question arises: Does the scenario where p and q are recovered in an alternating pattern exist in practical settings?

1.4 Our Contributions

This paper presents a novel leakage model in constant-time modular exponentiation invoked by the Miller-Rabin primality test in OpenSSL. The main contributions of this work are summarized as follows:

- Our study reveals a previously unnoticed vulnerability in the Miller-Rabin primality test in OpenSSL, and a novel leakage model is developed based on this finding. Assuming that an adversary could extract b bits from each window in fixed-window modular exponentiation, this model encompasses scenarios that did not arise in previous SCAs, referred to as misalignment. According to the branching behavior of the branch-and-prune algorithm, such misalignment could reduce the uncertainty in key reconstruction, thereby increasing the efficiency of this process. Some scenarios could even rebuild the key without generating false solutions.
- To evaluate the complexity of reconstructing p and q in the new model, we extended the discussion of global and local behavior of the branch-and-prune algorithm from bit to window size. Additionally, we estimated the average number of false solutions examined in key reconstruction, illustrating that misalignment cases contribute to a more efficient reconstruction process. In certain proposed scenarios, key reconstruction can be completed instantly without branching, demonstrating tolerance for additional erasures. Our analysis confirms that the speed advantage in misaligned cases persists while the introduced erasures approach the algorithm's maximum tolerance. When $\frac{b}{w} = 50\%$, where w is window size, the computational cost remains reasonable even with the additional erasure bits introduced.
- We analyzed the likelihood of proposed scenarios occurring by repeatedly invoking OpenSSL's key generation, confirming that misaligned scenarios exhibit non-negligible occurrence probabilities. To demonstrate the practical implications of our assumption, we performed SPA attacks targeting the fixed-window modular exponentiation in OpenSSL's Miller-Rabin primality test, running on an ARM Cortex-M4 processor. These attacks successfully recovered the least significant 4 bits from each 6-bit window.

In conclusion, attacking modular exponentiation in the Miller-Rabin primality test offers three advantages: (1) The key reconstruction process benefits from

our new leakage model's efficiency and erasure tolerance. (2) An adversary can directly recover p and q without considering the implementation of the Chinese Remainder Theorem and blinding[3]. (3) OpenSSL recommends at least 64 rounds of tests to achieve a false positive rate of 2^{-128}, indicating that 64 modular exponentiations will be invoked in the Miller-Rabin primality test. This implementation allows an adversary to obtain more side-channel information from a single trace generated during key generation.

1.5 Paper Organization

The remainder of the paper is organized as follows. Section 2 gives the preliminaries of the leakage model and describes Heninger and Shacham's work. Section 3 details the vulnerability in the Miller-Rabin primality test implemented in OpenSSL and introduces our proposed leakage model. In Section 4, we conducted a comprehensive evaluation of the global and local behavior of the key reconstruction based on the new model, including analysis of scenarios with additional erasures. Section 5 demonstrates our practical attack and experimental results. Section 6 offers concluding remarks.

2 Background

2.1 Fixed-Window Modular Exponentiation in OpenSSL

Several algorithms have been suggested for RSA modular exponentiation, including the square-and-multiply, the sliding-window, and the fixed-window approach. The latter has gained widespread adoption in cryptographic libraries due to its inherent feature against leaking the square and multiplication sequence.

Given a fixed window of size ω, a base a, a modular N, and an n-bit exponent p denoted as a sequence of digits in base 2^ω, fixed-window exponentiation outputs $r = a^p \mod N$ using two steps. It first computes a set of multipliers $a_j = a^j \mod N$ for $0 \leq j \leq 2^\omega - 1$. Then from the most significant to the least significant digit of p, r is accumulated by squaring ω times and multiplying a_{p_i} in each iteration.

A naive implementation of fixed-window exponentiation suffers from severe lookup table leaks, especially from cache-based channels. OpenSSL mitigated this vulnerability by integrating dummy load operations with a scatter-gather memory layout, following Intel's recommendation [13, 14]. This technique partitions each precomputed multiplier into blocks and interleaves blocks from different multipliers within the same cache line, thereby making cache-line access

[3] Attacks on modular exponentiation during decryption typically recover the private exponent d. Due to the widespread use of the Chinese Remainder Theorem in practical RSA implementations, such an attack often acquires partial information of d_p and d_q. As demonstrated in CacheBleed [48], this necessitates guessing both k_p and k_q (65,537 possible pairs), significantly increasing the key reconstruction time.

patterns independent of the specific multiplier being processed. Under this memory layout, identifying specific cache line accesses could leak partial bits of the window exponent, necessitating dummy loads to obscure cache access patterns. Building on this implementation, Yarom et al. [48] devised CacheBleed, a sub-cache-line attack capable of exposing the least significant three bits of each window through cache bank conflicts. In response, OpenSSL 1.0.2g implemented a patch that iterates over all chunks of each multiplier and applies a masking operation for selection. This countermeasure remains active by default in current OpenSSL releases.

2.2 Miller-Rabin Primality Test

Cryptographic libraries typically call a random number generator followed by a primality test to obtain secret components p and q. The Miller-Rabin test, a probabilistic algorithm that returns whether a given number is likely to be prime, is frequently employed to check for prime numbers.

Given a prime $p = 2^s \cdot p' + 1$ and an integer base a such that $0 < a < p$, then one of the following congruence relations holds:

- $a^{p'} \equiv 1 \mod p$;
- $a^{2^r \cdot p'} \equiv -1 \mod p$ for some $0 \leq r < s$.

This can be proved based on two facts:

- Fermat's litter theorem: $a^{n-1} \equiv 1 \mod p$;
- The only square roots of 1 modulo p are 1 and -1.

However, the inverse of the above property is incorrect. One of the above congruence relations holding does not guarantee p is prime. It only promises such p is a strong probable prime to base a. A small fraction of composites, known as strong pseudoprimes, also pass this test. The Miller-Rabin algorithm assesses the primality of a given number with different bases. If enough tests are passed, the number is said to be prime with high probability. The number of tests is determined according to the size of p and the desired probability. For example, the latest version of OpenSSL recommends at least 64 rounds of tests to attain a false positive rate of 2^{-128}.

2.3 Probability Generating Functions

The probability generating function (PGF) refers to the power series expression of the probability mass function of the random variable. It is often used to describe the probability distribution of a discrete random variable.

Definition 1 (Probability Generating Functions). *Let X be a discrete random variable taking values in the non-negative integers. The PGF of X is defined as*

$$G_X(s) = \mathrm{E}\, s^X = \sum_{x=0}^{\infty} s^x \mathrm{P}(X = x),$$

where E *denotes the expectation and* P *is the probability mass function.*

Algorithm 1 Miller-Rabin Primality Test

Require: prime candidate $p = 2^s \cdot p' + 1$, number of rounds j,
Ensure: statement "p is composite" or "p is likely prime"

```
for i = 1 to j do
    choose a random a ∈ {2, 3, ..., p − 2};
    u ← a^{p'} mod p;
    if u ≠ 1 and u ≠ p − 1 then
        for j = 1 to s − 1 do
            u = u^2 mod p
            if u = 1 then
                return ("p is composite")
            end if
        end for
        if u ≠ p − 1 then
            return ("p is composite")
        end if
    end if
end for
return ("p is likely prime")
```

The following properties can be obtained according to the definition of PGF, expectation, and variance.

- $G_X(1) = \sum_{x=0}^{\infty} \mathrm{P}(X = x) = 1$;
- $\mathrm{E}\, X = G'_X(1)$;
- $\mathrm{Var}\, X = G''_X(1) + G'_X(1) - (G'_X(1))^2$.

Theorem 1. *Assume that X_1,...,X_n are independent random variables, and $Y = X_1 + ... + X_n$. Then*

$$G_Y(s) = \prod_{i=1}^{n} G_{X_i}(s).$$

Theorem 2. *Assume that $\{X_i\}$ is a sequence of independent and identically distributed random variables with the common PGF G_X. Let $Y = X_1 + ... + X_M$, where M is a random variable independent of X_i. Denoting the PGF of M as G_M, the PGF of Y is*

$$G_Y(s) = G_M(G_X(s)).$$

2.4 The Heninger-Shacham Algorithm

At Crypto 2009, Heninger and Shacham [25] utilized the algebraic relationship between the private components to develop a branch-and-prune technique for key reconstruction with known bits. They discussed this algorithm in the context of cold boot attacks and demonstrated a lower bound on the number of known bits required for the algorithm to succeed within polynomial-time.

Let (N, e) be an RSA public key, where N is $2n$-bit and $e = 65,537$. The corresponding CRT private key set is $(p, q, d, d_p, d_q, q_{inv})$, which simultaneously satisfies the following equations.

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

where k, k_p and k_q can be enumerated if e is small. The key exhibits high levels of redundancy, as knowledge of any of its components can lead to the factorization of N. Heninger-Shacham's algorithm leverages these constraints to progressively determine the parameters. It starts from the least significant bit, and at each iteration, partial solutions are branched or pruned depending on the knowledge of bits for that position. Specifically, given the knowledge of bits 0 to $i-1$, the i-th bit of each argument can be solved by lifting the solution to the constraint equations mod 2^i to the solution mod 2^{i+1}.

Cold boot attacks could result in partial information of $(p, q, d, d_p, d_q, q_{inv})$ being obtained at random. However, in other side-channel models, only some of them may be obtained, such as $\{p, q\}$, $\{d\}$, or $\{d_p, d_q\}$. This study focuses on the first case in which partial knowledge of p and q is revealed. The following part reviews the behavior of the algorithm in this scenario.

According to Hensel's lifting lemma, the dependency between the i-th bit and known bits is expressed as

$$p[i] + q[i] \equiv (N - \bar{p}\bar{q})[i] \mod 2, \tag{1}$$

where $\bar{p}$ and $\bar{q}$ denote the partial solution of p and q up to i of the bits. Given $\bar{p}$ and $\bar{q}$, the algorithm's branching behavior at bit i can be summarized as follows.

- When both $p[i]$ and $q[i]$ are unknown, the algorithm branches;
- When one of $p[i]$ and $q[i]$ is unknown, there is a unique solution;
- When the i-th bit of p and q are known, the wrong solution is pruned with approximate 50% probability (empirically).

Understanding these cases is straightforward, as the right side of equation (1) is given.

Heninger and Shacham conducted a complexity analysis of the reconstruction by examining the algorithm's local and global branching behavior. They evaluated the number of erroneous solutions generated during the process. Specifically, at bit i, the algorithm generates one correct solution and some incorrect solutions. As the number of unknown bits increases, the expansion of branches tends to be exponential; conversely, it converges swiftly when the number of unknown bits decreases.

