T-Time: A Fine-grained Timing-based Controlled-Channel Attack against Intel TDX

Woomin Lee[0,*], Taehun Kim[0,*], Seunghee Shin[†], Junbeom Hur[*], and Youngjoo Shin[*]

[*]Korea University, [†]State University of New York at Binghamton
[*]{redcokeb,taehunk,jbhur,syoungjoo}@korea.ac.kr [†]sshin@binghamton.edu

Abstract. Intel's Trust Domain Extensions (TDX) is a Confidential Virtual Machine (CVM) technology designed to enhance security through Trusted Execution Environments (TEEs). Although TDX effectively mitigates interrupt-based stepping attacks, it remains vulnerable to controlled-channel attacks, which exploit page-level memory access patterns to infer secret-dependent control flows. Current defenses confine sensitive execution within single memory pages to reduce observable access patterns. We challenge this strategy by introducing T-Time, a fine-grained timing-based controlled-channel attack targeting Intel TDX. T-Time precisely measures dwell timethe interval between consecutive page faultsto uncover previously hidden sensitive control flows. We further enhance T-Times precision through a cache-based amplification technique. We validate T-Time in two practical scenarios: extracting a 4096-bit RSA private key from MbedTLS, and reconstructing a WebP image through timing analysis during decoding. Our findings demonstrate that existing page-level defenses are inadequate against fine-grained timing attacks.

1 Introduction

Intel's Trust Domain Extensions (TDX) is the latest Confidential Virtual Machine (CVM) technology designed to provide robust hardware-enforced isolation through Trusted Execution Environments (TEEs). TDX specifically addresses interrupt-based stepping attacks [8, 29, 30, 36, 39], a longstanding and critical security issue prominently associated with its predecessor, Intel SGX. Despite this significant advancement, TDX inherently lacks comprehensive protection against controlled-channel attacks, which remains a fundamental security challenge [1].

Controlled-channel attacks exploit predictable interactions between a victim's control flow and the underlying low-level system, enabling privileged attackers to infer sensitive information by observing page-level memory access patterns [37]. Due to the persistent nature of this vulnerability, Intel relies heavily on software developers to implement defenses against these attacks [9]. Developers are encouraged to ensure secret-dependent control flows remain within a single memory page, effectively minimizing observable page-level access patterns. This

[0] These authors contributed equally to this work.

strategy has traditionally been deemed sufficient, as controlled-channel attacks were thought to be inherently limited by page-level granularity.

However, this paper challenges the assumption that restricting secret-dependent execution to a single memory page provides adequate protection. We introduce *T-Time*, a novel fine-grained timing-based controlled-channel attack specifically designed to circumvent the existing page-level defense strategy in Intel TDX. T-Time precisely measures *dwell time*the time interval between consecutive page faultsto reveal secret-dependent execution flows that were previously undetectable through traditional controlled-channel methods.

To further enhance the precision and effectiveness of T-Time, we propose a novel cache-based amplification technique that significantly increases timing differences between secret-dependent code paths residing within the same page. Our technique flushes target cache lines and measures dwell time, thereby amplifying otherwise subtle timing discrepancies and enabling T-Time to reliably infer sensitive control-flow decisions.

We demonstrate the practical impact and severity of T-Time through two concrete case studies. First, we extract a full 4096-bit RSA private key from the MbedTLS library using only 20 signing operations, highlighting a significant cryptographic vulnerability. Second, we reconstruct a WebP image through detailed timing analysis during its decoding, emphasizing T-Time's capability to infer sensitive data in realistic application scenarios. These examples clearly illustrate that current software-driven defensive practices based on single-page confinement are insufficient against fine-grained timing attacks. Finally, we discuss several mitigation strategies to address the risks posed by T-Time. The source code for our attack is publicly available at https://github.com/koreacsl/T-Time.

Contributions. In this paper, we make the following contributions:

- We present T-Time, a novel fine-grained timing-based controlled-channel attack that circumvents existing page-level defense strategy.
- We propose a technique that precisely measures dwell time via induced page faults and a timing amplification using a cache flush technique.
- We demonstrate T-Time's effectiveness through case studies, recovering an RSA-4096 private key and reconstructing a WebP image.

Responsible disclosure. We disclosed our findings to Intel's PSIRT team, as well as to the maintainers of MbedTLS, WebP, Google Chrome, and Mozilla Firefox on April 23, 2025. In response, Intel PSIRT informed us that T-Time falls under existing side-channel guidance and requires no further action. The MbedTLS maintainer informed us that T-Time is ineffective against version 3.6.0 and later, since the `mbedtls_mpi_exp_mod()` function was rewritten to use a constant-time implementation. The WebP team responded that T-Time is not specific to libwebp and may impact other media decoders. Furthermore, they noted that, in general, Chrome cannot do much about timing attacks, and remarked that this vulnerability should be addressed at the hypervisor level, rather than by modifying individual libraries.

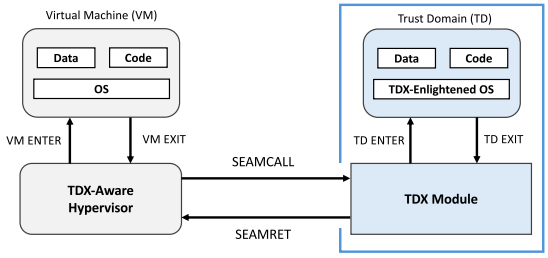


Fig. 1: Architecture of Intel TDX.

2 Background

2.1 Intel trust domain extension

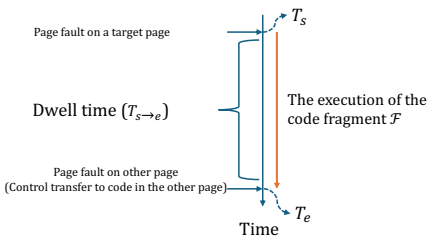
Intel TDX [12] is a hardware-based memory isolation technology that provides TEE for virtual machines (VMs). Intel TDX achieves this by introducing the TDX module, a core component providing hardware-enforced isolation for VMs. A VM protected by TDX is called a Trust Domain (TD). The TDX module [11] is responsible for ensuring secure communication and isolation between the hypervisor and the TD. As shown in Figure 1, the TDX module acts as an intermediary between the hypervisor and the TD, securely handling communication between them. The hypervisor transfers control to the TD via the TDX module by invoking the `SEAMCALL` instruction, and control is returned to the hypervisor when the TDX module executes the `SEAMRET` instruction.

