EnTurbo: Accelerate Confidential Serverless Computing via Parallelizing Enclave Startup Procedure

Yifan Zhu, Peinan Li*, Yunkai Bai, Yubiao Huang, Shiwen Wang, Xingbin Wang, Dan Meng and Rui Hou
Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
*Corresponding author is Peinan Li, lipeinan@iie.ac.cn

ABSTRACT

Serverless computing has gained widespread attention, and Trusted Execution Environments (TEEs) are well-suited for safeguarding user privacy. However, the additional startup procedure introduced by TEEs imposes considerable performance overhead on confidential serverless workloads. This paper introduces a novel parallelized enclave startup design, EnTurbo, which eliminates the integrity dependence of the enclave startup procedure, accelerating it while ensuring its security. Additionally, EnTurbo parallelizes the measurement procedure, enabling multi-thread measurement for acceleration with provable security. We evaluate EnTurbo by running confidential serverless workloads on SGX simulation mode. Results show that EnTurbo effectively speeds up enclave serverless by 1.42x-6.48x (SGXv1) and 1.33x-3.76x (SGXv2).

ACM Reference Format:
Yifan Zhu, Peinan Li*, Yunkai Bai, Yubiao Huang, Shiwen Wang, Xingbin Wang, Dan Meng and Rui Hou . 2024. EnTurbo: Accelerate Confidential Serverless Computing via Parallelizing Enclave Startup Procedure. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3658492

1 INTRODUCTION

Serverless computing has become prevalent in cloud service providers, such as AWS Lambda[1] and Google Cloud Functions[2], due to the convenient management of cloud infrastructure and software resources. With this technique, users can focus on the development of core functions without the need to pay for other services, including resource management or maintenance tasks. Exploiting the Trusted Execution Environments (TEE)[3, 4], confidential serverless computing is proposed to provide trusted serverless services, enabling users to protect their private codes and data.

However, various security mechanisms are deployed in TEEs to guarantee the confidentiality and integrity. Take Intel SGX [3, 5] as an example, the lifecycle of an enclave involves a sequence of operations: *Creation*, *Copying*, *Measurement*, *Verification*, and *Execution*. The procedure introduces high startup overhead due to time-consuming measurement and verification mechanisms before execution. In confidential serverless computing scenarios, this overhead is unacceptable since it induces higher costs.

To address this problem, it is promising to find a method to accelerate the startup procedure. We evaluated typical confidential serverless workloads [1, 6, 7] to breakdown their performance during both startup and execution (Figure 1). There are two interesting

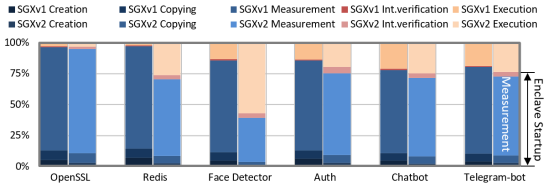


Figure 1: Normalized breakdown of confidential serverless workloads.

observations: ***1) Integrity dependence:*** The execution of enclave depends on its integrity, which is fulfilled by the "*Measurement*, *Verification*" operation sequence. This dependence leads to the enclave startup time constituting significant portion (76%-97% for SGXv1 [3] and 43%-97% for SGXv2 [5]) of a workload's lifecycle. ***2) Serialized Measurement:*** The measurement phase, involving time-consuming serial hash sequence of each enclave page, accounts for around 85.1% of the enclave's startup time on average.

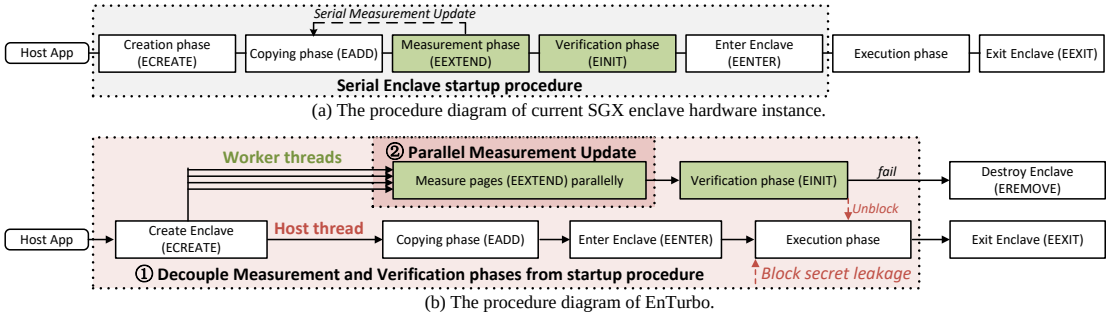
To mitigate the overhead for confidential serverless computing, some approaches have been proposed [8–10]. These approaches deploy snapshots or shared libraries to preload reusable content into an enclave, which remains influenced by the integrity dependence and the compute-intensive serialized measurement. when a new serverless request inconsistent with the existing pre-loaded content, it still suffers from the time-consuming startup procedure.