In their random model, it is assumed that the knowledge of $p[i]$ and $q[i]$ is mutually independent, each occurring with a probability of δ. To establish the relation between the number of branches examined during the reconstruction and

δ, the authors calculated the expected number of wrong solutions generated from a good and incorrect one, denoted as $\mathrm{E}\, Z_g$ and $\mathrm{E}\, W_b$, respectively. Subsequently, using PGF and the sequence, they derived the expected total number of branches produced at step i, represented as $\mathrm{E}\, X_i = \frac{\mathrm{E}\, Z_g}{1-\mathrm{E}\, W_b}(1 - (\mathrm{E}\, W_b)^i)$. Given that the expressions for the expected values of $\mathrm{E}\, Z_g$ and $\mathrm{E}\, W_b$ are dependent solely on δ, it follows that when $\mathrm{E}\, W_b < 1$, $\mathrm{E}\, X_i$ can be bounded by a constant that only depends on δ.

Adding up the incorrect solutions generated at each bit, one can calculate the expected total number of branches examined for an n-bit key. Heninger and Shacham also provided the variance of X_i as a measure of how far $\sum_{i=0}^{n-1} X_i$ is spread from its average value. The following lemma plays an important role in this computation.

Lemma 1.

$$\mathrm{Var} \sum_{i=1}^{n} X_i \leq n^2 \max_i \mathrm{Var}\, X_i$$

In sum, if the algorithm has partial knowledge of p and q, it checks no more than $29n^2 + 29n$ keys with a probability higher than $\frac{n^2-1}{n^2}$ when $\delta = 0.59$.

3 A Novel Leakage Model

This section presents a vulnerability identified in the Miller-Rabin primality test implemented in OpenSSL. Assuming that an adversary is capable of extracting b bits from each window in fixed-window modular exponentiation, this vulnerability leads to a new leakage model that may minimize the uncertainty in Heninger and Shacham's reconstruction method, thereby potentially reducing the number of keys that need to be checked. The vulnerability arises for two reasons:

- The input of modular exponentiation in the primality test is $p' = \frac{p-1}{\tau(p-1)}$, where $\tau(p-1)$ denotes the largest power of 2 that divides by $p-1$;
- At the beginning of modular exponentiation, OpenSSL pads p' to the public size of p, which is n.

The most significant 0 bits of the exponent could be disclosed by analyzing the number of iterations performed in modular exponentiation. To avoid such a leakage, OpenSSL uses all bits stored in the exponent variable. Listing 1 is a code snippet of constant-time modular exponentiation used in OpenSSL. It computes $rr = a^p \mod m$. **BIGNUM** is a structure that holds a single large integer. The member variable **top** represents the number of words used. **BN_BITS2** indicates the word size specified in the *bn.h* file. Hence, **bits** corresponds to the public size of the exponent, which ensures that the number of iterations in fixed-window modular exponentiation is independent of the actual size of p.

```
1 int bn_mod_exp_mont_fixed_top(BIGNUM *rr, const BIGNUM *a
      , const BIGNUM *p, const BIGNUM *m, BN_CTX *ctx,
      BN_MONT_CTX *in_mont)
```

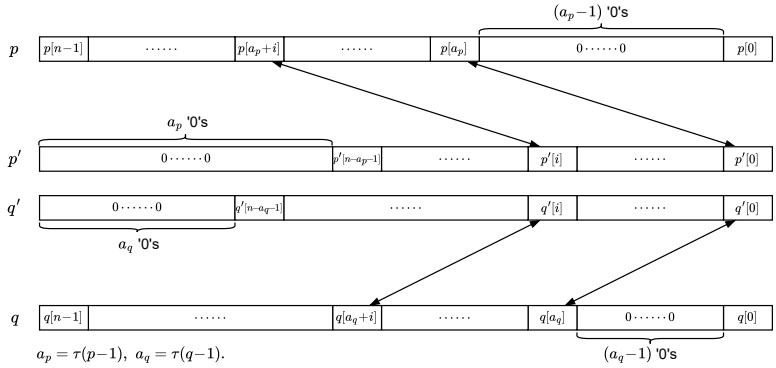


Fig. 1: Map the recovered $p'[i]$ and $q'[i]$ to p and q.

```
2 {
3     ...
4     /* Use all bits stored in stored in variable p to
      prevent leakage of leading zero bits. */
5     bits = p->top * BN_BITS2;
6     ...
7 }
```

Listing 1: OpenSSL pads the exponent to its public size.

The operands involved in modular exponentiation are p' or q', extended to a length of n. In our assumed side-channel model, p' and q' are recovered in the same manner. So, $p'[i]$ and $q'[i]$ are known, or neither is known. However, this does not necessarily imply that the same relationship exists between $p[i]$ and $q[i]$.

Upon reconstructing the key using equation (1), the recovered $p'[i]$ and $q'[i]$ are mapped to $p[i + \tau(p-1)]$ and $q[i + \tau(q-1)]$. Figure 1 shows the mapping from p' and q' to p and q, where

$$p = p'2^{\tau(p-1)} + 1,$$
$$q = q'2^{\tau(q-1)} + 1.$$

Suppose $\tau(p-1) > \tau(q-1)$, both p and q can be partitioned into three blocks, as shown in Figure 2. Specifically,

- **Block 1** ($\tau(q-1) > i \geq 0$): $p[i] = q[i] = 0$, except $p[0] = q[0] = 1$;
- **Block 2** ($\tau(p-1) > i \geq \tau(q-1)$): $p[i] = 0$ and $q[i]$ may be revealed;
- **Block 3** ($n-1 \geq i \geq \tau(p-1)$): each window of p (or q) is revealed in a fixed pattern.

When $\tau(p-1) < \tau(q-1)$, the situation is analogous. **Block 2** is absent when $\tau(p-1) = \tau(q-1)$.

Given $\tau(p-1)$ and $\tau(q-1)$, **Block 1** is fully determined, and either p or q in **Block 2** can be uniquely identified, yielding exactly one valid partial solution

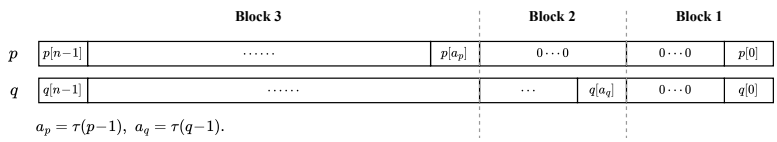


Fig. 2: p and q recovered from the side-channel oracle.

for **Block 3**. Consequently, the efficiency of key reconstruction depends on its performance in **Block 3**, subject to the knowledge of $p[i]$ and $q[i]$ within each window. To formalize our new model, we categorize **Block 3** into two cases.

- The aligned case: $p[i]$ and $q[i]$ are either recovered or unknown;
- The misaligned case: $p[i]$ and $q[i]$ could be one of the following: both recovered, one recovered, or both unknown.

Table 1 demonstrates both scenarios for a window size of 6, with 4 bits successfully recovered in each window. The remainder of this paper will utilize this assumption as a case study and present corresponding experimental validations. Our work is not constrained by this specific assumption. In Section 4, we develop a generalized analysis that extends our results to various assumptions, including 3-bit recovery from 5-bit windows[4] and 2-bit recovery from 4-bit windows[5].

Table 1: Left: aligned case; Right: misaligned case.

$i+2$	$i+1$	i	$i-1$	$i-2$	$i-3$
X	X	1	0	1	0
X	X	1	0	1	1

$i+2$	$i+1$	i	$i-1$	$i-2$	$i-3$
X	X	1	0	1	0
1	0	1	1	X	X

The occurrence of each scenario is determined by the absolute value of the difference between $\tau(p-1)$ and $\tau(q-1)$, denoted as τ. Notably, the recovered bits exhibit a periodic pattern within each window. This establishes τ as defining a ring over the integer w. For simplicity, we omit the modulus of $\tau \equiv x \mod w$ in subsequent discussions.

- $\tau \equiv 0$ indicates the case of alignment;
- $\tau \not\equiv w$ indicates the case of misalignment.

In the subsequent section, we demonstrate that the reconstruction process for the misaligned case exhibits a lower complexity due to the reduced level of uncertainty.

[4] This leakage pattern has been demonstrated in [48].
[5] This leakage pattern represents the specific case we analyzed for 1024-bit RSA.

4 Algorithm Runtime Analysis

This section explores the complexity of key reconstruction in different cases in the proposed leakage model by assessing the number of partial solutions examined during the process.

4.1 Global Branching Behavior

Given that the cases of the proposed leakage model differ only in their window-level recovery patterns, we first establish a general equation characterizing the algorithm's global behavior, then investigate the local branching process in the window.

Let $Y_\imath$ denote the number of partial solutions checked at window $\imath$ in **Block 3**, and a sum of $Y_\imath$ yields the total number of partial keys examined in the third block, where $1 \leq \imath \leq \lceil (n - max(\tau(p-1), \tau(q-1)))/w \rceil$.

$$Y_\imath = \sum_{\jmath=1}^{X_{\imath-1}} Z_\jmath + Zc, \tag{2}$$

where Z and Zc are random variables that count, within a window, the number of key validation attempts triggered by a bad and a good solution, respectively. There is only one correct partial key in the whole process. Since the number of incorrect solutions is not constant, it is imperative to estimate how many incorrect solutions are likely to remain after lifting the window $\imath$, denoted as $X_\imath$.

Let C and W be two random variables representing the number of incorrect candidates remaining after a good and a bad solution passes a window in **Block 3**, respectively. Then

$$X_\imath = \sum_{\jmath=1}^{X_{\imath-1}} W_\jmath + C. \tag{3}$$

To depict the distribution of $Y_\imath$, we have the following expressions. The derivation is performed using a generating function and can be found in Appendix A.