Page fault handling. A page fault occurs when a process accesses a virtual address that violates the current page table configuration, such as when the address is not mapped to physical memory. When a page fault occurs within a TD, the exception is intercepted by the TDX module, and control is transferred to the hypervisor via the `SEAMRET` instruction. The hypervisor, operating outside the TD context, resolves the faulttypically by mapping the required page or updating the page tables. Once the fault is handled, TD execution resumes through the `SEAMCALL` instruction, which returns control to the TDX module and re-enters the TD.

Memory encryption. The processor enforces strict access control policies to isolate TD-assigned memory from all other applications, including the host and other TDs. To ensure data confidentiality and integrity, each TD's memory is encrypted using Intels Multi-key Total Memory Encryption (MKTME) [10] mechanism.

2.2 Controlled-channel attack

TEEs are hardware-based isolation technologies designed to protect sensitive workloads [4] or virtual machines (VMs) [12, 28] from a potentially compromised OS or hypervisor. However, since the OS or hypervisor still controls low-level



(a) Timing difference in the target page.

```
...
1: invoke_func_in_other_page ();
2: // return to the target page (page fault occurs at T_s)
3:
4: // secret dependent code path
5: if ( secret ) {
6:   execute control flow f_1
7: }
8: else {
9:     execute control flow f_0
10: }
11:
12: // Transfer control to other page (page fault occurs at T_e)
13: invoke_func_in_other_page ();
...
```

(b) Content of the target page.

Fig. 2: T-Time attack

resources such as memory management, several new attack surfaces for side-channel attacks have emerged. One notable example is the controlled-channel attack [37], which exploits the attackers privileged control over memory management to infer the victims memory access patterns. This attack is performed in four steps. In step 1, the attacker clears the present bits in the page table entries corresponding to memory regions of the TEE-protected application or VM. In step 2, the attacker induces the victim to perform memory accesses. Since the present bits are unset, any such memory access by the victim results in a page fault. In step 3, the attacker handles the trapped fault and logs the page fault sequence triggered by the victim's accesses. In step 4, the attacker analyzes the collected faulting addresses and recovers security-sensitive information. Because the controlled-channel attack relies on page faults to gather access patterns, its spatial resolution is inherently limited to the granularity of a memory page.

3 Building Attack Primitive

The main idea behind T-Time is that different execution flows lead to different execution times. By measuring dwell time—the duration between two consecutive page faults, T-Time uncovers secret-dependent control flows that have been deemed safe.

Figure 2 illustrates the concept of dwell time and provides an example of how it can be utilized in the T-Time attack. As shown in Figure 2(a), the execution of the code fragment $\mathcal{F}$, whose execution time reveals a secret, is surrounded by two consecutive page faults, occurring at pages P_s and P_e. Dwell time $\mathcal{T}_{s\to e}$ refers to the time interval $(T_e - T_s)$ between the time (T_s) when a fault occurs at page P_s and the time (T_e) when a fault occurs at page P_e. By measuring this dwell time, we can infer the execution time of $\mathcal{F}$, which in turn leaks the secret.

Figure 2(b) illustrates how dwell time can be leveraged in the T-Time attack. Since different control flows, such as f_1 and f_0, typically execute a different number of instructions, the corresponding dwell times also differ. By precisely measuring these dwell times, an attacker can distinguish between secret-dependent

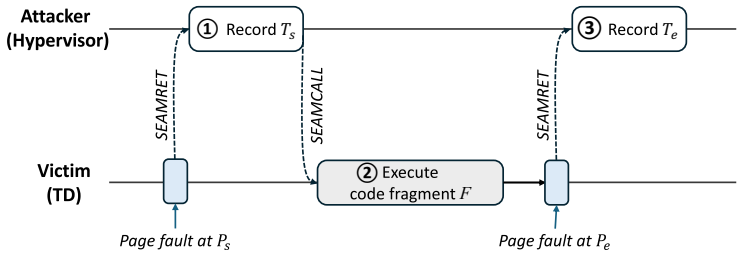


Fig. 3: Measuring dwell time between two successive page faults.

execution paths, even when both reside within the same memory page. Thus, this enables the extraction of sensitive information that would otherwise be assumed secure under page-level confinement. It is noteworthy that traditional controlled-channel attacks cannot infer the secret if both control flows (f_1 and f_0) result in identical page access sequences, highlighting the advantage of T-Time attack.

In Section 3.1, we present our attack. In Section 3.2, we propose a cache-based amplification technique. In Section 3.3, we evaluate the performance of the T-Time attack primitive and the amplification technique.

3.1 T-Time attack

Figure 3 illustrates a T-Time attack. The attacker attempts to infer the execution time of a code path $\mathcal{F}$ (e.g., lines 5-10 in Figure 2(b)). The attacker chooses two target pages, P_s and P_e, such that a page fault occurs at P_s immediately before the execution of $\mathcal{F}$ (e.g., line 1 in Figure 2(b)), and another page fault occurs at P_e immediately after the execution of $\mathcal{F}$ (e.g., line 13 in Figure 2(b)). By measuring the dwell time $\mathcal{T}_{s\to e}$, the attacker can infer the execution time of $\mathcal{F}$.

As an initialization step, the attacker clears the present bits of pages P_s and P_e. When a victim begins executing code, the attack proceeds as follows. ① When a page fault occurs at P_s, the attacker's page fault handler is invoked by `SEAMRET` from the TDX module. The handler then records a timestamp T_s just before re-entering the victim TD via `SEAMCALL`. ② Upon re-entry, the victim continues to execute code including $\mathcal{F}$. ③ At the second fault at P_e, the attacker's handler records T_e immediately after regaining control and computes the dwell time as $\mathcal{T}_{s\to e} = T_e - T_s$.