In this paper, we explore strategies to accelerate the startup procedure. Fortunately, there are ***two noteworthy insights for acceleration:*** 1) An enclave's execution does not rely on its integrity verification result, allowing decoupling the integrity dependence of the enclave startup procedure for parallelization at the premise of security. 2) The serial measurement procedure can be replaced with a less time-consuming paralleled measurement procedure if it does not compromise security. Motivated by these insights, we propose EnTurbo, a novel design aimed at accelerating confidential serverless computing through the parallelization of the enclave startup procedure. Our contributions are outlined as follows:

- EnTurbo eliminates the integrity dependence of enclave startup procedure, allowing the enclave to execute in parallel with the "Measurement, Verification" operation sequence. Additionally, it deploys a multithread measurement for further acceleration.
- We address the potential risks introduced by the decoupled startup procedure. (1) Eliminate the conflicts of the overlapped primitive operation to guarantee the correctness of integrity verification. (2) Restrict the potential information leakage before integrity is verified.
- We explore the suitable parallelization strategy of multithread measurement and demonstrate its security is equivalent to SGX through a cryptographic proof.
- We present a viable hardware implementation of EnTurbo on SGX and evaluates that the performance improvements of confidential serverless workloads are 1.42x-6.48x in SGXv1 and 1.33x-3.76x in SGXv2.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3658492



(a) The procedure diagram of current SGX enclave hardware instance.

(b) The procedure diagram of EnTurbo.

Figure 2: The different procedure diagrams of an enclave's main primitive operation sequence between SGX and EnTurbo.

2 UNDERSTANDING STARTUP PROCESS OF CONFIDENTIAL SERVERLESS COMPUTING

Serverless computing is well recognized by cloud users due to flexible scheduling and auto-scaling features [11]. Deploying TEEs to protect the privacy of serverless users is considered as a promising solution [4, 8–10]. The predominant challenge in this paradigm is the startup time of each workload.

2.1 Startup Procedure of Intel SGX

A typical execution process of an SGX enclave is depicted in Figure 2 (a). In the *Creation phase*, CPU executes the ECREATE primitive to establish a new Enclave Control Structure (SECS), which maps virtual address of the enclave to an isolated Enclave Page Cache (EPC). During the *Copying phase*, CPU executes EADD to copy the binary into the EPC at a page granularity. Each page copied into the EPC undergoes measurement by the one-way hash algorithm SHA-256 [15] (EEXTEND) at the *Measurement phase*. Subsequently, the measurement value is serially updated to MRENCLAVE structure in the SECS. This structure, along with other signatures, is utilized by EINIT for integrity *Verification*. Finally, the host app calls the EENTER primitive to enable the Thread Control Structure (TCS) and enters the enclave at *Execution phase*.

2.2 Threat Model

Following the threat model of SGX [3], we assume that the attacker has the ability to control the Supervisor (OS). After verifying the integrity, the code and data in enclave is trusted. This paper excludes microarchitecture side-channel or covert-channel attacks [12], which can be addressed by a secure hardware design [13]. Denial-of-Service attack is also beyond the scope of this paper.

2.3 Observation

To understand the startup performance impact of TEEs, we conduct tests on existing confidential serverless within SGX enclaves and provide a breakdown in Figure 1. The experimental environment is detailed in §7.1, and the observations are as follows:

Observation 1: Enclave execution depends on integrity verification result, introducing extra startup overhead on confidential severless workloads. This execution-after-verification mechanism can protect enclave from being attacked by the OS. But such a integrity dependence accounts a lot of startup time of serverless workloads. The startup time of an enclave occupies the enclave serverless lifecycle around 86.43% in SGXv1, and 76.44% in SGXv2.

Observation 2: The Measurement procedure occupies the majority of the enclave startup procedure. The reason is that the EEXTEND primitive's limitation of measuring a 512-bit data chunk at a time, necessitating 16 times execution for one page [14]. Additionally, each page undergoes serial measurement due to strict hash dependence [15]. The test results show that the execution time for EEXTENDs is approximately 11.3 times longer than that of the EADD primitive (around 85.1% of enclave startup procedure).