Theorem 3.

$$\mathrm{E}\, Y_\imath = \frac{\mathrm{E}\, C\, \mathrm{E}\, Z}{1 - \mathrm{E}\, W}(1 - (\mathrm{E}\, W)^{\imath-1}) + \mathrm{E}\, Zc. \tag{4}$$

Theorem 4.

$$\mathrm{Var}\, Y_\imath = c_3(\mathrm{E}\, W)^{2(\imath-1)} - c_2(\mathrm{E}\, W)^{\imath-1} + c_1, \tag{5}$$

with

$$
\begin{aligned}
c_1 &= \frac{\mathrm{E}\,C\,\mathrm{Var}\,W + (1-\mathrm{E}\,W)\mathrm{Var}\,C}{(1-(\mathrm{E}\,W)^2)(1-\mathrm{E}\,W)}(\mathrm{E}\,Z)^2 \\
&\quad + \frac{\mathrm{E}\,C}{1-\mathrm{E}\,W}\mathrm{Var}\,Z + \mathrm{Var}\,Zc, \\
c_2 &= \frac{\mathrm{E}\,C\,\mathrm{Var}\,W}{(1-\mathrm{E}\,W)^2\mathrm{E}\,W}(\mathrm{E}\,Z)^2 + \frac{\mathrm{E}\,C}{1-\mathrm{E}\,W}\mathrm{Var}\,Z, \\
c_3 &= (\frac{\mathrm{E}\,C\,\mathrm{Var}\,W}{(1-(\mathrm{E}\,W)^2)(1-\mathrm{E}\,W)\mathrm{E}\,W} - \frac{\mathrm{Var}\,C}{1-(\mathrm{E}\,W)^2})(\mathrm{E}\,Z)^2.
\end{aligned}
$$

The expression presented above is derived under the assumption that $\mathrm{E}\,W \neq 1$. For the sake of generality, when $\mathrm{E}\,W = 1$,

$$\mathrm{E}\,Y_\imath = (\imath - 1)\,\mathrm{E}\,C\,\mathrm{E}\,Z + \mathrm{E}\,Zc. \tag{6}$$

Similarly,

$$
\begin{aligned}
\mathrm{Var}\,Y_\imath = (&\frac{(\imath-1)(\imath-2)}{2}\mathrm{E}\,C\,\mathrm{Var}\,W + (\imath-1)\mathrm{Var}\,C)(\mathrm{E}\,Z)^2 \\
&+ (\imath-1)\,\mathrm{E}\,C\,\mathrm{Var}\,Z + \mathrm{Var}\,Zc.
\end{aligned} \tag{7}
$$

4.2 Local Branching Behavior at Each Window

Recall that the program's local behavior at each bit depends on the knowledge of $p[i]$ and $q[i]$. Within a window, the leakage pattern is fixed by providing w, b, and τ, where w is given by the implementation of modular exponentiation, b is subject to the attacker's capabilities, and τ is determined by randomly generated p and q. In the reconstruction process, the only variable is the probability of a wrong branch that will be pruned when $p[i]$ and $q[i]$ are known, denoted as φ.

For the sake of discussion, we consider the scenario where the least significant b bits of each window are recovered. When $\tau \equiv 0$, a correct partial solution passes the window and generates $2^{w-b}-1$ wrong candidates. An incorrect partial solution produces 2^{w-b} erroneous candidates after passing the evaluation with probability $(1-\varphi)^b$. This is the case of alignment.

Depending on the degree of misalignment, the number of situations where $p[i]$ and $q[i]$ are unknown within a window, denoted as Δ, could be reduced at most to

$$\begin{cases} 0 & \text{if } w-b \leq b, \\ w-2b & \text{if } w-b > b. \end{cases}$$

Considering the attack on fixed-window modular exponentiation, $w - b > b$ implies that less than half of the bits of p and q are obtained. This situation produces a large number of incorrect candidates, rendering key reconstruction infeasible within a reasonable time.[6] Hence, only the case of $w-b \leq b$ is discussed in this study.

[6] According to [40], the branch-and-prune algorithm works if more than 50% of the secret bits are acquired. Although known bits could be slightly lower than 50%, such as $p[i]$ and $q[i]$ are recovered alternatively, $w - b > b$ indicates two many erasures.

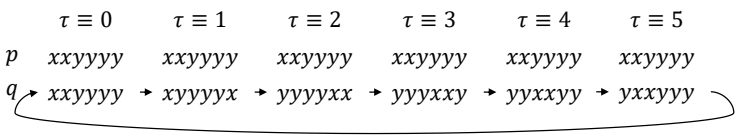


Fig. 3: Take $w = 6$ and $b = 4$ as an example to illustrate the leakage pattern of the window in **Block 3**.

Assuming $\tau(p-1) \geq \tau(q-1)$, $\overbrace{x...x}^{w-b}\ \overbrace{y...y}^{b}$ denotes the least window of p in **Block 3**, where x and y represent unknown and known bits, respectively.[7] As τ increases from 0 to w, the leakage pattern in the corresponding window of q constructs a cycle, while Δ initially decreases from $w-b$ to 0 and then increases back to $w-b$, as shown in Figure 3.

Based on the branching behavior exhibited by the algorithm, it can be observed that the variable C is directly related to Δ.

$$\mathrm{E}\, C = 2^{\Delta} - 1, \qquad \mathrm{E}\, C^2 = (2^{\Delta} - 1)^2.$$

Let Λ denote the number of bits where $p[i]$ and $q[i]$ are known in the window. Λ determines the probability of retaining an incorrect input solution, Δ specifies how this solution forks.

$$\mathrm{E}\, W = 2^{\Delta}(1-\varphi)^{\Lambda}, \qquad \mathrm{E}\, W^2 = 2^{2\Delta}(1-\varphi)^{\Lambda}.$$

When discussing the number of partial keys verified within a given window, it should be noted that Z and Zc are determined by Δ, Λ, and the position of known bits. Consider $\tau \equiv 1$ and $\tau \equiv 5$ depicted in Figure 3. When $\tau \equiv 1$, a correct partial key passes through five bits and branches at the last bit. However, the latter case produces a false branch at the 5-th bit, requiring an additional calculation at the 6-th bit.

The general expressions for the expected values and variances of Z and Zc require knowledge of the position of known bits, leading to complex expressions. To simplify the computation, we provide $\mathrm{E}\, Zc$, $\mathrm{E}\, Zc^2$, $\mathrm{E}\, Z$, and $\mathrm{E}\, Z^2$ under specific w and b, as shown in Table 2. These results can be directly extended to other scenarios.

To demonstrate the results in Table 2, we use the alignment case as an example. After the first 4 evaluations, a correct solution passes the lower 4 bits of the window. Two branches are generated after evaluating the next bit. Then, two candidates are evaluated at the last bit. Thus, a correct solution involves a total of 7 evaluations. In contrast, an incorrect solution may be discarded after each of the first 4 evaluations with a probability of φ. Once an incorrect solution is retained with probability $(1-\varphi)^4$, it behaves identically to a correct solution in the last two bits. Consequently, $\mathrm{E}\, Z = \sum_{i=1}^{4} i(1-\varphi)^{i-1}\varphi + 7(1-\varphi)^4$.

Under the random model, [25] empirically demonstrates that the probability of independently pruning an incorrect candidate approaches 1/2 for known $p[i]$

[7] $\tau(p-1) < \tau(q-1)$ is analogous since p and q are equivalent here.

and $q[i]$. Since our model can be viewed as a special case of this random model, $\varphi = 1/2$ is expected to hold here. Our simulations confirm this expectation.

Table 2: The expected values and variances for Z and Zc when $w = 6$ and $b = 4$.

	$\tau \equiv 0$	$\tau \equiv 1$	$\tau \equiv 2$	$\tau \equiv 3$	$\tau \equiv 4$	$\tau \equiv 5$
$\mathrm{E}\, Zc$	7	6	6	6	6	7
$\mathrm{E}\, Zc^2$	49	36	36	36	36	49
$\mathrm{E}\, Z$	$\frac{33}{16}$	3	4	3	$\frac{5}{2}$	$\frac{9}{4}$
$\mathrm{E}\, Z^2$	$\frac{107}{16}$	$\frac{43}{4}$	$\frac{35}{2}$	$\frac{27}{2}$	$\frac{21}{2}$	$\frac{35}{4}$

4.3 Bounding the Total Number of Keys Examined

The total number of keys examined over an entire program can be represented as

$$Y = |\tau(p-1) - \tau(q-1)| + \sum_{\imath=1}^{l} Y_\imath + \varepsilon,$$

where $l = \lfloor (n - max(\tau(p-1), \tau(q-1)))/w \rfloor$ is the number of complete windows in **Block 3**. Specifically, the first term counts the evaluations executed in blocks 2. The second term plus ε represents the total number of keys examined in **Block 3**, where ε denotes the number of keys evaluated in the last window. If $n - max(\tau(p-1), \tau(q-1))$ is divisible by w, $\varepsilon = 0$. For $\mathrm{E}\, W \leq 1$ ($w - b \leq b$), the program either converges continuously or alternates between divergence and convergence in each window. Thus, the tree does not spread exponentially, which implies that ε will not be too large. In addition, empirical investigations demonstrate that $\tau(p-1)$ and $\tau(q-1)$ are usually small, as shown in Figure 4. Therefore, the primary contribution to $\mathrm{E}\, Y$ arises from the accumulation of $\mathrm{E}\, Y_\imath$.

Given w, b, τ, and estimation of φ, $\mathrm{E}\, C$, $\mathrm{E}\, W$, $\mathrm{E}\, Zc$, and $\mathrm{E}\, Z$ are constants. According to Theorem 3, we give a bound on $\mathrm{E}\, Y$ to facilitate estimation.

$$\mathrm{E}\, Y < \mathrm{E} \left[\sum_{\imath=1}^{\lceil \frac{n}{w} \rceil} Y_\imath \right] \tag{8}$$

$$\begin{cases} \leq \left\lceil \dfrac{n}{w} \right\rceil \left(\dfrac{\mathrm{E}\, C\, \mathrm{E}\, Z}{1 - \mathrm{E}\, W} + \mathrm{E}\, Zc \right) & \text{if } \mathrm{E}\, W < 1, \\ = \dfrac{\left\lceil \frac{n}{w} \right\rceil^2 - \left\lceil \frac{n}{w} \right\rceil}{2} \mathrm{E}\, C\, \mathrm{E}\, Z + \left\lceil \dfrac{n}{w} \right\rceil \mathrm{E}\, Zc & \text{if } \mathrm{E}\, W = 1. \end{cases}$$

Next, Lemma 1 gives a bound for the variance of $\sum_\imath Y_\imath$, where $1 \leq \imath \leq \lceil \frac{n}{w} \rceil$.

$$\mathrm{Var} \sum_\imath Y_\imath \leq \left\lceil \frac{n}{w} \right\rceil^2 \max_\imath \mathrm{Var}\, Y_\imath$$

From equation 5, we observe that it converges to c_1 as $\imath$ increases. When $\imath = 1$, $c_3 - c_2 \leq 0$. As $\imath$ grows, the expression $c_3(\mathrm{E}\, W)^{2(\imath-1)} - c_2(\mathrm{E}\, W)^{\imath-1}$ remains negative but increases monotonically. Consequently, $\max \mathrm{Var}\, Y_\imath = c_1$ at $\imath = \left\lceil \frac{n}{w} \right\rceil$, where c_1 is a constant determined by w, b, τ, and the estimate of φ. The result holds when $\mathrm{E}\, W = 1$, where the maximum variance occurs at $\imath = \left\lceil \frac{n}{w} \right\rceil$.