Precisely measuring dwell time serves as the primitive of the T-Time attack. However, implementing the attack poses two main challenges. The first challenge involves monitoring the victim's execution flow at the page level. Traditional controlled-channel attacks exploit an attacker's high privilege to manipulate the present bit in the page table entry. However, Intel TDX strictly prohibits such manipulation, even for attackers with hypervisor privileges. To address this, we employ a method introduced by Aktas et al. [1], leveraging the `TDH.MEM.RANGE.BLOCK` API. This API temporarily blocks victim access to targeted memory pages by clearing their present bits. When the victim accesses a

blocked page, an extended page table (EPT) violation triggers a VM exit, transferring control to the hypervisor along with the faulting address. The hypervisor can then identify the accessed pages and subsequently unblock them, enabling continuous and non-destructive monitoring of the victim's execution flow.

The second challenge is obtaining highly precise timing information from the TD. To accomplish this, we instrument the KVM hypervisor to capture timestamps immediately before and after interactions with the TD. We record the timestamp counter immediately before invoking `SEAMCALL`, which transfers control from the hypervisor to the TD. We record it again immediately after `SEAMRET`, when control returns from the TDX module back to the hypervisor. We record the timestamp counter immediately before invoking `SEAMCALL`, which transfers control from the hypervisor to the TD. This method ensures accurate measurement of dwell time, facilitating reliable differentiation between distinct execution paths.

3.2 Timing amplification using cache flush technique

We introduce a novel cache-based timing amplification technique designed to enhance the distinguishability of subtle timing variations. The key idea is to flush specific cache lines within the target code fragment $\mathcal{F}$ and subsequently measure their dwell time, thus amplifying subtle timing discrepancies. Leveraging this method, the T-Time attack can reliably differentiate secret-dependent control flows that would otherwise appear indistinguishable.

In Intel TDX environments, however, flushing cache lines poses significant challenges. Specifically, memory requests are encrypted and tagged with unique KeyIDs, preventing attackers from directly flushing the target cache lines associated with the victim TDs KeyID. To circumvent this limitation, we adopt the aliased access approach [1]. This method exploits the cache coherence protocol, where memory requests issued with a different KeyID (aliased access) cause existing cache lines associated with the victims KeyID to be evicted and replaced by data corresponding to the new KeyID. By utilizing aliased access, we achieve an indirect yet effective cache line flush.

Our technique involves the following two steps. ① The attacker flushes a target cache line by performing an aliased access. ② Then, the attacker measures the dwell time of the victim on the target page.

3.3 Performance evaluation

We performed experiments to evaluate the effectiveness of the T-Time attack. The experiments were conducted on an Intel Xeon Silver 4510T processor with KVM as the hypervisor. The host ran Ubuntu 24.10 (kernel 6.11.0) and Intel TDX 1.5.0.6, while the guest VM operated on Ubuntu 24.04 LTS (kernel 6.8.0-36-generic). To ensure precise timing measurements, we disabled DVFS, C-states, hardware prefetching, and reserved a dedicated physical core for the VM.

We evaluate the precision of dwell time measurements using T-Time. For this evaluation, we use a code snippet containing conditional branches within

```
if (secret)
{
    asm (".rept M"
         "imul %r12, %r13"  // Repeat M times if secret is true.
         ".endr" );
}
else
{
    asm (".rept N"
         "imul %r12, %r13"  // Repeat N times if secret is false.
         ".endr");
}
```

Listing 1: An example of secret-dependent code.

a single 4KB memory page, where execution paths depend on a secret value (see Listing 1). When the secret is true, the code executes the `imul r64, r64` instruction M times; otherwise, it executes N times. We measured dwell times over 1,000,000 iterations for the `else` path, varying N between 20 and 100, while fixing $M = 10$ for the `if` path. We denote T_n as the average dwell time when the `imul` instruction executes n times in the secret-dependent path.

We use two metrics for this comparison: the dwell time difference Δ_n and the Fisher score F_n, defined as:

$$\Delta_n = T_n - T_{10}, \quad F_n = \frac{(T_n - T_{10})^2}{\sigma_n^2 + \sigma_{10}^2}.$$

Table 1 presents the average dwell times and corresponding Fisher scores between the two execution paths. A Fisher score F_n below 1 indicates that the measured dwell time does not provide sufficient power to distinguish between the execution paths reliably.

To verify the effectiveness of our cache-based timing amplification technique, we compared it against a baseline T-Time attack without amplification. Without amplification, timing differences become noticeable at Δ_{60}, indicating that secret-dependent behavior becomes distinguishable starting at $n = 60$. At this point, the Fisher score reaches $F_{60} = 0.95$, confirming the reliability of the T-Time attack primitive. When employing our amplification technique, we flush the target cache line (lines 911) from the cache hierarchy. With amplification,

Table 1: Average dwell time difference (Δ) and Fisher score (F) of the attack.

		20	30	40	50	60	70	80	90	100
T-Time	Delta (Δ_n)	3.20	30.09	30.14	16.64	49.72	82.09	113.45	122.29	161.23
	F-score (F_n)	0.00	0.32	0.31	0.09	0.95	2.19	4.36	4.38	8.14
T-Time (amp)	Delta (Δ_n)	240.36	301.61	276.19	296.60	405.68	416.39	428.92	455.60	497.02
	F-score (F_n)	17.06	19.95	25.87	23.98	55.06	55.03	60.39	63.24	78.06

the dwell time difference at Δ_{20} markedly increases from the baseline 3.20 to 240.36. Correspondingly, the Fisher score improves significantly to $F_{20} = 17.06$, highlighting that the amplification substantially enhances the distinction between execution paths. Thus, cache-based amplification dramatically improves the capability of T-Time, particularly in cases where timing differences would otherwise be too subtle to detect.