3 ENTURBO: PARALLELIZING ENCLAVE STARTUP PROCEDURE

We present EnTurbo, a design for accelerating confidential serverless computing by parallelizing the enclave startup procedure. EnTurbo incorporates two key mechanisms: ① ***Decouple the "Measurement, Verification" phases from startup procedure, and enable the enclave to execute parallel with these phases.*** The results of integrity verification primarily impact the enclave's security rather than its execution efficiency. If the enclave's integrity is guaranteed at startup, "Measurement" and "Verification" can be decoupled as redundant phases. ② ***Redesign the existing measurement procedure with a parallelized method for further acceleration.*** The purpose of the measurement is to generate a unique value for integrity verification, which can be accelerated through parallelization while ensuring its security.

Implementation: EnTurbo accelerates enclave startup through deploying a multi-threading method. As shown in Figure 2 (b), there are two types of threads in EnTurbo during its lifecycle: (1) *Host thread* serves as the main thread, responsible for copying the trusted binary to the enclave, followed by its execution. (2) *Worker thread* acts as the assistant thread, measuring EPC pages and verifying their integrity. To further accelerate the measurement phase, multiple Worker threads are deployed to measure the content in parallel. Host thread does not need to wait for the Worker threads to finish measuring, but directly executing the trusted code. It only awaits the integrity verification result before exiting the enclave. Once integrity verification fails, EnTurbo destroys the enclave.

Challenge 1: Both Host and Worker threads contribute to the generation of measurement value, and the order of their execution affects the correctness of the integrity verification result. Any conflict between these two types of threads will make the measurement value unexpected, result in verification failures for enclave instances that should have started normally.

Challenge 2: Before the integrity of an enclave is verified, it is crucial to prevent attackers from manipulating the enclave to reveal secrets. The decoupling of the enclave startup procedure creates a potential threat that attackers can modify the enclave source binary to make it execute malicious code, thus exposing secrets before *Verification* phase.

Challenge 3: The redesigned parallelized measurement acceleration must maintain security. The multi-threading measurement changes the algorithm sequence of the original measurement.

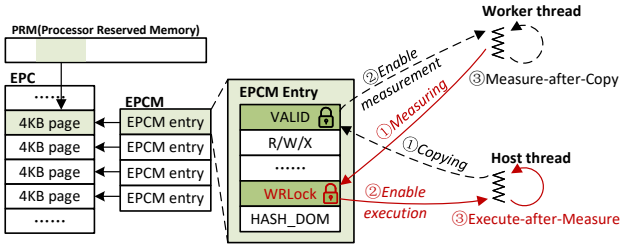


Figure 3: Deploy active flags to protect page access sequence. **The VALID flag activates Worker threads to access a page when it is enabled. The WRLock flag works before *Execution* phase, activating Host thread to enter the enclave when it is changed. The HASH_DOM is referred in §6.2.**

EnTurbo must guarantee that after reordering this sequence, its security is at least as robust as the current SGX.

4 CONFLICT-FREE INTEGRITY GUARANTEE

As discussed at *Challenge 1*, the paralleled execution of Host and Worker thread effects the measurement process. After thoroughly understanding this side effect, we explored the following two potential conflicts of both types of threads: ***(1) Input Conflict:*** The Host thread modifies enclave pages at *Copying* and *Execution* phase, disrupting the measurement input of the Worker thread. ***(2) Update conflict:*** Both Worker and Host threads update the measurement value to ensure the integrity of the startup procedure. If these conflicts occur, the measurement value will be incorrect.

To address *Input Conflict*, EnTurbo deploys a serial page access sequence. To address *Update Conflict*, EnTurbo makes these two types of threads update measurement value in different locations.

4.1 Guarantee "Copy-Measure-Execute" Page Access Sequence against Input Conflict

EnTurbo ensures a serial "Copy-Measure-Execute" page access sequence for Host and Worker threads during the parallelized enclave startup, eliminating *Input Conflict*. Depending on the types of enclave pages, EnTurbo adheres to *two rules* to ensure correct measurement: **(R1)** All pages must be copied before being measured. **(R2)** Writable pages cannot be modified at *Execution* phase before they are measured by Worker threads. Any deviation from these rules will result in an incorrect verification result.

As discussed in §2.3, conflicts related to (R1) occur infrequently, given that the *Measurement* phase takes considerably more time than the *Copying* phase. In contrast, conflicts related to (R2) occur frequently because the writable stack page is located in higher address space, measured later than others. To address these conflicts, we deploy different strategies that adhere to these rules, eliminating conflicts while maintaining marginal performance overhead.

(R1) ***Guarantee the Measure-after-Copy sequence for all pages.*** The EEXTEND in SGX startup avoids conflicts by checking whether the EADD sets the VALID bit of EPCM Entry. It will trigger exceptions if EnTurbo simply allows primitives to access the EPC simultaneously [14]. In Figure 3 (dashed line), EnTurbo reuses the VALID bit, deploying an interruptible polling logic in EEXTEND. This rule ensures the Worker thread waits for the VALID bit to be enabled before measuring a page, activating it to measure the page only when the VALID bit changes to 1b'1 by EADD.