Table 3 gives $\mathrm{E}\, Y$ and $\mathrm{Var}\, Y$ when $w = 6$, $b = 4$, and $\varphi = 1/2$. Our analysis reveals that both: (i) the expected number of keys examined during reconstruction, and (ii) their degree of dispersion are lower in misaligned cases compared to aligned cases. Despite the fact that the number of recovered bits remains constant, the system exhibits significantly reduced uncertainty in misalignment scenarios. Furthermore, the cases in which $\tau \equiv 2, 3, \text{and } 4$ yield identical estimates for $\mathrm{E}\, Y$ and $\mathrm{Var}\, Y$. These cases can be treated as equivalent in subsequent analyses. Although cases where $\tau \equiv 2$ and $\tau \equiv 3$ show minor differences in $\mathrm{Var}\, Y$, we classify them into the same category. This grouping is justified because both have exactly one bit where $p[i]$ and $q[i]$ are recovered. The observed difference in variance arises from the position of this bit.

Table 3: $\mathrm{E}\, Y$ and $\mathrm{Var}\, Y$ when $w = 6$, $b = 4$, and $\varphi = 1/2$.

	$\tau \equiv 0$	$\tau \equiv 1$	$\tau \equiv 2$	$\tau \equiv 3$	$\tau \equiv 4$	$\tau \equiv 5$
$\mathrm{E}\, C$	3	1	0	0	0	1
$\mathrm{E}\, W$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$\mathrm{E}\, Zc$	7	6	6	6	6	7
$\mathrm{E}\, Z$	$\frac{33}{16}$	3	4	3	$\frac{5}{2}$	$\frac{9}{4}$
$\mathrm{E}\, Y <$	$\frac{61}{4}\left\lceil \frac{n}{6} \right\rceil$	$10\left\lceil \frac{n}{6} \right\rceil$	$6\left\lceil \frac{n}{6} \right\rceil$	$6\left\lceil \frac{n}{6} \right\rceil$	$6\left\lceil \frac{n}{6} \right\rceil$	$10\left\lceil \frac{n}{6} \right\rceil$
$\mathrm{Var}\, C$	0	0	0	0	0	0
$\mathrm{Var}\, W$	$\frac{15}{16}$	$\frac{7}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{7}{16}$
$\mathrm{Var}\, Zc$	0	0	0	0	0	0
$\mathrm{Var}\, Z$	$\frac{623}{256}$	$\frac{7}{4}$	$\frac{3}{2}$	$\frac{9}{2}$	$\frac{17}{4}$	$\frac{59}{16}$
$\mathrm{Var}\, Y \leq$	$\frac{107}{4}\left\lceil \frac{n}{6} \right\rceil^2$	$\frac{119}{15}\left\lceil \frac{n}{6} \right\rceil^2$	0	0	0	$\frac{121}{15}\left\lceil \frac{n}{6} \right\rceil^2$

4.4 Discussion on Additional Erasures

Results in Table 3 suggest that the algorithm could be reconstructed in a reasonable time, even if the recovered partial key contains additional erasures. In this subsection, we examine scenarios in which some windows are totally erased. This happens in some SCAs. For example, in cache-timing attacks, system interruptions may lead to information loss [8, 47]. In [37], system noise in power or EM traces may cause some windows to be recovered with a low probability.

Let ρ be the probability that a given window fails to be recovered in either p or q. In this context, the recovery of a window does not imply retrieving all bits, but rather refers specifically to the recovery corresponding to our assumption. For each window, there exist three possible cases:

– Both of p and q are recovered with probability $(1-\rho)^2$;
– One of p and q is recovered with probability $2\rho(1-\rho)$;
– Both of p and q fail to be recovered with probability ρ^2.

As additional erasures do not impact the algorithm's global behavior, we only need to re-estimate C, W, Zc and Z.

$$\begin{aligned}\mathrm{E}\,C &= (2^{\Delta}-1)(1-\rho)^2 + (2^{w-b}-1)(2\rho-2\rho^2) \\ &\quad + (2^{w}-1)\rho^2, \\ \mathrm{E}\,W &= 2^{\Delta}(1-\varphi)^{\Lambda}(1-\rho)^2 + 2^{w-b}(2\rho-2\rho^2) + 2^{w}\rho^2.\end{aligned} \tag{9}$$

According to equation (8), EW should be ≤ 1 to avoid an exponential increase in the complexity of the key reconstruction. Given w, b, φ, the expression for $\mathrm{E}\,W$ in (9) depends solely on ρ [8].

Case1: $w = 6$, $b = 4$, and $\varphi = 1/2$.

$$\mathrm{E}\,W = \frac{225}{4}\rho^2 + \frac{15}{2}\rho + \frac{1}{4}.$$

Solving the inequality $\mathrm{E}\,W \leq 1$ yields $-\frac{1}{5} \leq \rho \leq \frac{1}{15}$. Since ρ is a probability that should be in $[0,1]$, we have $\rho \leq \frac{1}{15}$.

Table 4: $\mathrm{E}\,Y$ and $\mathrm{Var}\,Y$ when $w = 6$, $b = 4$, $\varphi = 1/2$ and $\rho = \frac{1}{15}$.

	$\tau \equiv 0$	$\tau \equiv 1$	$\tau \equiv 2$	$\tau \equiv 3$	$\tau \equiv 4$	$\tau \equiv 5$
$\mathrm{E}\,C$	$\frac{49}{15}$	$\frac{343}{225}$	$\frac{49}{75}$	$\frac{49}{75}$	$\frac{49}{75}$	$\frac{343}{225}$
$\mathrm{E}\,W$	1	1	1	1	1	1
$\mathrm{E}\,Zc$	$\frac{1{,}631}{225}$	$\frac{497}{75}$	$\frac{1{,}603}{225}$	$\frac{1{,}561}{225}$	$\frac{1{,}519}{225}$	$\frac{1{,}673}{225}$
$\mathrm{E}\,Z$	$\frac{2{,}653}{900}$	$\frac{301}{75}$	$\frac{1{,}211}{225}$	$\frac{973}{225}$	$\frac{833}{225}$	$\frac{742}{225}$
$\mathrm{E}\,Y \approx$	$4.8\left\lceil\frac{n}{6}\right\rceil^2$	$3.1\left\lceil\frac{n}{6}\right\rceil^2$	$1.8\left\lceil\frac{n}{6}\right\rceil^2$	$1.4\left\lceil\frac{n}{6}\right\rceil^2$	$1.2\left\lceil\frac{n}{6}\right\rceil^2$	$2.5\left\lceil\frac{n}{6}\right\rceil^2$
$\mathrm{Var}\,C$	$\frac{3{,}584}{225}$	$\frac{876{,}176}{50{,}625}$	$\frac{103{,}124}{5{,}625}$	$\frac{103{,}124}{5{,}625}$	$\frac{103{,}124}{5{,}625}$	$\frac{876{,}176}{50{,}625}$
$\mathrm{Var}\,W$	$\frac{301}{15}$	$\frac{4{,}417}{225}$	$\frac{1{,}456}{75}$	$\frac{1{,}456}{75}$	$\frac{1{,}456}{75}$	$\frac{4{,}417}{225}$
$\mathrm{Var}\,Zc$	$\frac{702{,}464}{50{,}625}$	$\frac{29{,}372}{1{,}875}$	$\frac{1{,}202{,}616}{50{,}625}$	$\frac{1{,}004{,}654}{50{,}625}$	$\frac{57{,}736}{3{,}375}$	$\frac{724{,}346}{50{,}625}$
$\mathrm{Var}\,Z$	$\frac{16{,}907{,}891}{8{,}100{,}000}$	$\frac{120{,}799}{5{,}625}$	$\frac{1{,}489{,}754}{50{,}625}$	$\frac{1{,}502{,}396}{50{,}625}$	$\frac{1{,}348{,}886}{50{,}625}$	$\frac{1{,}197{,}686}{50{,}625}$
$\mathrm{Var}\,Y \approx$	$47.5\left\lceil\frac{n}{5}\right\rceil^3$	$40\left\lceil\frac{n}{5}\right\rceil^3$	$30.6\left\lceil\frac{n}{5}\right\rceil^3$	$19.8\left\lceil\frac{n}{5}\right\rceil^3$	$14.5\left\lceil\frac{n}{5}\right\rceil^3$	$11.5\left\lceil\frac{n}{5}\right\rceil^3$

Table 4 presents the estimation of $\mathrm{E}\,Y$ and $\mathrm{Var}\,Y$ computed from equations (6) and (7), accounting for $\frac{1}{15}\%$ unrecovered windows (equivalent to additional erasures). Here, $\approx$ is used instead of $<$ and $\leq$, as we discard lower-order polynomial terms that have a negligible impact on overall expression. Comparing Table 4 with Table 3, the expected number of partial keys verified in the reconstruction scales quadratically with n (as opposed to linearly), due to the introduced erasures. It is observed that the speed advantage in misaligned cases persists

[8] $\mathrm{E}\,W$ is independent of Δ and Λ if $\varphi = \frac{1}{2}$.

even when the introduced erasures approach the algorithm's maximum tolerance. Moreover, the variance of Y indicates that the aligned case is more prone to generating candidates with larger search spaces, which leads to a longer time for key reconstruction in practice.

Case2: $w = 4$, $b = 2$, *and* $\varphi = 1/2$.

$$\mathrm{E}\, W = 9\rho^2 + 6\rho + 1.$$

Solving the inequality $\mathrm{E}\, W \leq 1$ yields $\rho = 0$. This is consistent with the results of [25] and [40], which show that revealing 50% of the bits of p and q is necessary to prevent the algorithm's complexity from growing exponentially.

In this case, $\frac{b}{w} = 50\%$ indicates that the key reconstruction cannot tolerate additional erasures. However, $\tau = 2$ in 5 suggests that allowing some additional erasures may be feasible in practice. When $\tau = 2$, the reconstruction process verifies at most n correct partial keys. In practical scenarios, although an additional 25-bit erasure would expand the candidate solution space to 2^{25}, the computational cost remains well within the means of modestly resourced adversaries [17]. For 1024-bit RSA($w = 4$), an additional 25-bit erasure corresponds to approximately 10% windows remaining unrecovered.