4 T-Time Attack on MbedTLS RSA

We demonstrate T-Time attack that recovers complete 4096-bit RSA private keys from multiple traces of the MbedTLS (v3.5.2) signature routine, which is implemented using non-constant time algorithms.

```
/* mbedtls_mpi_exp_mod(), mpi_select(), and mpi_montmul() are located
   on a page Pe, Ps, Pm respectively. */

int mbedtls_mpi_exp_mod () {
    for (int i = 0; i < bit_length; i++) {
        if (e[i] == 0) {
            mpi_select();                   /* Page fault at Ps */
            // Return from mpi_select()     /* Page fault at Pe */
            mpi_montmul();                  /* Page fault at Pm */
            // Return from mpi_montmul()    /* Page fault at Pe */
        } else {
            mpi_select();                   /* Page fault at Ps */
            // Return from mpi_select()     /* Page fault at Pe */
            mpi_montmul();                  /* Page fault at Pm */
            // Return from mpi_montmul()    /* Page fault at Pe */
            mpi_select();                   /* Page fault at Ps */
            // Return from mpi_select()     /* Page fault at Pe */
            mpi_montmul();                  /* Page fault at Pm */
            // Return from mpi_montmul()    /* Page fault at Pe */
        }
    }
}
```

Listing 2: Simplified version of `mbedtls_mpi_exp_mod()`.

4.1 Signature process of mbedTLS

MbedTLS [18] implements RSA modular exponentiation through a windowed square-and-multiply method within the function `mbedtls_mpi_exp_mod()`. Since prior work has shown that side-channel attacks against the square-and-multiply algorithm with a window size of 1 can be generalized to arbitrary window sizes [20], our attack similarly targets the setting with a fixed window size of 1 [5, 6, 19, 27].

Listing 2 simplifies `mbedtls_mpi_exp_mod()` function[1]. with setting of window size of 1. The function, which resides on a page P_e, executes modular exponentiation by conditionally branching based on each bit (e_i) of the exponent,

[1] The full source code is available at https://github.com/Mbed-TLS/mbedtls/blob/v3.5.2/library/bignum.c

resulting in distinct page access sequences. Specifically, when the exponent bit e_i is 0, the function calls `mpi_select()` on page P_s and `mpi_montmul()` on page P_m exactly once, resulting in accesses to two pages in total. Here, `mpi_montmul()` performs a square operation. In contrast, when e_i is 1, each of these functions is called twice, resulting in accesses to four pages. In this case, `mpi_montmul()` executes a square operation and an additional multiply operation.

Although these functions reside on separate memory pages, traditional controlled-channel attacks have difficulty precisely reconstructing these call sequences. This difficulty arises because different combinations of exponent bits can lead to identical page access patterns. For example, a page fault sequence such as $P_s \quad P_m \quad P_s \quad P_m$ could correspond either to a single exponent bit being 1 ($e_i = 1$) or to two consecutive exponent bits both being 0 ($e_i = 0$, $e_{i+1} = 0$). This ambiguity makes it challenging for traditional attacks to accurately infer secret-dependent execution flows.

4.2 Recovery of RSA Private Keys using T-Time

The idea underlying our attack is to exploit the fact that `mbedtls_mpi_exp_mod()` takes secret-dependent control paths that occupy different cache lines. Although the page fault sequences generated by each control path do not show distinct patterns, we can make their dwell times distinguishable by applying our cache-based timing amplification technique presented in Section 3.2.

As shown in Table 2, `mbedtls_mpi_exp_mod()` executes different control paths depending on the exponent bits e_i, with each control path residing at distinct memory addresses. For example, when $e_i = 0$, the control path executes code (lines 7-10 in Listing 2), located at address range `0x25a24` $\sim$ `0x25a6f`, whereas when $e_i = 1$, it executes code (lines 12-19 in Listing 2) at `0x25b69` $\sim$ `0x25bc1`.

This observation yields two critical insights. First, the page fault sequences triggered by distinct exponent bit occur at different execution code. Second, these execution codes reside on separate cache lines.

Leveraging these characteristics, our cache-based timing amplification technique specifically targets the cache lines associated with the execution path for $e_i = 1$, effectively amplifying its dwell time. As a result, the eight dwell time measurements for $e_i = 1$ (i.e., $T_{s\to e}, T_{e\to m}, T_{m\to e}, \ldots$ listed in Table 2) become significantly larger than those measured when $e_i = 0$. Exploiting this distinct timing difference forms the core principle underlying our attack.

Table 2: The control structure of `mbedtls_mpi_exp_mod()` (v3.5.2)

Control path	Virtual address	Page fault sequence	Dwell time measurements
$e_i = 0$	`0x25a24` $\sim$ `0x25a6f`	P_s, P_e, P_m, P_e	$\mathcal{T}_{s\to e}$, $\mathcal{T}_{e\to m}$, $\mathcal{T}_{m\to e}$, $\mathcal{T}_{e\to s}$
$e_i = 1$	`0x25b69` $\sim$ `0x25bc1`	P_s, P_e, P_m, P_e, P_s, P_e, P_m, P_e	$\mathcal{T}_{s\to e}$, $\mathcal{T}_{e\to m}$, $\mathcal{T}_{m\to e}$, $\mathcal{T}_{e\to s}$, $\mathcal{T}_{s\to e}$, $\mathcal{T}_{e\to m}$, $\mathcal{T}_{m\to e}$, $\mathcal{T}_{e\to s}$

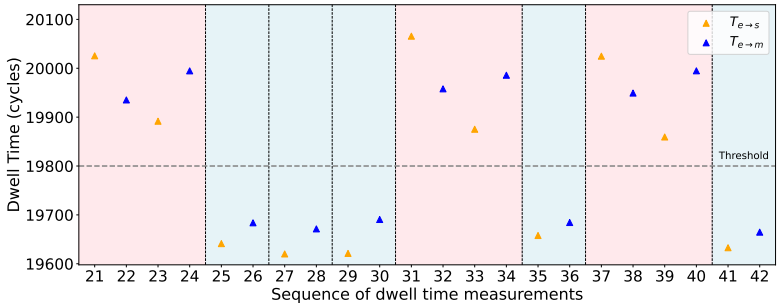


Fig. 4: Trace for exponent bits of RSA private keys.