(R2) ***Guarantee the Execute-after-Measure sequence for writable enclave pages.*** Firstly, EnTurbo reduces the probability of writable page modification conflicts via binary relocation. It relocates the enclave binary file layout to control the measurement order via trusted SDK, linking the writable pages to the lower continuous address space so they can be measured earlier. Secondly, EnTurbo introduces a 1-bit WRLock flag in the reserved field of EPCM entry, marking writable pages locate in lower addresses. In Figure 3 (solid line), the WRLock is traversed by the Worker thread to confirm that writable pages have been measured. As the enclave's virtual memory layout is continuous, when WRLock is detected to 1b'0, Worker thread modifies TCS, enabling the enclave to execute.

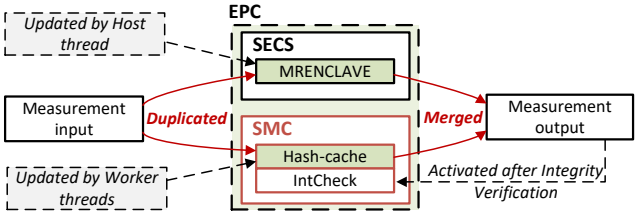


Figure 4: Design EPC structure for duplicate-merge mechanism. **The red box refers the new SMC structure. The Host thread updates the value in MRENCLAVE, while the Worker threads update it in Hash-cache. The IntCheck as the flag indicates whether integrity verification is successful.**

4.2 Duplicate-Merge Measurement Mechanism against Update Conflict

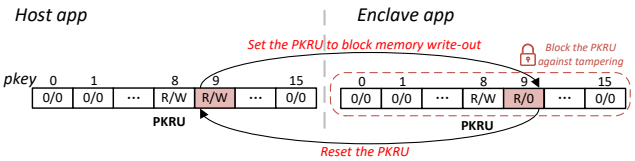
In SGX, both EADD and EEXTEND primitives access SECS atomically to guarantee that it is not exploited by attackers. If this atomic policy is simply removed for decoupling, the measurement value will deviate from expectation due to the out-of-order MRENCLAVE update. Therefore, *EnTurbo stores the intermediate hash results from Worker thread and Host thread separately in different location, and merge them after Measurement phase concludes.*

As shown in Figure 4, we introduce an additional EPC page named *Shadow Measurement Cache (SMC)* to store the status of the EnTurbo instance during the decoupled startup procedure. Within SMC, we define *Hash-cache* data chunk to duplicate the initial value of MRENCLAVE in SECS. At measurement phase, *Hash-cache* stores the intermediate hash results of Worker threads. Concurrently, as Host thread copies TCS or other EPC pages, its hash values are updated to MRENCLAVE structure in original SEC. When measurement phase is ended, the SMC page itself hashed by Worker threads updates MRENCLAVE. Finally, the *IntCheck* flag in SMC stores the integrity verification result at the Verification phase.

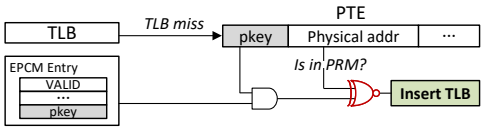
5 SECURE EXECUTION BEFORE INTEGRITY VERIFICATION

As for the *challenge 2* discussed in §3, there is an extended threat model after decoupling the startup procedure: ***The enclave content before Verification is untrusted.*** An attacker can maliciously manipulate the enclave to execute the modified binary, revealing secrets before the measurement output is verified.

Potential Attack Vectors: We analyze the interaction way between an enclave and the outside to explore potential attack vectors before enclave's integrity is verified. Specifically, the following vectors might be exploited by attackers: **(A1)** Directly disclosing secrets to outside via inserting spy memory access instructions. **(A2)** Maliciously tampering with the OCALL event of the enclave, disclosing secrets to external untrusted library via registers. **(A3)** Leaking secrets through error codes in Asynchronous Enclave Exit (AEX) events or exploiting control flow to maliciously construct exceptions [16]. EnTurbo deploys defense strategies, which only work before the *Verification* phase is completed, against these vectors.



(a) Enable primitive to control the PKRU register when context switches. In enclave app, R/0 bit refers that the non-PRM memory domain (pkey=9) is not writable.



(b) Add XNOR logic to microcode, denying OS change the *pkey* to non-PRM.

Figure 5: Employ Memory Access Protection to support security of EnTurbo.