Table 5: $\mathrm{E}\, Y$ when $w = 4$, $b = 2$, and $\varphi = 1/2$.

	$\tau \equiv 0$	$\tau \equiv 1$	$\tau \equiv 2$	$\tau \equiv 3$
$\mathrm{E}\, C$	3	1	0	1
$\mathrm{E}\, W$	1	1	1	1
$\mathrm{E}\, Zc$	5	4	4	5
$\mathrm{E}\, Z$	$\frac{9}{4}$	3	4	3
$\mathrm{E}\, Y <$	$\frac{27}{8}\left\lceil \frac{n}{4} \right\rceil^2 + \frac{27}{8}\left\lceil \frac{n}{4} \right\rceil$	$\frac{3}{2}\left\lceil \frac{n}{4} \right\rceil^2 + \frac{5}{2}\left\lceil \frac{n}{4} \right\rceil$	$4\left\lceil \frac{n}{4} \right\rceil$	$\frac{3}{2}\left\lceil \frac{n}{4} \right\rceil^2 + \frac{7}{2}\left\lceil \frac{n}{4} \right\rceil$

In sum, misaligned cases outperform the aligned case in both efficiency and tolerance to additional erasures. Although it is possible in SCAs that the recovered partial key contains errors, we omit this discussion in this work. According to the result presented in [32], the success rate is as low as 4% when p and q contain 1% errors and 25% erasures. In the absence of more efficient error-correction algorithms, the conventional approach treats potential errors as erasures in subsequent processing.

5 Experiment Evaluation

This section illustrates the practical implications of our proposed model from the following perspectives.

– Statistical analysis of the distribution of aligned and misaligned scenarios confirms that they exhibit non-negligible occurrence probabilities.

– SPA attacks are conducted on the target implementation, successfully recovering $\tau(x-1)$ and the lower 4 bits within each 6-bit window, empirically validating the practical implications of our leakage assumption.
– An evaluation of key reconstruction is performed between theoretical predictions and practical outcomes.

5.1 Experimental Setup

The experiments in Sections 5.2 and 5.4 were performed on an AMAX server equipped with 64 CPU cores and 503 GB of RAM. A Python module `pyOpenSSL` is used to generate the key pairs of RSA. Leveraging multiprocess parallel computing, each experiment was completed in minutes to tens of minutes.

The experiments in Section 5.3 were carried out on the ChipWhisperer CW308 platform with an STM32F303 target board. The STM32F303 features an ARM Cortex-M4 core operating at 7.38 MHz. Power traces were recorded using CW-Lite at a sampling rate of 4×7.38 MS/s (4 samples per clock cycle) with 10-bit resolution. The trace processing was performed on a ThinkPad X1 laptop running Ubuntu 20.04, equipped with an Intel i7-8550U CPU (1.80 GHz) and 8 GB of RAM. We targeted the RSA implementation in the current version of OpenSSL, using an additional trigger signal to facilitate trace recording. Our analysis focuses on two functions:

– `BN_is_bit_set()`, invoked by `ossl_bn_miller_rabin_is_prime()` in the source file `bn_prime.c`.
– `MOD_EXP_CTIME_COPY_FROM_PREBUF()`, called by `bn_mod_exp_mont_fixed_top()` in the source file `bn_exp.c`.

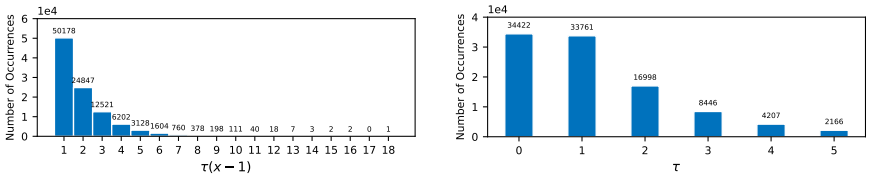
5.2 The Distribution of Aligned and Misaligned Cases

To demonstrate that the misaligned cases occur with non-negligible probability, we conducted the statistical analysis by repeatedly invoking OpenSSL to generate the private key set, from which we extracted p and q.

This test was performed 100,000 times for 1024-, 2048-, and 4096-bit RSA, respectively. We recorded the value of $\tau(p-1)$ and $\tau(q-1)$. Due to the symmetric roles of p and q in this context, the distributions of $\tau(p-1)$ and $\tau(q-1)$ are identical. Figure 4.(a) presents the distributions of $\tau(x-1)$ for 2048-bit RSA, where x represents either prime. Notably, $\tau(x-1)$ consistently takes relatively small values. Figure 4.(b) shows the distributions for the proposed scenarios. As noted previously, these scenarios can be classified by the value of Δ due to the similarity of the computational complexity. Specifically:

– Class 1 ($\tau \equiv 0$): 34,422;
– Class 2 ($\tau \equiv 1, 5$): 35,927;
– Class 3 ($\tau \equiv 2, 3, 4$): 29,651;

Each class occurs with non-negligible probability. Similar behavior is observed for both RSA-1024 and RSA-4096.



(a) (b)

Fig. 4: (a) and (b) represent the distributions of $\tau(x-1)$ and τ, respectively, for RSA-2048 over 100,000 trials.

5.3 Recovering $\tau(x-1)$ and `idx[3:0]` via SPA Attacks

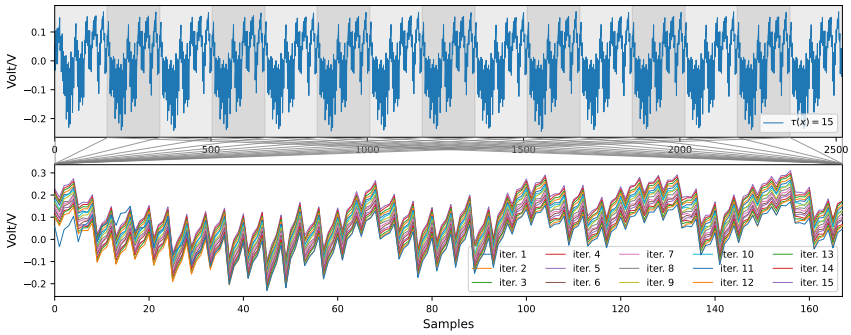
Recovering $\tau(x-1)$ To map the information recovered by SCAs back to p and q, the values of $\tau(p-1)$ and $\tau(q-1)$ need to be known. In OpenSSL, the Miller-Rabin primality test calculates $\tau(x-1)$ and reduces x to x' before invoking modular exponentiation.

Listing 2 is a code snippet illustrating the process of calculating $\tau(x-1)$ in OpenSSL. The implementation repeatedly calls `BN_is_bit_set()`, which tests whether the `a`-th bit in `w1` is set to 1. Since this operation is not constant-time, the exact number of function calls, which is $\tau(x-1)$, can be leaked through the power channel.

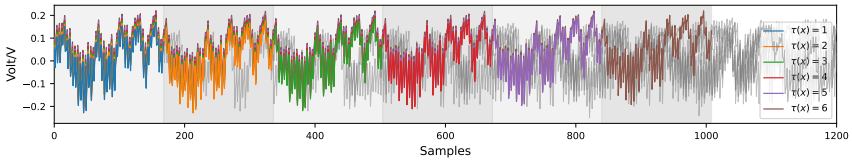
```
1 int ossl_bn_miller_rabin_is_prime(const BIGNUM *w, int
      iterations, BN_CTX *ctx, BN_GENCB *cb, int enhanced,
      int *status)
2 {
3     ...
4     /* (Step 1) Calculate largest integer 'a' such that
      2^a divides w-1 */
5     a = 1;
6     while (!BN_is_bit_set(w1, a))
7         a++;
8     /* (Step 2) m = (w-1) / 2^a */
9     if (!BN_rshift(m, w1, a))
10        goto err;
11    ...
12 }
```

Listing 2: Computing $\tau(x-1)$ in OpenSSL.

In function `ossl_bn_miller_rabin_is_prime()`, the variable `w` holds the candidate prime to be tested. line 6–7 in Listing 2 sequentially check each bit of `w-1` starting from the second least significant bit to count the number of trailing zeros. This operation exhibits two properties that introduce a distinct timing channel exploitable via SPA: (1) The execution time of this operation depends on the value of $\tau(x-1)$; (2) The execution time for each iteration is nearly constant.



(a)



(b)

Fig. 5: Determining the value of $\tau(x-1)$ via SPA.

Our experimental analysis successfully identified the power trace corresponding to a single iteration, enabling a precise determination of the iteration count through visual inspection of the power traces. Figure 5.(a) displays the power trace when $\tau(x-1) = 15$. Each iteration is distinguishable, separated by alternating gray and white intervals. Although all subsequent intervals contain the 168 samples, the first interval initially contains a reduced count due to missing control instructions in the initial loop. To ensure consistent analysis, we padded the first iteration's samples to 168 by borrowing preceding data points. This preprocessing explains the visible deviation in the blue trace (Bottom of Figure 5.(a)) compared to the others. To enhance visibility, we deliberately offset the traces vertically along the y-axis, as well as in Figure 5.(b). Notably, the power trace of all subsequent iterations exhibits highly consistent patterns, enabling a reliable identification of the iteration count through visual inspection.

Figure 5.(b) presents a comparison of power traces for $\tau(x-1) \equiv 1$ to 6. The colored traces represent the execution of the loop body, whereas the gray traces correspond to unrelated operations. These colored traces overlap at the execution of each iteration. A straightforward method to determine the number of iterations involves cross-correlating a reference single-loop trace (or a reference trace that contains 30 loops) with the target trace, and the number of overlapping segments directly yields the value of $\tau(x-1)$.

Moreover, Figure 4 indicates $\tau(p-1)$ and $\tau(q-1)$ are typically small. Hence, an exhaustive search for possible values of these two parameters may also lead to successful key reconstruction.

Recovering Partial Bits of p and q In this subsection, we conducted an SPA attack on the fixed-window modular exponentiation implemented in OpenSSL to validate the practical implications of our leakage assumption.