Threat model. We assume a privileged attacker with full access to the operating system, who is capable of inferring guest physical addresses through legacy controlled-channel attacks. The attacker is aware that the victim is running the default MbedTLS library (version 3.5.2) inside a TDX-protected VM. Leveraging this knowledge, the attacker determines in advance target code pages (P_s, P_m, and P_e) whose access patterns make them vulnerable to the T-Time attack. The attacker aims to extract the RSA private key from a victim process executing within the TDX-protected environment on the same host system.

Attack process. The attacker aims to extract the victim's RSA private key during RSA signing operations. Specifically, the attacker targets three pages, P_s, P_m, and P_e, to measure dwell times, applying the cache-based amplification technique to the execution path corresponding to the exponent bit $e_i = 1$, as detailed in Table 2. A sequence of eight consecutive dwell times (i.e., $T_{s\to e}, T_{e\to m}, T_{m\to e}, \ldots, T_{e\to s}$) exhibiting notably high cycle counts indicates that the corresponding exponent bit e_i is 1.

Figure 4 illustrates an example of the dwell time traces obtained from these measurements. For simplicity and clarity, the figure includes only $T_{e\to s}$ and $T_{e\to m}$, which are sufficient for accurately reconstructing the private keys. Sequence numbers 21 through 24 exhibit notably high dwell times, highlighted in red in Figure 4, indicating that the execution path corresponding to $e_i = 1$ was taken. These high dwell times result from cache misses at the memory addresses associated with this execution path, induced by our amplification technique. Conversely, sequence numbers 25 and 26, highlighted in blue, show significantly lower dwell times due to cache hits, indicating that the execution path for $e_i = 0$ was taken. Based on this distinction, the collected trace allows us to infer the exponent bits as `10001010`.

Evaluation. We evaluate our attack on an Intel Xeon Silver 4510T CPU, with the host system running Ubuntu 24.10 and Intel TDX Module version 1.5.0.6, and the guest VM running Ubuntu 24.04 LTS. To mitigate measurement noise, we collect 20 RSA signing traces using the same private key and calculate the average dwell time for each transition. Our attack successfully recovers the complete RSA private key without any bit errors by combining only 20 traces, completing the reconstruction in under 25 seconds.

5 T-Time Attack on Libwebp

5.1 WebP image format and decoding process in libwebp

WebP [7] is a modern image compression format developed by Google, supporting both lossy and lossless compression modes. The lossy mode of WebP uses techniques similar to those used in video encoding, specifically from Google's VP8 video codec [2]. In simple terms, a WebP image is divided into small blocks (16Œ16 pixels), known as macroblocks, which are compressed and later reconstructed during decoding. Each macroblock is further divided into sixteen 4Œ4 sub-blocks, which serve as the basic units for transformation and coefficient encoding.

Macroblocks encode image information in terms of basic brightness (DC coefficients) and finer details like edges and textures (AC coefficients). WebP uses a method called YUV 4:2:0, where brightness information (luminance or Y) is stored in greater detail than color information (chrominance or U and V), since the human eye is more sensitive to brightness than color details. In this format, all three components (Y, U, V) are encoded using both DC and AC coefficients.

```
1  int ParseResiduals() {
2      if (!is_i4x4_) {
3          GetCoeffs();                    // Parse Y DC coefficients
4      }
5
6      for (int y=0; y<4; y++) {
7          for (int x=0; x<4; x++) {
8              GetCoeffs();                // Parse Y block coefficients
9          }
10     }
11
12     for (int ch=0; ch<4; ch=ch+2) {     // U(0), V(2) channels
13         for (int y=0; y<2; y++) {
14             for (int x=0; x<2; x++) {
15                 GetCoeffs();            // Parse U/V block coefficients
16             }
17         }
18     }
19 }
```

Listing 3: Simplified version of `ParseResiduals()`.

Listing 3 simplifies the decoding process for a single macroblock[2]. During decoding, `ParseResiduals()` in libwebp is responsible for decoding and reconstructing the macroblock coefficients. As part of the decoding routine, `GetCoeffs()` is repeatedly invoked to extract the transform coefficients from each sub-block. The decoding sequence within this function follows three primary stages. First, if the macroblock is not encoded in a 4x4 intra-prediction

[2] The full source code is available at https://github.com/webmproject/libwebp/blob/v1.5.0/src/dec/vp8_dec.c

mode (i.e., `is_i4x4_` is false), `GetCoeffs()` is called once to decode the DC coefficients for the luminance (Y) component. The DC coefficient for the Y component is shared across the macroblocks sixteen 4Œ4 sub-blocks.

In the second stage, the function processes detailed luminance information by iterating over each of the sixteen 4Œ4 sub-blocks within the macroblock. If the macroblock is not encoded in 4Œ4 intra-prediction mode (i.e., `is_i4x4_` is false), `GetCoeffs()` is invoked to recover only the AC coefficients for each sub-block. In contrast, if the macroblock is encoded in 4Œ4 intra-prediction mode (`is_i4x4_` is true), `GetCoeffs()` is called to decode both DC and AC coefficients for each sub-block.

Finally, the function decodes chrominance details (color information) for the U and V channels. Because WebP utilizes YUV 4:2:0 subsampling, chrominance decoding occurs over four 4Œ4 sub-blocks (two for U and two for V). For each chroma block, `GetCoeffs()` is invoked to decode both DC and AC coefficients. In total, `GetCoeffs()` is called 24 times per macroblock: 16 times for Y, and 4 times each for U and V.

5.2 Inferring image pixels with T-Time

The fundamental idea behind attacking the WebP decoding process is exploiting differences in execution time within the `GetCoeffs()` function. Specifically, `GetCoeffs()` exhibits varying execution times depending on the presence of coefficients in the current block. By measuring these dwell times with a T-Time attack, an attacker can infer the presence or absence of coefficients. High dwell time indicates the presence of more coefficients, which reflects higher visual complexity in the corresponding image block. By repeatedly measuring the dwell times across all 24 invocations of `GetCoeffs()` per macroblock16 for luminance and 8 for chrominancean attacker can recover the coefficient activity pattern across the block. Ultimately, this extracted information enables the reconstruction of structural features of the underlying image, such as edges, contours, and textured regions, without requiring access to actual pixel values.