5.1 Block Write-Out via Memory Access Protection

To defend against **(A1)**, *EnTurbo restricts memory write-out before the Verification phase.* We leverage Intel Memory Protection Key (MPK), a common solution in prior works [17, 18]. The user process controls memory access permissions via the protection key *(pkey)* of Page Table Entry (PTE) by configuring the *PKRU* register. After entering the enclave, EnTurbo blocks memory write-out through MPK. However, because the enclave is untrusted until integrity verification completes, EnTurbo cannot simply reuse prior solutions to address threat model conflicts between MPK and SGX. There are potential countermeasures to bypass the MPK isolation: **(A)** The malicious host app alters the MPK configurations. **(B)** The tampered enclave code changes *PKRU*, making non-PRM (Processor Reserved Memory) writability. **(C)** The OS changes the *pkey* corresponding to non-PRM to make it consistent with the enclave's. EnTurbo deploys corresponding mitigative mechanisms:

(A) ***Enable secure primitives to configure the MPK.*** After Host thread reads the address and offset in PAGEINFO structure, each EADD primitive sets the corresponding *pkey* in EPCM entries linked to the newly allocated EPC page, bypassing the control of untrusted host app and OS. As shown in Figure 5 (a), EnTurbo embeds additional PKRU setting instructions into all user-mode primitive that may enter or exit the enclave (e.g. EENTER and ERESUME), making it switch protection domain between the host app and enclave.

(B) ***Hardware-based protection against PKRU tampering.*** Considering the defense method of SGXLock [18], which detects PKRU configuration instruction stored in enclave's binary. However, this defense method is not effective in our threat model, as attackers can modify the detector code. Therefore, we add a write-lock to control PKRU. As long as the Host thread enters the enclave, the PKRU, exclusively configured by the trusted primitive, remains unmodifiable until enclave exits or the *IntCheck* flag is enabled.

(C) ***Additional memory access checking against pkey tampering in PTE.*** As shown in Figure 5 (b), we add an *exclusive-NOR (XNOR)* check in the memory address translation process. After a TLB miss, the virtual address of this memory request is translated to a physical address. Then, the memory access logic is taken over by FSM page walker. When the FSM detects that the address belongs to a non-PRM page, but the *pkey* is consistent with an enclave, it blocks the access. Therefore, the OS cannot bypass the MPK via tampering with the *pkey* of external PTE.

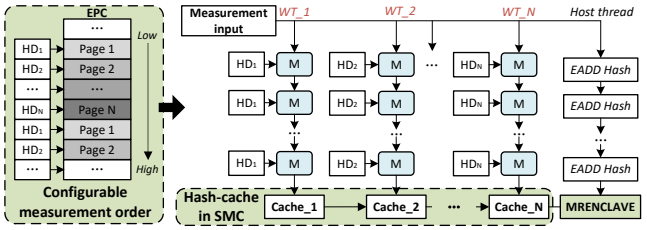


Figure 6: Process of parallel measurement. The "M" box refers the measurement operation on a page. The "HD_N" box marks an allocated page not measured yet, where HD_N refers the corresponding HASH_DOM.

5.2 Avoid Information Leakage from Proactively-Constructed Events

To defend against **(A2)** *and* **(A3)**, *EnTurbo thwarts the AEX or OCALL events, hiding that whether these events are triggered or not.* Moreover, it ensures that information cannot transmit via the events. When an event occurs, information is stored in SSA. Then, EnTurbo halts the event operation by HLT instruction [14] until another external interrupt occurs, protecting secrets. Once a normal external interrupt occurs during HLT, AEX reports the latest external interrupt to jump to the corresponding entry for handling. When the context is switched to enclave, ERESUME reads the original SSA information and *IntCheck* in *SMC*. If ERESUME checks that enclave's integrity has not been verified and exception has not been handled yet, the CPU continues executing HLT. Therefore, EnTurbo avoids leaking secret by triggering events on purpose.

6 PARALLEL MEASUREMENT WITH PROVABLE SECURITY

As detailed in §2.3, most of the startup process time is dedicated to the *Measurement* phase due to the serial hashing. To overcome *Challenge 3*, EnTurbo reorders the measurement sequence, parallelizing it while ensuring its security. As shown in Figure 6, EnTurbo divides enclave pages into multiple blocks and makes these block measured in a configurable order simultaneously. Then, the measurement value of every block, along with the hash result of EADD sequence, are hashed into MRENCLAVE as the output (§4.2).

6.1 Cryptographic Proof

The current SGX measurement employs a non-linear SHA-256 hash in sequence, providing robust anti-collision ability [15]. Attackers can hardly modify the content of a page to construct attack code while ensuring that its hash result remains unchanged. After altering the order of enclave measurement, the original serial non-linear hashing is broken , its security needs to be revisited.