Target function: Listing 3 presents the core implementation of fixed-window modular exponentiation in OpenSSL, illustrating the complete operation in each window. The computation proceeds as follows:

- Line 4-5 perform the squaring.
- Line 6-7 extract the current window of bits from the exponent.
- Line 9 retrieves the precomputed multiplier corresponding to the bits extracted in line 7.
- Line 10 multiplies the intermediate result by the retrieved multiplier.

```
1 int bn_mod_exp_mont_fixed_top(BIGNUM *rr, const BIGNUM *a
      , const BIGNUM *p, const BIGNUM *m, BN_CTX *ctx,
      BN_MONT_CTX *in_mont)
2 {
3     ...
4     for (i = 0; i < window; i++)
5         bn_mul_mont_fixed_top(&tmp, &tmp, &tmp, mont,ctx)
6     bits -= window;
7     wvalue = bn_get_bits(p, bits) & wmask;
8     /* Fetch the appropriate  precomputed value */
9     MOD_EXP_CTIME_COPY_FROM_PREBUF(&am, top, powerbuf,
      wvalue,window)
10    bn_mul_mont_fixed_top(&tmp, &tmp, &am, mont, ctx)
11    ...
12 }
```

Listing 3: Fixed-window modular exponentiation in the current version of OpenSSL.

Listing 4 defines the target function `MOD_EXP_CTIME_COPY_FROM_PREBUF()`. For a window size of 6, `buf` holds 64 precomputed multipliers, and their arrangement can be visualized as a 4×16 table. The variable `idx` (or `wvalue`) acts as the index for selecting the multiplier to read, where its value represents the partial secret. To protect `idx` from potential attacks, this function employs a dummy load that sequentially reads all 64 precomputed multipliers and selects the correct one via the masking technique. The mask is computed as: `(0 - constant_time_eq_int()&1)`. This expression yields **0** (all zeros) or $-\mathbf{1}$ (all ones, in two's complement).

In line 7-8, `idx` is decomposed into: (1) the upper 2 bits (stored in `i`) as the row index; (2) the lower 4 bits for (stored in `idx`) as the column index. Then line 10–13 make the selection of the row, where the mask equals to all ones only when the row index matches the upper 2 bits of `idx`. Similarly, in line 15-22, the mask outputs all ones only when the loop counter matches the lower 4 bits of `idx` and **0** otherwise. By distinguishing whether the mask is **0** or all ones, we

can recover secret bits in each window. Our experiments focus on line 15-22 in Listing 4, which handle the extraction of 4 bits per window(`idx[3:0]`), rather than retrieving all bits in each 6-bit window. This is because the computation in line 10-13 is too brief to reveal detectable leaks through SPA.

```
1 int MOD_EXP_CTIME_COPY_FROM_PREBUF(BIGNUM *b, int top,
      unsigned char *buf, int idx, int window)
2 {
3     ...
4     int width = 1 << window;
5     int xstride = 1 << (window - 2);
6     BN_ULONG y0, y1, y2, y3;
7     i = idx >> (window - 2);
8     idx &= xstride - 1;
9
10    y0 = (BN_ULONG)0 - (constant_time_eq_int(i,0)&1);
11    y1 = (BN_ULONG)0 - (constant_time_eq_int(i,1)&1);
12    y2 = (BN_ULONG)0 - (constant_time_eq_int(i,2)&1);
13    y3 = (BN_ULONG)0 - (constant_time_eq_int(i,3)&1);
14
15    for (i = 0; i < top; i++, table += width) {
16        BN_ULONG acc = 0;
17        for (j = 0; j < xstride; j++) {
18                acc |= ( (table[j + 0 * xstride] & y0) |
19                         (table[j + 1 * xstride] & y1) |
20                         (table[j + 2 * xstride] & y2) |
21                         (table[j + 3 * xstride] & y3) )
22                       & ((BN_ULONG)0 - (
      constant_time_eq_int(j,idx)&1));}
23        b->d[i] = acc;}
24    ...
25 }
```

Listing 4: Implementation of fetching the precomputed multiplier.

The top of Figure 6 shows a trace fragment for `MOD_EXP_CTIME_COPY_FROM_PREBUF()`, extracted from the complete power trace, where `idx[3:0]`$= 0$. The target function performs 16 iterations, line 22 in Listing 4 compares `idx[3:0]` with `j`, where `j` is range from 0 to 15. `(0 - constant_time_eq_int()&1)` outputs all ones when `idx[3:0]`$=$`j`. We highlighted the 16 iterations in the target function using alternating gray and white backgrounds, and a distinct downward dip can be observed in the first block, indicating `idx[3:0]`$= 0$. The middle plot of Figure 6 overlays the traces of all 16 iterations, where the red line represents `idx[3:0]`$= 0$, while the remaining 15 lines are shown in gray. Zooming in (bottom plot), the red line is visually distinct from the gray lines. Similar results are observed when `idx[3:0]` takes values from 1 to 15.

Figure 7 shows the trace fragments extracted from the power trace, with each subplot representing the sensitive operations in `MOD_EXP_CTIME_COPY_FROM_PREBUF()`. The subplots are sorted by `idx[3:0]`, ranging from 0 to 15. Red

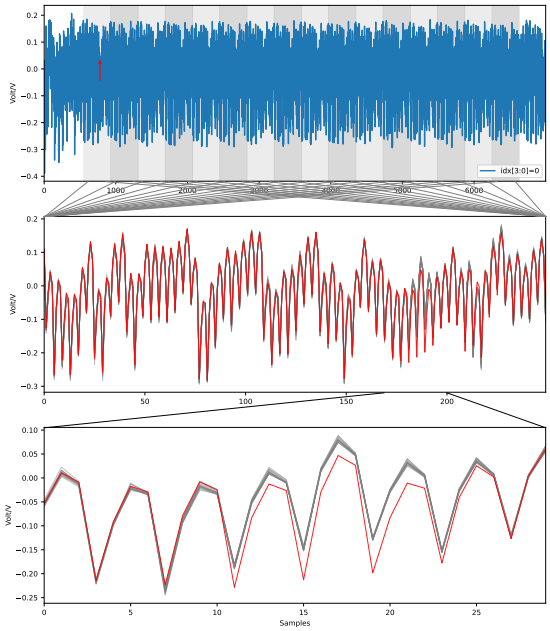


Fig. 6: The power traces of the sensitive operations in `MOD_EXP_CTIME_COPY_FROM_PREBUF()`

arrows in the subplots mark leakage locations. It can be observed that the values of `idx[3:0]` corresponding to different traces can be directly distinguished through visual inspection.

In sum, we could recover the lower 4 bits of each window by launching SPA attack on an ARM Cortex-M4 device, demonstrating power-based leakage. It should be noted that the upper 2 bits of `idx` are more difficult to obtain directly through visual observation.

5.4 An Evaluation of Key Reconstruction

Figure 8 displays the distributions of incorrect partial solutions observed in 100,000 trials for RSA-2048 and RSA-4096. The green line represents cases where $\tau \equiv 2$, 3 and 4, none of which produced incorrect solutions. These scenarios occur 29,651 times for RSA-2048 and 29,545 times for RSA-4096, respectively. We adopted non-uniform intervals in this figure since the number of incorrect solutions in the remaining two classes varied significantly, ranging from hundreds to hundreds of thousands. For example, when $\tau \equiv 0$, the false solutions in Figure 8.(b) varied between $1,849$ to $111,723$. As the interval width increases, the probability that the number of incorrect solutions falls within a given range drops rapidly.[9] The blue and orange lines demonstrate that most the results cluster

[9] Interestingly, both of the figures have an unexplained increase in the interval $[5,000-6,000]$. We skip it here because of its minimal influence on the overall analysis. The

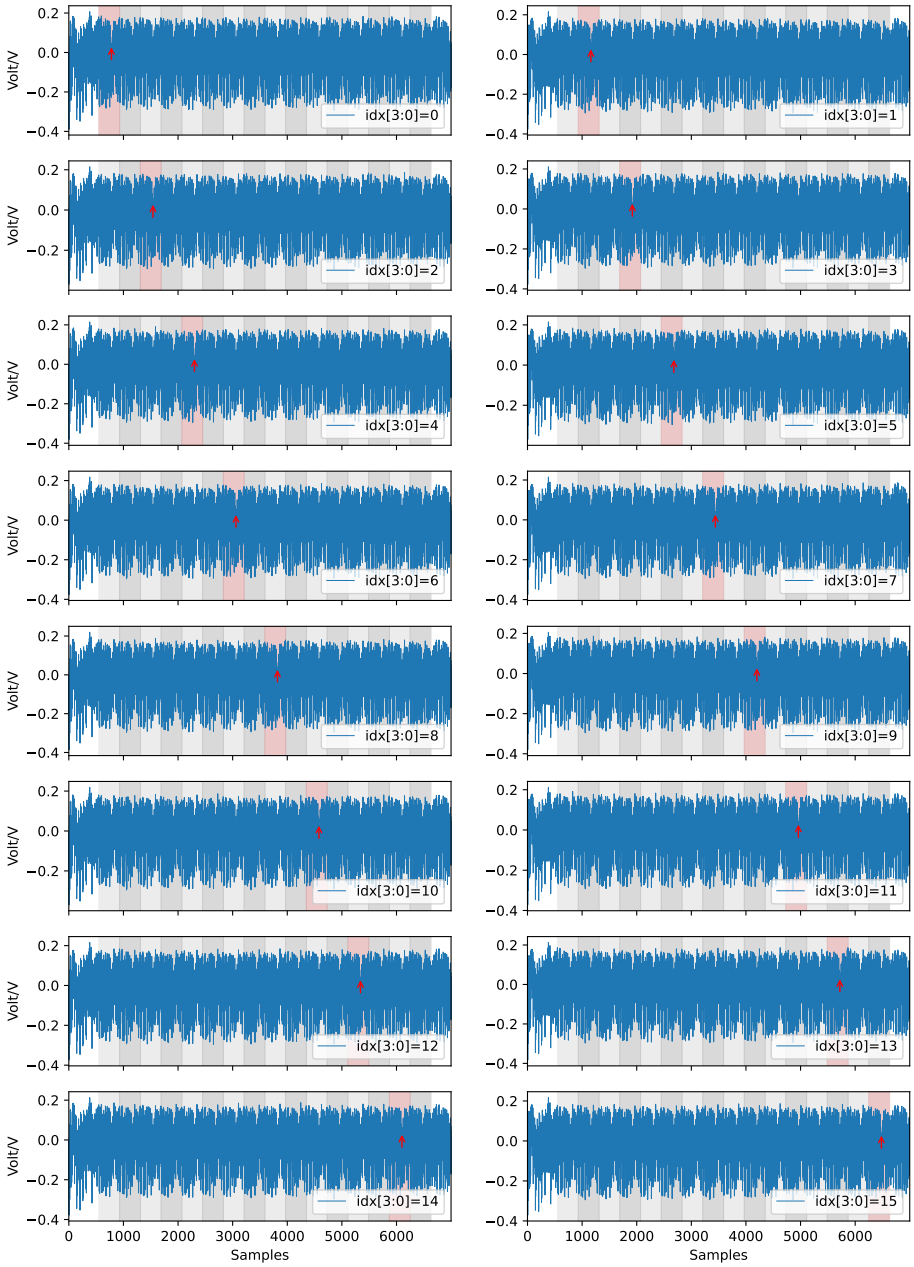
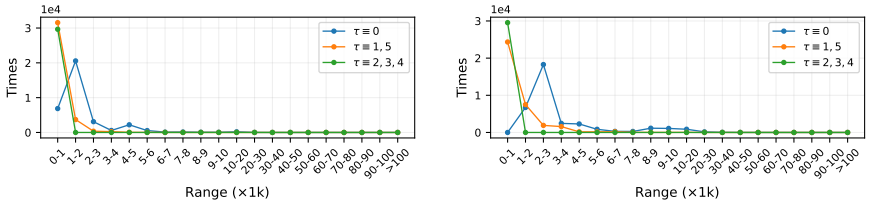


Fig. 7: The power traces of the sensitive operations in `MOD_EXP_CTIME_COPY_FROM_PREBUF()`, where `idx[3:0]` takes values from 1 to 15.