It is noteworthy that traditional controlled-channel attacks cannot reconstruct the WebP image, as the implementation of libwebp, including the `GetCoeffs()` function, has no distinguishable patterns at page-level granularity.

Threat model. The attacker aims to reconstruct a WebP image during its decoding by a victim process (e.g., a web browser) running inside an Intel TDX-protected virtual machine, which is hosted on a system under the attacker's control. The attacker is aware that the victim employs an unmodified, up-to-date libwebp library (e.g., version 1.5.0). By leveraging page tracking techniques, the attacker can determine guest physical addresses of the libwebp library, enabling precise observation of the decoding execution.

Attack process. The attacker aims to infer the execution time of the `GetCoeffs()` function by targeting two pages of the libwebp library, P_p and P_g, which contain `ParseResiduals()` and `GetCoeffs()`, respectively. By measuring the dwell time $\mathcal{T}_{g \to p}$, the time interval between a page fault at P_g (triggered by the invo-

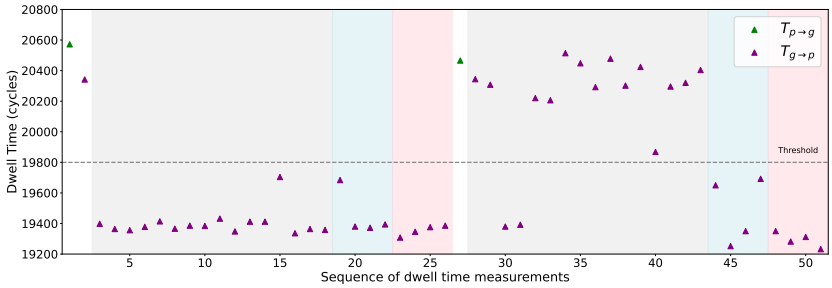


Fig. 5: Trace for two consecutive macroblocks (Grey, blue, and red box indicate decoding phase of Y, U, and V blocks, respectively.).

cation of `GetCoeffs()`) and a subsequent fault at P_p (caused by the return to `ParseResiduals()`), the attacker can successfully infer its execution time.

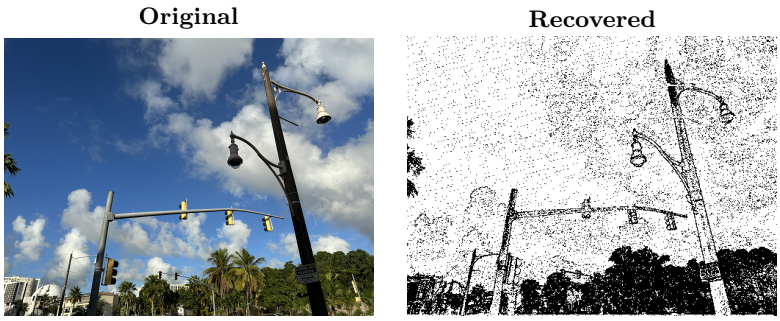
When a victim starts image decoding (by browsing an image on a web page), the trace of the dwell times begins. Since one call to `ParseResiduals()` causes a total of 24 subsequent calls to `GetCoeffs()` for decoding coefficients, the trace for each macroblock B results in a sequence $C = \{c_i\}_{(0 \le i < 24)}$, where each c_i represents the execution cycles of a corresponding `GetCoeffs()` invocation. A higher cycle count of c_i indicates the presence of a coefficient in i-th call of `GetCoeffs()`, whereas lower counts do not. This provides 24 bits of information about a single macroblock in the image, enough for reconstructing the image's structural content.

For example, Figure 5 shows the obtained trace for two vertically aligned macroblocks B_1 and B_2. The trace for each macroblock begins with a sharp execution time peak corresponding to the `ParseResiduals()` function, allowing an attacker to infer macroblock boundaries, followed by a sequence of `GetCoeffs()` measurements. The number of observed `GetCoeffs()` executions is different for these two macroblocks, 25 for B_1 and 24 for B_2, due to the different control paths in decoding the DC coefficients. That is, B_1 contains an additional call to `GetCoeffs()` because it has to decode DC coefficients (i.e., `is_4x4` is true in Listing 3).

The first sequence (page fault sequence numbers 1-26) corresponds to the decoding of the first macroblock B_1, which contains both DC and AC coefficients. Specifically, page fault sequence number 1 involves the starting of macroblock decoding. Number 2 involves the decoding of DC coefficients, and numbers 3-18 mark the transition to the AC decoding routine. Numbers 19-22 and 23-26 cover

Table 3: Traces of `GetCoeffFast()` dwell times for AC coefficients (H and L indicate high and low cycles, respectively.)

Macroblock	Y block																U block				V block			
B_1	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
B_2	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L



(a) Recovered using Y (luma) channel

(b) Recovered using YUV channels

Fig. 6: Attack results.

the chroma U and V channels, respectively. The second sequence (page fault sequence numbers 27-51) corresponds to those of B_2, which can be analyzed in the same way except for decoding the DC and AC coefficients. Table 3 shows the obtained information of the coefficients for B_1 and B_2, from which the original image can be inferred.

Evaluation. We conducted our attack on an Intel Xeon Silver 4510T CPU, with the host system running Ubuntu 24.10 and Intel TDX Module version 1.5.0.6, and the guest VM running Ubuntu 24.04 LTS. To evaluate the effectiveness of the proposed T-Time attack, we assembled a test set comprising various images obtained from publicly available sources. The dataset includes complex, high-resolution photographs. For each image, we collected a single trace comprising a page fault sequence and corresponding dwell times, revealing the memory access behavior during the decoding process. These traces were subsequently analyzed using a custom reconstruction tool to produce visual approximations of the original images in PNG format. Figure 6 presents three representative examples of reconstructed images, each derived from a single decoding trace. Figure 6(a) shows an image reconstructed by inferring only the coefficients of the luminance (Y) channel. Figure 6(b) demonstrates improved reconstruction quality achieved by incorporating coefficients from all Y, U, and V channels.