Problem Formulation: Our parallel measurement algorithm reuses the SHA-256 hash algorithm foundation. We aim to prove that the parallel hashing is not weaker than current SGX. Considering the input is P1, P2, and the output is M, the corresponding SHA-256 is defined as: $M = H(P_1||P_2)$. We set the hash as an atomic operation, assuming that the total number of data chunk need to be hashed is N. The current SGX measurement output is:

$$M_{SGX} = H(P_1||P_2||\ldots||P_{N-1}||P_N) \tag{1}$$

In the measurement phase, the final measurement output of every Work threads is transformed into a sequence of 64-byte hashes. Assuming 4 Worker threads are deployed for acceleration, after parallelization, the measurement output is:

$$\begin{aligned} M_{parallel} =& H[H(P_1||\dots||P_{N/4})||\dots||H(P_{1+3N/4}||\dots||P_N)] \\ =& H(Q_1||Q_2||Q_3||Q_4) \end{aligned} \tag{2}$$

Security proof: Employing a proof by contradiction, we aim to demonstrate the robust anti-collision ability. Assuming that there is an attacker who can compromise our parallel algorithm (*equation* 2), constructing a set of input $\{q1, q2, q3, q4\}$, such that:

$$M_{parallel} = H(q_1||q_2||q_3||q_4) = H(Q_1||Q_2||Q_3||Q_4) \tag{3}$$

Then, for any hash initial state X, there is:

$$H(X||q_1||q_2||q_3||q_4) = H(X||Q_1||Q_2||Q_3||Q_4) \tag{4}$$

According to *equation* 4, this attacker also has the ability to construct an specific hash sequence (e.g. $\{K_{N-3}, K_{N-2}, K_{N-1}, K_N\}$) to replace the last 4 hash sequence of the *equation* 1, so that original M_{SGX} hash will also under the collision attack:

$$\begin{aligned} M_{SGX} =& H(P_1||\dots||P_{N-4}||P_{N-3}||P_{N-2}||P_{N-1}||P_N) \\ =& H(P_1||\dots||P_{N-4}||K_{N-3}||K_{N-2}||K_{N-1}||K_N) \end{aligned} \tag{5}$$

In conclusion, if there is an attacker who can attack the parallelized algorithm of EnTurbo, he can also attack the current SGX measurement algorithm. This kind of attacker does not exist in the current SGX threat model. Therefore, our measurement algorithm is cryptographically equivalent to the current SGX.

6.2 Profiling to Fit Multi-thread Acceleration

Explore measurement order per Worker thread to guarantee the correctness of integrity verification. To ensure the correctness of measurement value, the multi-thread acceleration must guarantee that each Worker thread measures its page block in a predefined order (Figure 6). We introduce a profiling method to fit this parallelization: The SDK, customized for EnTurbo, adds a *HASH_DOM* flag stored in the reserved field of each EPCM Entry, dividing the content into distinct blocks. Worker threads execute EEXTEND simultaneously to match the corresponding HASH_DOM when measuring the target block. If the page does not align with the designated block, the Worker thread skips it and proceed to the next page until it finds the corresponding block for measurement.

Explore the number of worker threads by empiricism. The bitwidth of HASH_DOM is determined by the hardware features of SGX platforms. In our experimental environment, the accommodated number of Worker threads is 8 (see §7.2 for details).

7 EVALUATION

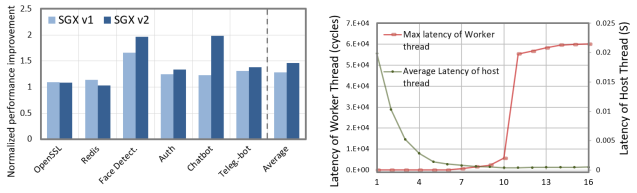
7.1 Experimental Setup

Intel SGX is not open source for direct microcode modification. Similar to the related work [8, 17], we employed the simulation mode of the Intel SGX SDK to emulate the hardware and primitive design matched the new attribute we proposed.

Primitives emulation: Existing SGX SDK lacks support for EEXTEND and EINIT to achieve measurement and verification in simulation mode. Therefore, we deploy SHA-256 hash algorithm into SDK to emulate EEXTEND, and deploy a dedicated data structure store in binary to emulate enclave access to the *SMC*. Table 1 outlines other non-intrusive extensions made by primitives in the SGX simulation mode.

Table 1: Primitive extensions introduced by EnTurbo.