(a) RSA-2048 (b) RSA-4096

Fig. 8: The distributions of incorrect partial solutions examined in 100,000 experiments for RSA-2048 and RSA-4096.

around the expected value, with only a few trails significantly exceeding it. This observed distribution aligns well with our theoretical predictions.

Table 6 compares the average number of incorrect partial solutions examined in theoretical estimates and experimental results. For both RSA-2048 and RSA-4096, the first row corresponds to the values derived from table 3, representing our estimated count of false branches generated during key reconstruction when $\varphi = 1/2$. The second row demonstrates the average number of false branches counted in our experiments. The observed discrepancies between theoretical predictions and empirical results may arise from inaccuracies in estimating φ, which implies a potential relationship between φ and the uncertainty in the key reconstruction. Furthermore, the third row presents the observed proportions of the three scenarios among 100,000 trials. In summary, the estimated complexity of the key reconstruction is consistent with results observed in practical attacks.

Table 6: Comparison of the average number of incorrect partial solutions examined in theoretical estimates and experimental results.

		$\tau \equiv 0$	$\tau \equiv 1, 5$	$\tau \equiv 2, 3, 4$
RSA-2048	Theoretical	1,579	683	0
	Practical	1,739	640	0
	Occurrence	34.4%	35.9%	29.7%
RSA-4096	Theoretical	3,158	1,366	0
	Practical	3,484	1,278	0
	Occurrence	34.5%	36%	29.5%

As shown in Table 6, the difference in average complexity between the alignment and misalignment cases seems trivial, since hundreds of false candidates can be examined in seconds on a modern computer. However, in scenarios that

increase in $[10,000 - 20,000]$ is reasonable since the interval is enlarged by a factor of 10.

generate extensive candidates, especially in RSA-4096, key reconstruction may take several hours. This occurs even as $\frac{4}{6} \approx 66.7\%$ bits are recovered. With additional erasure bits being introduced, the reconstruction time increases significantly, demonstrating the efficiency advantages of our misaligned scenarios.

6 Conclusion

In this work, we demonstrated that the complexity of the branch-and-prune algorithm is closely related to observed leakage patterns, and identify a new vulnerability in OpenSSL's implementation of the Miller-Rabin primality test. Our proposed leakage model, termed misaligned scenarios, leads to a reduced uncertainty in the key reconstruction process, thereby enhancing its practical efficiency. These scenarios originate from two key factors:

- The input of modular exponentiation in the Miller-Rabin primality test is p' (or q'), which is derived from $p-1$ (or $q-1$) by truncating the least significant zero bits.
- To prevent high-bit leakage, OpenSSL pads the exponent with leading zeros before performing modular exponentiation.

When recovering p' and q' with a fixed pattern (extracting b bits in each window), mapping these bits back to p and q reveals new leakage patterns. A key observation is that when either $p[i]$ or $q[i]]$ is determined, the reconstruction process produces no false candidates. Our theoretical analysis demonstrates: (1) significantly reduced computational complexity in these misaligned scenarios and (2) their occurrence with non-negligible probability. By introducing additional erasure bits, we further investigated the limits of the branch-and-prune algorithm in the proposed scenarios. Furthermore, experimental results validate both the existence and practical implications of these leakage patterns.

References

1. Aciiçmez, O.: Yet another microarchitectural attack: Exploiting I-Cache. In: Proceedings of the 2007 ACM workshop on Computer Security Architecture. pp. 11–18. ACM Press, New York, NY, USA (2007). https://doi.org/10.1145/1314466.1314469
2. Aciiçmez, O., Koç, Ç.K., Seifert, J.P.: On the power of simple branch prediction analysis. In: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security. pp. 312–320. ACM Press, New York, NY, USA (2007). https://doi.org/10.1145/1229285.1266999
3. Aciiçmez, O., Gueron, S., Seifert, J.P.: New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures. In: Galbraith, S.D. (ed.) 11th IMA International Conference on Cryptography and Coding. LNCS, vol. 4887, pp. 185–203. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77272-9_12

4. Aciiçmez, O., Schindler, W.: A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on OpenSSL. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 256–273. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79263-5_16
5. Alam, M., Khan, H.A., Dey, M., Sinha, N., Callan, R.L., Zajic, A.G., Prvulovic, M.: One&done: A single-decryption EM-based attack on OpenSSL's constant-time blinded RSA. In: Enck, W., Felt, A.P. (eds.) USENIX Security Symposium 2018. pp. 585–602. USENIX Association (2018)
6. Aldaya, A.C., Brumley, B.B.: When one vulnerable primitive turns viral: Novel single-trace attacks on ECDSA and RSA. IACR TCHES **2020**(2), 196–221. (2020). https://doi.org/10.13154/tches.v2020.i2.196-221
7. Aldaya, A.C., Brumley, B.B., ul Hassan, S. Pereida García C., Tuveri, N.: Port Contention for Fun and Profit. In: 2019 IEEE Symposium on Security and Privacy. pp. 870–887. IEEE (2019). https://doi.org/10.1109/SP.2019.00066
8. Aldaya, A.C., García, C.P., Tapia, L.M.A., Brumley, B.B.: Cache-timing attacks on RSA key generation. IACR TCHES **2019**(4), 213–242. (2019). https://doi.org/10.13154/tches.v2019.i4.213-242
9. Bernstein, D.J., Breitner, J., Genkin, D., Bruinderink, L.G., Heninger, N., Lange, T., van Vredendaal, C., Yarom, Y.: Sliding right into disaster: Left-to-right sliding windows leak. In: Fischer, W., Homma, N. (eds) CHES 2017. LNCS, vol. 10529, pp. 555–576. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-66787-4_27
10. Blömer, J., May, A.: New partial key exposure attacks on RSA. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 27–43. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_2
11. Boneh, D., Durfee, G., Frankel, Y.: An attack on RSA given a small fraction of the private key bits. In: Ohta, K., Pei, D. (eds.) ASIACRYPT'98. LNCS, vol. 1514, pp. 25–34. Springer, Heidelberg (1998). https://doi.org/10.1007/978-3-540-45146-4_2
12. Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., Sadeghi, A.R.: Software grand exposure: SGX cache attacks are practical. In: Enck, W., Mulliner, C. (eds.) 11th USENIX Workshop on Offensive Technologies. USENIX Association, Vancouver, BC, Canada (2017)
13. Brickell, E.: A vision for platform security (invited talk). In: Oswald, E., Rohatgi, P. (eds) CHES 2008. LNCS, vol. 5154, pp. 444. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_29
14. Brickell, E. Graunke, G., Neve, M., Seifert, J.P.: Software mitigations to hedge AES against cache-based software side channel vulnerabilities. IACR Cryptology ePrint Archive, 2006:52 (2006). http://eprint.iacr.org/2006/052
15. Briongos, S., Malagón, P., Moya, J.M., Eisenbarth, T.: RELOAD+REFRESH: Abusing cache replacement policies to perform stealthy cache attacks. In: Capkun, S., Roesner, F. (eds.) USENIX Security Symposium 2020. pp. 1967–1984. USENIX Association (2020)
16. Brumley, D., Boneh, D.: Remote timing attacks are practical. In: USENIX Security Symposium 2003. USENIX Association (2003)
17. Chuengsatiansup, C., Feutrill, A., Sim, R.Q., Yarom, Y.: RSA key recovery from digit equivalence information. In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. pp. 193-211. Springer(2022). https://doi.org/10.1007/978-3-031-09234-3_10
18. Cohney, S., Kwong, A., Paz, S., Genkin, D., Heninger, N., Ronen, E., Yarom, Y.: Pseudorandom black swans: Cache attacks on CTR_DRBG. In: 2020 IEEE Symposium on Security and Privacy. pp. 1241–1258. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00046

19. Cohney, S.N., Green, M.D., Heninger, N.: Practical state recovery attacks against legacy RNG implementations. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 265–280. ACM Press (2018). https://doi.org/10.1145/3243734.3243756
20. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology **10**(4), 233–260 (1997). https://doi.org/10.1007/s001459900030
21. Gras, B., Razavi, K., Bos, H., Giuffrida, C.: Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In: Enck, W., Felt, A.P. (eds.) USENIX Security Symposium 2018. pp. 955–972. USENIX Association (2018)
22. Guo, Y., Zigerelli, A., Zhang, Y., Yang, J.: Adversarial prefetch: New cross-core cache side channel attacks. In: 2022 IEEE Symposium on Security and Privacy. pp. 1458–1473. IEEE (2020). https://doi.org/10.1109/SP46214.2022.9833692
23. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: Cold boot attacks on encryption keys. In: van Oorschot, P.C. (ed.) USENIX Security Symposium 2008. pp. 45–60. USENIX Association (2008)
24. Henecka, W., May, A., Meurer, A.: Correcting errors in RSA private keys. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 351–369. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_19
25. Heninger, N., Shacham, H.: Reconstructing RSA private keys from random key bits. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 1–17. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_1
26. Herrmann, M., May, A.: Solving linear equations modulo divisors: On factoring given any bits. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 406–424. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_25
27. Howgrave-Graham, N.: Finding small roots of univariate modular equations revisited. In: Darnell, M. (ed.) 6th IMA International Conference on Cryptography and Coding. LNCS, vol. 1355, pp. 131–142. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0024458
28. Huo, T., Meng, X., Wang, W., Hao, C., Zhao, P., Zhai, J., Li, M.: Bluethunder: A 2-level directional predictor based side-channel attack against sgx. IACR TCHES **2020**(1), 321–347. (2019). https://doi.org/10.13154/tches.v2020.i1.321-347
29. Hu, X., Meunier, Q. L., Encrenaz, E.: Blind-Folded: Simple power analysis attacks using data with a single trace and no training. IACR TCHES **2025**(1), 475-496. (2025). https://doi.org/10.46586/tches.v2025.i1.475-496
30. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
31. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
32. Kunihiro, N., Shinohara, N., Izu, T.: Recovering RSA secret keys from noisy key bits with erasures and errors. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778 pp. 180–197. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_12
33. Lee, S., Shih, M.W., Gera, P., Kim, T., Kim, H., Peinado, M.: Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In: Kirda, E., Ristenpart, T. (eds.) USENIX Security Symposium 2017. pp. 557–574. USENIX Association (2017)