6 Mitigation

In this section, we propose several mitigation strategies to defend against the T-Time attack.

Constant-time algorithms. Since T-Time exploits timing variations, a crucial mitigation involves eliminating timing discrepancies between secret-dependent execution paths. Developers should therefore employ constant-time algorithms [13, 14, 22] to ensure security-sensitive code executes in a uniform manner, independent of secret values. Achieving constant-time execution involves techniques such as aligning sensitive code within the same cache line and standardizing execution durations across secret-dependent control paths.

Enhancing Intel TDX security. We propose additional mitigations specifically tailored for Intel TDX environments. First, the TDX module could monitor for abnormal patterns of repeated page faults within the same memory region. By restricting the invocation of the `TDH.MEM.RANGE.BLOCK` API in these identified regions, the module can effectively reduce the feasibility of frequent page faults used in T-Time. Second, we suggest disabling the timestamp counter while a TD is actively running on a physical core. As T-Time relies heavily on precise timestamp measurements to distinguish secret-dependent execution paths, disabling the timestamp counter upon TD entry and re-enabling it on TD exit can significantly mitigate the attack's effectiveness.

Introducing AEX-Notify on Intel TDX. Constable et al. [3] proposed AEX-Notify, a hardwaresoftware co-designed mitigation technique that defends against interrupt-based side-channel attacks [21, 25, 31] targeting SGX enclaves. The key idea of AEX-Notify is to make enclaves aware of interrupt events, enabling them to distinguish between normal and adversarial context switches, thereby reducing information leakage through side-channels. Based on this, a similar mechanism, TDExit-Notify [24], has been discussed for Intel TDX. TDExit-Notify allows a TD to detect interrupt events, thereby mitigating attacks like T-Time. Furthermore, with TDExit-Notify support, TLBlur [33]a compiler-assisted mitigation technique originally developed to protect SGX enclaves from controlled-channel attackscan be extended to support Intel TDX environments with minimal modification.

7 Related Work

Controlled-channel attack. Xu et al. [37] first introduced controlled-channel attacks by exploiting the page-fault side channel in Intel SGX. Subsequent studies [32, 34] improved their work by developing stealthier methods to infer enclave memory access patterns without explicitly inducing page faults. Additionally, numerous studies [6, 15–17, 23, 35, 36, 38] have demonstrated the feasibility of controlled-channel attacks across various TEE platforms, such as AMD SEV and Intel TDX.

In line with our research, Wang et al. [34] introduced the T-SPM attack, a timing-based page-fault side-channel attack against SGX enclaves. Our work

differs fundamentally from T-SPM in three critical aspects. First, T-Time explores the applicability of timing-based attacks specifically within the Intel TDX environment rather than SGX. Second, T-SPM can only measure coarse-grained timing differences and thereby struggles to distinguish between execution paths with closely similar runtimes. In contrast, T-Time leverages cache line flushing to amplify subtle timing differences, allowing it to reliably differentiate between such execution paths. Third, our evaluation of T-Time focuses on contemporary implementations of MbedTLS and WebP, where secret-dependent control flows are confined within single memory pages. Despite this defensive strategy effectively mitigating traditional coarse-grained attacks, our findings demonstrate that T-Time can still reliably extract security-sensitive information through fine-grained timing analysis.

Side-channel attacks on CVM. Recent research has extensively explored various attacks targeting CVMs. Heckler [26] introduces malicious interrupt injection techniques that manipulate the register states of CVMs. This method enables attackers to alter both data and control flow within applications. Consequently, it effectively bypasses security mechanisms in Intel TDX-protected environments.

Another work, TDXdown [35], addresses vulnerabilities associated with single-stepping mitigations implemented in Intel TDX. It leverages observable instruction execution counts within a TD. This technique successfully performs nonce truncation attacks. It effectively extracts ECDSA private keys from widely-used cryptographic libraries, including wolfSSL and OpenSSL.

Additionally, CounterSEVeillance [6] exploits performance counters to target RSA signature implementations in MbedTLS version 3.5.2 running on AMD SEV platforms. This attack demonstrates the feasibility of extracting a full 4096-bit RSA private key from a single execution trace within eight minutes. However, this attack is ineffective in Intel TDX environments since these platforms disable performance counters by default. In contrast, our proposed attack successfully recovers the same private key within only 30 seconds. This significantly enhances the practicality and speed of timing-based attacks on Intel TDX.

8 Conclusion

In this paper, we introduced T-Time, a fine-grained timing-based controlled-channel attack on Intel TDX. T-Time measures dwell time within memory pages to bypass existing defenses. To enhance its precision, we also introduced a cache-based amplification technique that magnifies subtle timing variations, enabling the attacker to distinguish sensitive control flows even when confined to a single memory page. In two case studies, we recovered a 4096-bit RSA private key from MbedTLS with only 20 signatures and reconstructed image content from libwebp via fine-grained dwell time analysis. These results underscore the limitations of current page-level defenses against such attacks. Finally, we proposed potential mitigation strategies to harden Intel TDX against such threats.

Acknowledgement

This research was supported by a National Research Foundation of Korea (NRF) grant, funded by the Korean government (MSIT) (RS-2023-NR077166, RS-2025-00563143, RS-2021-NR060143, RS-2023-00227165). This research was supported by the Korea University Grant. This work was supported by NSF CAREER Award CCF-2146475.