Operations	Brief Descriptions of the Extended Operation
EADD	Set up the *pkey* of enclave in every EPCM Entry
EINIT	Enable the *IntCheck* flag if integrity verification is done
EENTER,EEXIT	Inserted the WRPKRU to change the PKRU access permission
ERESUME	Inserted HLT, WRPKRU and the SSA checking instructions
AEX	1. Inserted HLT, WRPKRU instructions 2. Changed the exception information transmission mode
EEXTEND	1.Check the flags (WRLock, HASH_DOM) in EPCM 2.Enable the TCS if WRLock flag is 1b'0 3.Measuring the page content in *SMC*



(a) Performance improvement. (b) Latency of Host and Work threads.

Figure 7: Evaluating the decoupled startup procedure.

Hardware behavior emulation: EnTurbo's hardware emulation mainly focuses on the existing address translation mechanism. In current design, the TLB miss handling process causes an additional check on PTE and EPCM Entry by the FSM. To emulate this additional check in the TLB miss events, a negligible checking overhead (less than 10 cycles [8]) are introduced.

Platform and benchmark: We run experiments on Intel Core i7-10700 and Xeon Platinum 8380 CPU which supports SGXv1 and SGXv2, respectively. The software environment setups Ubuntu 20.04, Linux 5.19 Kernel. As shown in Figure 1, we choose the encry/decryption benchmark OpenSSL [6], the memory-intensive benchmark Redis [7] and privacy preserving serverless workloads [1]. For benchmarks requiring a non-C runtime (e.g., Python), the existing libOS Gramine [19] is deployed to support their execution in the SGX enclave, and their performance is measured. The baseline for comparison is the current SGX simulation mode.

7.2 Performance Evaluation

To evaluate the performance introduced by EnTurbo, we conduct separate evaluations of the two key mechanisms and perform a comprehensive experiment. Specifically, we insert *RDTSC* instruction into primitives and the enclave app to measure the execution time, revealing the impact on startup latency of EnTurbo.

Performance improvement of the decoupled enclave startup. After EnTurbo decouple the "Measurement, Verification" phases from startup procedure, the ideal effect is saving all the time of these phases. The evaluation involves deploying a Worker thread and a Host thread to run these benchmarks. Figure 7 (a) shows the acceleration effect, demonstrating that the average acceleration can reach up to 1.28x and 1.46x faster than the original SGXv1 and SGXv2, respectively. For the cases of OpenSSL and Redis, which require OCALLs to load shared library, the acceleration is not fully effect. In these cases, the effect after decoupling startup is close to the normal SGX startup procedure, with negligible side effects on startup efficiency. In SGXv2, the performance improvement introduced by EnTurbo is more effective. This is because the Enclave Dynamic Memory Management moves the heap/stack page allocation procedure into *Execution* phase, shortening measurement procedure.

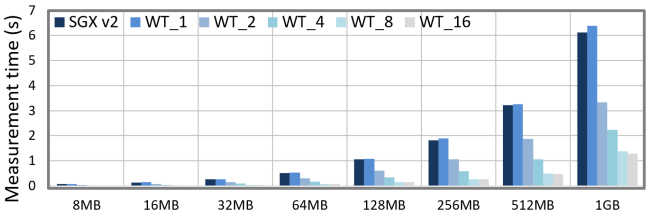


Figure 8: The measurement time of different enclave sizes affected by different numbers of Work threads (WT_(1–16)).

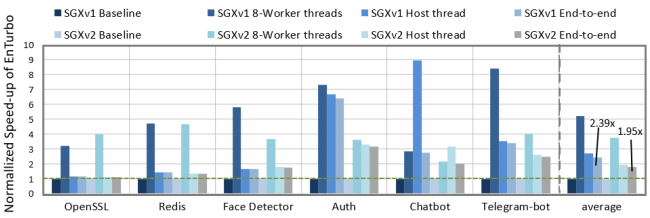


Figure 9: Performance evaluation of the EnTurbo.

The more balanced the execution time allocated between these two types of threads, the faster they will execute after decoupling.

Explore the suitable number of Worker threads for Parallel Measurement. EnTurbo pre-defines the recommended number of threads for parallelization in different TEE platforms. We deploy a Host thread along with 1 to 16 Worker threads to explore the ideal thread number to accelerate the startup procedure. The number of Worker threads is mainly determined by two factors:

Firstly, we explore the Worker thread number to minimize the overall latency of these two threads due to the Conflict-Free Integrity Guarantee (§4). Figure 7 (b) shows that when the number of Worker threads is beyond 10, the max latency has exceeded its own execution time, because the speed of copying a page by the Host thread does not match the Worker threads. In such a condition, the Host thread latency is reduced into a reasonable period because of the accelerated measurement on writable page.