34. Li, M., Zhang, Y., Wang, H., Li, K., Cheng, Y.: CIPHERLEAKS: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security Symposium 2021. pp. 717–732. USENIX Association (2021)
35. Lipp, M., Kogler, A., Oswald, D.F., Schwarz, M., Easdon, C., Canella, C., Gruss, D.: PLATYPUS: Software-based power side-channel attacks on x86. In: 2021 IEEE Symposium on Security and Privacy. pp. 355–371. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00063
36. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: 2015 IEEE Symposium on Security and Privacy. pp. 605–622. IEEE (2015). https://doi.org/10.1109/SP.2015.43
37. Mangard, S., Oswald, E., Popp, T.:Power analysis attacks: Revealing the secrets of smart cards. Springer Science & Business Media (2008)
38. Mantel, H., Schickel, J., Weber, A., Weber, F.: How secure is green IT? The case of software-based energy side channels. In: López, J., Zhou, J., Soriano, M. (eds.) ESORICS 2018, Part I. LNCS, vol. 11098, pp. 218–239. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-99073-6_11
39. Moghimi, D., Van Bulck, J., Heninger, N., Piessens, F., Sunar, B.: CopyCat: Controlled instruction-level attacks on enclaves. In: Capkun, S., Roesner, F. (eds.) USENIX Security Symposium 2020. pp. 469–486. USENIX Association (2020)
40. Paterson, K.G., Polychroniadou, A., Sibborn, D.L.: A coding-theoretic approach to recovering noisy RSA keys. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 386–403. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_24
41. Percival, C.: Cache missing for fun and profit (2005). http://css.csail.mit.edu/6.858/2014/readings/ht-cache.pdf
42. Rivest, R.L., Shamir, A.: Efficient factoring based on partial information. In: Pichler, F. (ed.) EUROCRYPT 1985. LNCS, vol. 219, pp. 31–34. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39805-8_3
43. Schwarz, M., Weiser, S., Gruss, D., Maurice, C., Mangard, S.: Malware guard extension: Using sgx to conceal cache attacks. In: Polychronakis, M., Meier, M. (eds.) 14th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. LNCS, vol. 10327, pp. 3–24. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-60876-1_1
44. Saito K., Ito A., Ueno R., Homma N.: One truth prevails: A deep-learning based single-trace power analysis on RSA–CRT with windowed exponentiation. IACR TCHES **2020**(4): 490-526. (2022). https://doi.org/10.46586/tches.v2022.i4.490-526
45. Shinde, S., Chua, Z.L., Narayanan, V., Saxena, P.: Preventing page faults from telling your secrets. In: Chen, X., Wang, X., Huang, X. (eds.) ASIACCS 2016. pp. 317–328. ACM Press, Xi'an, China (2016). https://doi.org/doi.org/10.1145/2897845.2897885
46. Weiser, S., Spreitzer, R., Bodner, L.: Single trace attack against RSA key generation in intel SGX SSL. In: Kim, J., Ahn, G.J., Kim, S., Kim, Y., López, J., Kim, T. (eds.) ASIACCS 2018. pp. 575–586. ACM Press, Incheon, Republic of Korea (2018). https://doi.org/10.1145/3196494.3196524
47. Yarom, Y., Falkner, K.: FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In: Fu, K., Jung, J. (eds.) USENIX Security Symposium 2014. pp. 719–732. USENIX Association (2014)

48. Yarom, Y., Genkin, D., Heninger, N.: CacheBleed: A timing attack on OpenSSL constant time RSA. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 346–367. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_17
49. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-VM side channels and their use to extract private keys. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012. pp. 305–316. ACM Press (2012). https://doi.org/10.1145/2382196.2382230

A Computing the Expectation and Variance of $Y_\imath$

This appendix presents the derivation of the expectation and variance of the number of incorrect keys generated at each window during the key reconstruction.

Expectation of $Y_\imath$ Let $G_{Y_\imath}(s)$ be the PGF for the $Y_\imath$. In equation (2), $X_{\imath-1}$ and $\{Z_\jmath\}$ are independent random variables. Since $\{Z_\jmath\}$ are identically distributed, each with PGF $G_Z(s)$, according to Theorem 2, a PGF for $Z_1 + ... + Z_{X_{\imath-1}}$ can be given by function composition $G_{X_{\imath-1}}(G_Z(s))$, where $G_{X_{\imath-1}}(s)$ is the PGF of $X_{\imath-1}$. Denote the PGF of Zc as $G_{Zc}(s)$, then

$$G_{Y_\imath}(s) = G_{X_{\imath-1}}(G_Z(s))G_{Zc}(s).$$

Differentiating above equation gives $G'_{Y_\imath}(s) =$

$$G'_{X_{\imath-1}}(G_Z(s))G'_Z(s)G_{Zc}(s) + G_{X_{\imath-1}}(G_Z(s))G'_{Zc}(s). \tag{10}$$

$G'_{Y_\imath}(s)$ at $s = 1$ is exactly the expectation of $Y_\imath$. With the fact that $G_Z(1) = G_{X_{\imath-1}}(1) = G_{Zc}(1) = 1$, we obtain

$$G'_{Y_\imath}(1) = G'_{X_{\imath-1}}(1)G'_Z(1) + G'_{Zc}(1). \tag{11}$$

Let G_W and G_C be the PGFs for independent random variables W and C, respectively. We have $G_{X_\imath}(s)$ satisfying the recurrence

$$G_{X_\imath}(s) = G_{X_{\imath-1}}(G_W(s))G_C(s).$$

Similarly,

$$G'_{X_\imath}(1) = G'_{X_{\imath-1}}(1)G'_W(1) + G'_C(1). \tag{12}$$

Supposing that the algorithm lifts from window $\imath = 1$ and we know that the partial solution obtained from **Block 2** is error-free, it follows $X_1 = C$. Rewriting (12) as a geometric sequence yields

$$G'_{X_\imath}(1) = \mathrm{E}\, X_\imath = \frac{\mathrm{E}\, C}{1 - \mathrm{E}\, W}(1 - (\mathrm{E}\, W)^\imath).$$

Substituting $G'_{X_{\imath-1}}(1)$ in (11) gives

$$G'_{Y_\imath}(1) = \mathrm{E}\, Y_\imath = \frac{\mathrm{E}\, C\, \mathrm{E}\, Z}{1 - \mathrm{E}\, W}(1 - (\mathrm{E}\, W)^{\imath-1}) + \mathrm{E}\, Zc.$$

Variance of $Y_\imath$ As $\operatorname{Var} Y_\imath = G''_{Y_\imath}(1) + G'_{Y_\imath}(1) - (G'_{Y_\imath}(1))^2$, we first obtain the second derivative of $G_{Y_\imath}(1)$ by differentiating (10). Then

$$\begin{aligned} G''_{Y_\imath}(1) =& G''_{X_{\imath-1}}(1)(G'_Z(1))^2 + G'_{X_{\imath-1}}(1)G''_Z(1) \\ &+ 2G'_{X_{\imath-1}}(1)G'_Z(1)G'_{Zc}(1) + G''_{Zc}(1). \end{aligned}$$

Substituting $G''_{Y_\imath}(1)$ and $G'_{Y_\imath}(1)$, we obtain the following expression based on the properties of PGF.

$$\operatorname{Var} Y_\imath = \operatorname{Var} X_{\imath-1}(\operatorname{E} Z)^2 + \operatorname{E} X_{\imath-1}\operatorname{Var} Z + \operatorname{Var} Zc.$$

Similarly,

$$\operatorname{Var} X_\imath = \operatorname{Var} X_{\imath-1}(\operatorname{E} W)^2 + \operatorname{E} X_{\imath-1}\operatorname{Var} W + \operatorname{Var} C.$$

We have the following general solution

$$\operatorname{Var} X_\imath = c_3(\operatorname{E} W)^{2(\imath-1)} - c_2(\operatorname{E} W)^{\imath-1} + c_1,$$

where

$$\begin{aligned} c_1 &= \frac{\operatorname{E} C \operatorname{Var} W + (1 - \operatorname{E} W)\operatorname{Var} C}{(1 - (\operatorname{E} W)^2)(1 - \operatorname{E} W)}, \\ c_2 &= \frac{\operatorname{E} C \operatorname{Var} W}{(1 - \operatorname{E} W)^2}, \\ c_3 &= c_2 - c_1 + \operatorname{Var} C. \end{aligned}$$

Substituting $\operatorname{Var} X_{\imath-1}$ into $\operatorname{Var} Y_\imath$ yields equation (5).

In Case of $\operatorname{E} W = 1$ The expression presented above is derived under the assumption that $\operatorname{E} W \neq 1$. When $\operatorname{E} W = 1$, $G'_{X_\imath}(1) = G'_{X_{\imath-1}}(1) + G'_C(1)$. Solving this recurrence yields

$$\operatorname{E} X_\imath = \imath \operatorname{E} C.$$

Then

$$\operatorname{E} Y_\imath = (\imath - 1) \operatorname{E} C \operatorname{E} Z + \operatorname{E} Zc.$$

Similarly, the expression of $\operatorname{Var} X_\imath$ can be simplified as

$$\operatorname{Var} X_\imath = \frac{\imath(\imath - 1)}{2}\operatorname{E} C \operatorname{Var} W + \imath \operatorname{Var} C.$$

Then equation (7) can be obtained by substituting $\operatorname{Var} X_{\imath-1}$ into $\operatorname{Var} Y_\imath$.