References

1. Aktas, E., Cohen, C., Eads, J., Forshaw, J., Wilhelm, F.: Intel trust domain extensions (tdx) security review. Google security review (2023)
2. Bankoski, J., Wilkins, P., Xu, Y.: Technical overview of vp8, an open source video codec for the web. In: 2011 IEEE International Conference on Multimedia and Expo. pp. 1–6. IEEE (2011)
3. Constable, S., Van Bulck, J., Cheng, X., Xiao, Y., Xing, C., Alexandrovich, I., Kim, T., Piessens, F., Vij, M., Silberstein, M.: Aex-notify: Thwarting precise single-stepping attacks through interrupt awareness for intel sgx enclaves. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 4051–4068 (2023)
4. Costan, V., Devadas, S.: Intel sgx explained. Cryptology ePrint Archive (2016)
5. Gast, S., Juffinger, J., Schwarzl, M., Saileshwar, G., Kogler, A., Franza, S., Köstl, M., Gruss, D.: Squip: Exploiting the scheduler queue contention side channel. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 2256–2272 (2023)
6. Gast, S., Weissteiner, H., Schröder, R.L., Gruss, D.: Counterseveillance: Performance-counter attacks on amd sev-snp. In: Network and Distributed System Security Symposium 2025: NDSS 2025 (2025)
7. Google: An image format for the web (2025), https://developers.google.com/speed/webp
8. 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 Transactions on Cryptographic Hardware and Embedded Systems pp. 321–347 (2020)
9. Intel: Intel® 64 and ia-32 architectures software developer manuals (2023), https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
10. Intel: Intel®architecture memory encryption technologies (2024), revision 336907-005US
11. Intel: Intel trust domain extensions (intel tdx) module base architecture specification (2025), https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/trust-domain-extensions.html, revision 348549-005US
12. Intel: Intel® trust domain extensions(intel®tdx) (2025), https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/trust-domain-extensions.html
13. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Advances in CryptologyCRYPTO96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16. pp. 104–113. Springer (1996)
14. Langley, A., Hamburg, M., Turner, S.: Rfc 7748: Elliptic curves for security (2016)

15. Li, M., Wilke, L., Wichelmann, J., Eisenbarth, T., Teodorescu, R., Zhang, Y.: A systematic look at ciphertext side channels on amd sev-snp. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 337–351 (2022)
16. Li, M., Zhang, Y., Lin, Z.: Crossline: Breaking" security-by-crash" based memory isolation in amd sev. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2937–2950 (2021)
17. Li, M., Zhang, Y., Lin, Z., Solihin, Y.: Exploiting unprotected i/o operations in amds secure encrypted virtualization. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 1257–1272 (2019)
18. Linaro: Mbedtls (2025), https://www.trustedfirmware.org/projects/mbed-tls/
19. Lipp, M., Kogler, A., Oswald, D., Schwarz, M., Easdon, C., Canella, C., Gruss, D.: Platypus: Software-based power side-channel attacks on x86. In: IEEE SP. pp. 355–371 (2021)
20. 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 (2015)
21. Moghimi, D., Van Bulck, J., Heninger, N., Piessens, F., Sunar, B.: Copycat: Controlled instruction-level attacks on enclaves. In: 29th USENIX security symposium (USENIX security 20). pp. 469–486 (2020)
22. Molnar, D., Piotrowski, M., Schultz, D., Wagner, D.: The program counter security model: Automatic detection and removal of control-flow side channel attacks. In: International Conference on Information Security and Cryptology. pp. 156–168. Springer (2005)
23. Morbitzer, M., Huber, M., Horsch, J., Wessel, S.: Severed: Subverting amd's virtual machine encryption. In: Proceedings of the 11th European Workshop on Systems Security. pp. 1–6 (2018)
24. PradyumnaShome: Closing the intel tdx page fault side channel, or, the case for tdexit-notify (2024), https://collective.flashbots.net/t/closing-the-intel-tdx-page-fault-side-channel-or-the-case-for-tdexit-notify/3775
25. Puddu, I., Schneider, M., Haller, M., Čapkun, S.: Frontal attack: Leaking control-flow in sgx via the cpu frontend. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 663–680 (2021)
26. Schlüter, B., Sridhara, S., Kuhne, M., Bertschi, A., Shinde, S.: Heckler: Breaking confidential vms with malicious interrupts. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 3459–3476 (2024)
27. Schwarz, M., Weiser, S., Gruss, D., Maurice, C., Mangard, S.: Malware guard extension: Using sgx to conceal cache attacks. In: Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14. pp. 3–24 (2017)
28. Sev-Snp, A.: Strengthening vm isolation with integrity protection and more. White Paper, January **53**(2020), 1450–1465 (2020)
29. Sieck, F., Zhang, Z., Berndt, S., Chuengsatiansup, C., Eisenbarth, T., Yarom, Y.: Teejam: Sub-cache-line leakages strike back. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(1), 457–500 (2024)
30. Van Bulck, J., Piessens, F., Strackx, R.: Sgx-step: A practical attack framework for precise enclave execution control. In: SysTEX. pp. 1–6 (2017)
31. Van Bulck, J., Piessens, F., Strackx, R.: Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 178–195 (2018)

32. Van Bulck, J., Weichbrodt, N., Kapitza, R., Piessens, F., Strackx, R.: Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 1041–1056 (2017)
33. Vanoverloop, D., Sanchez, A., Toffalini, F., Piessens, F., Payer, M., Van Bulck, J.: Tlblur: Compiler-assisted automated hardening against controlled channels on off-the-shelf intel sgx platforms. In: USENIX Security (2025)
34. Wang, W., Chen, G., Pan, X., Zhang, Y., Wang, X., Bindschaedler, V., Tang, H., Gunter, C.A.: Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 2421–2434 (2017)
35. Wilke, L., Sieck, F., Eisenbarth, T.: Tdxdown: Single-stepping and instruction counting attacks against intel tdx. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 79–93 (2024)
36. Wilke, L., Wichelmann, J., Rabich, A., Eisenbarth, T.: Sev-step a single-stepping framework for amd-sev. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 180–206 (2024)
37. Xu, Y., Cui, W., Peinado, M.: Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In: 2015 IEEE Symposium on Security and Privacy. pp. 640–656 (2015)
38. Zhang, R., Gerlach, L., Weber, D., Hetterich, L., Lü, Y., Kogler, A., Schwarz, M.: Cachewarp: Software-based fault injection using selective state reset. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 1135–1151 (2024)
39. Zhang, Z., Tao, M., O'Connell, S., Chuengsatiansup, C., Genkin, D., Yarom, Y.: Bunnyhop: Exploiting the instruction prefetcher. In: USENIX Security. pp. 7321–7337 (2023)