Secondly, we explore the number of Worker threads to accelerate measurement phase without wasting unnecessary hardware resources. As shown in Figure 8, the speed of measurement phase is an average 1.79x, 3.15x, 6.06x, 7.15x times faster than the current SGX at the 2x, 4x, 8x, and 16x numbers of Worker threads, respectively. When the number of Work threads exceeds 8, the cost performance of each thread decelerates. The reason is that the higher the number of threads, the greater the impact on communication latency (e.g. Cache miss, context switching). Therefore, in our experimental environment, the recommended number of Worker threads is [8,10].

Put it all together. We perform an extensive performance evaluation of the entire design. We configure 8 Worker threads for measurement. As depicted in Figure 9, the results illustrate that the end-to-end execution speed of different cases are improved 1.42x-6.48x on SGXv1 and 1.33x-3.76x on SGXv2, respectively.

7.3 Memory and Hardware Evaluation

Memory overhead: The additional *SMC* page introduced by EnTurbo in the enclave occupies 4KB of data. This is due to the *HashCache* structure in it, which stores 512-bit temporary data of every Worker thread (maximum number of it is 16). Such a design marginally imposes overhead on enclave memory space.

Hardware overhead: EnTurbo introduces a lightweight microcode change similar to related works [17, 18], making it easily updatable. The *WRLock* flag and *pkey* utilize reserved bits in EPCM and SECS, respectively. These hardware costs are also negligible.

8 RELATED WORK

Confidential serverless startup acceleration. Inspired by cold-start acceleration efforts in serverless computing [11], startup acceleration design to the confidential serverless workload has also garnered attention. Penglai [9] proposes a template-based enclave, allowing users to fork a trusted template known as the *Shadow Enclave*. PIE [8] presents a *plug-in enclave* that facilitates the sharing of trusted libraries, enabling serverless workloads to initialize quickly without loading libraries during enclave startup. Reusable enclave [10] supports rapid reset and reuse of enclaves through a snapshot-based approach. However, both snapshots and templates must adhere to the serial primitive operation sequence when initially created. EnTurbo is orthogonal to these works, further reducing the confidential serverless startup time.

9 CONCLUSION

To address the execution efficiency of confidential serverless workloads, we introduced a parallelization design known as EnTurbo. Firstly, EnTurbo enables the enclave to initiate execution in parallel with the integrity verification process while maintaining its security. Secondly, EnTurbo parallelizes the measurement process by proposing an alternative but still secure method. With these two mechanisms, EnTurbo significantly accelerates the startup of confidential serverless computing.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant Nos. 62125208 and 62202467. The corresponding author is Peinan Li.

REFERENCES

[1] Amazon AWS,"AWS Lambda," https://aws.amazon.com/lambda/
[2] Google,"Google Cloud Functions," https://cloud.google.com/functions/
[3] V.Costan et al.,"Intel sgx explained," in Cryptol, IACR, 2016, pp.1–118.
[4] F.Alder et al.,"S-faas: Trustworthy and accountable function-as-a-service using intel sgx," in CCSW'19, ACM, 2019, pp.185–199.
[5] F.McKeen et al.,"Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave," in HASP, ACM, 2016, pp.1–9.
[6] Intel, "SGX-Openssl," https://github.com/intel/intel-sgx-ssl/
[7] "Redis," https://github.com/redis/redis
[8] M.Li et al., "Confidential serverless made efficient with plug-in enclaves," in ISCA, IEEE, 2021, pp. 306–318.
[9] E.Feng et al., "Scalable Memory Protection in the PENGLAI Enclave," in OSDI, USENIX, 2021, pp. 275–294.
[10] S.Zhao et al., "Reusable enclaves for confidential serverless computing," in USENIX Security'23. 2023. pp. 4015–4032.
[11] Z.Li et al., "The serverless computing survey: A technical primer for design architecture," in ACM Computing Surveys (CSUR). 2022. pp. 1–34.
[12] A.Aldaya et al., "Port contention for fun and profit," in S&P'19, IEEE,pp.870–887.
[13] M.Taram et al., "Secsmt: Securing SMT processors against contention-based covert channels," in USENIX Security 2022, pp. 3165–3182.
[14] Guide, Part,"Intel® 64 and IA-32 Architectures Software Developer Manuals".
[15] Q.Dang, "Secure hash standard," https://doi.org/10.6028/NIST.FIPS.180-4
[16] Y. Xu et al., "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in S&P'15, IEEE, 2015, pp. 640–656.
[17] J.Gu et al., "A {Hardware-Software} Co-design for Efficient {Intra-Enclave} Isolation," in USENIX Security 22, 2022, pp. 3129–3145.
[18] Y.Cheng et al., "SGXLock:Towards Efficiently Establishing Mutual Distrust Between Host Application and Enclave for SGX," USENIX Security 22,2022.
[19] "Gramine Library OS with Intel SGX Support," in https://github.com/gramineproject/gramine.