Cryonics: Trustworthy Function-as-a-Service using Snapshot-based Enclaves

Seong-Joong Kim
The Affiliated Institute of ETRI & KAIST
sungjungk@kaist.ac.kr

Myoungsung You
KAIST
famous77@kaist.ac.kr

Byung Joon Kim
The Affiliated Institute of ETRI
bjkim@nsr.re.kr

Seungwon Shin
KAIST
cloude@kaist.ac.kr

ABSTRACT

Recent research has proposed the use of trusted execution environments (TEEs), such as SGX, in serverless computing to safeguard against threats from insecure system software, malicious co-located tenants, or suspicious cloud operators. However, integrating SGX, one of the most mature TEE, with serverless computing results in significant performance degradation due to the function startup latency caused by enclave creation. This performance degradation arises because SGX is not designed with serverless function startup procedures in mind, where numerous application codes, libraries, and data are re-initialized upon each function invocation. The inherent limitations of SGX contribute to significant performance degradation, whether through the addition of every page into the enclave, or the restriction of page permissions, which ultimately cause TLB flushes, context switches, and re-entering the enclave. In this paper, we first take key observations resident in the intrinsic features of the serverless function and propose Cryonics, a method of serving snapshot-based enclave that accelerates the startup time of the function instance by creating a future-proof working set of that. We consider the page locality and obsolete pages of the enclaved function instance to create a lightweight working set used for serving requests. Our evaluation shows that Cryonics achieves up to 100x outperformed startup time compared to existing cold-start-based methods and reveals the stability of the startup time.

SoCC '23, October 30–November 1, 2023, Santa Cruz, CA, USA

ACM ISBN 979-8-4007-0387-4/23/11.
https://doi.org/10.1145/3620678.3624789

CCS CONCEPTS

• **Security and privacy** → *Distributed systems security*.

KEYWORDS

Serverless Computing, Trusted Execution Environment

ACM Reference Format:
Seong-Joong Kim, Myoungsung You, Byung Joon Kim, and Seungwon Shin. 2023. Cryonics: Trustworthy Function-as-a-Service using Snapshot-based Enclaves. In *ACM Symposium on Cloud Computing (SoCC '23), October 30–November 1, 2023, Santa Cruz, CA, USA*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3620678.3624789

1 INTRODUCTION

As cloud services grow in scale, serverless computing, with its Function as a Service (FaaS), has emerged as a popular option in the cloud environment. Serverless computing allows developers to construct large-scale applications via compact, ephemeral functions activated by specific events. Furthermore, serverless computing employs a utility billing model, thereby charging developers solely during the active duration of their functions. Owing to its simplicity and economic benefits, leading cloud service vendors such as Google, AWS, and Azure, are vigorously introducing their proprietary serverless computing platforms [18, 27, 36].

Serverless applications often incorporate functions that handle privacy-sensitive data, such as online payment functions processing users' credit card information or face recognition functions receiving users' biometric data. Moreover, in serverless environments, function providers lack visibility into the underlying server infrastructure, leaving them reliant on the cloud provider's security measures for data protection and function security. Given these challenges, there is a growing need to safeguard user privacy in serverless environments against vulnerable functions [28–30], malicious tenants [16], and even untrustworthy cloud insiders. Trusted execution environments (TEEs), like Intel SGX [26], can offer hardware-based, robust isolation (referred to as an

enclave) that separates user processes (and sensitive data) from the rest of the system, including the operating system. Consequently, TEEs are regarded as a promising approach for enhancing the overall security of serverless platforms and facilitating privacy-preserving serverless functions.

Unfortunately, existing SGX designs are not well-suited for serverless workloads. In our preliminary experiment (Section 3), we port a typical serverless function into an in-house enclave using a library OS [46] and measure the function's startup time. The results reveal a significant increase in startup latency even when the function's memory footprint is small, with the latency growing exponentially as the function size expands. The root cause of this issue stems from the intricate enclave initialization process (e.g., memory copy, encryption, measurement, and permission adjustment) and the limited enclave size, resulting in frequent enclave eviction during function startup. Even with less limiting the enclave size and supporting dynamic memory management, i.e., SGX version 2, it requires a considerable period of spawning a function instance to make an enclave itself and various 3rd party library initialization. Given the nature of serverless computing, where multiple functions are re-initialized upon each invocation, the increased function startup time poses a challenge for the deployment of current SGX designs in serverless computing. Moreover, although some previous studies [2, 6, 25, 44] have endeavored to address this issue, they continue to face critical limitations, such as supporting only specific serverless function types [6] and necessitating modifications to the underlying SGX hardware [25].

Our approach. To overcome these limitations, this paper adopts snapshotting for serverless functions, an approach gaining increasing prominence in recent studies [14, 47]. This method addresses the challenges of the inevitable surge in startup latency inherent in cold starts, as well as the wasteful resource consumption resulting from the continuous occupation of scarce resources in warm starts. Although prior studies focus on unenclaved (encrypted) functions, we extend snapshotting to consider SGX context and page transition between an enclaved function and the regular memory.

This paper introduces Cryonics, a novel snapshot-based SGX enclave spawning system tailored for serverless functions. It enclaves a function instance within an Enclave Page Cache (EPC), consequently generating a lightweight and efficient snapshot of the enclaved function. Instead of establishing a new enclave and re-initializing the function instance, Cryonics redeploys the previously created snapshot upon function invocation. This strategy ensures the protection of the function instance within the enclave while significantly minimizing the function startup latency. Cryonics integrates two snapshot optimization techniques: page recycle tracking and obsoleted page identification. These methods decrease the snapshot size and enhance loading speed by maintaining only the essential pages within a snapshot that are anticipated to be referenced frequently during the operation of the function. During the loading of a snapshot, Cryonics employs a proactive snapshot reloading mechanism. This method loads multiple pages in a snapshot, expecting a future-proof working set, all together into the enclave area, thereby eliminating repetitive enclave exits and re-entries. Consequently, it mitigates the inefficiency of loading every page within a snapshot using the page fault handler.

We conducted an exhaustive evaluation of our system utilizing a diverse range of serverless workloads, generated by FunctionBench [21]. The results demonstrate that Cryonics, while utilizing SGX to safeguard these functions, accomplishes a function startup time that is up to 100 times faster than existing methodologies for a wide array of function types. Furthermore, the two snapshot optimization techniques we introduced notably reduced the size of the snapshot by as much as 80%, and successfully incorporated the majority of the pages necessary for the function's operation within the snapshot. In combination with serverless computing and SGX, the proposed Cryonics design aims to achieve the fast startup of a function securely.

Contributions. the contributions of this paper include:

(i) A comprehensive study on the performance of serverless computing under the enclave: figure out the root cause of bottleneck point lies in the SGX design.
(ii) Designing for fully aware of serverless and SGX model: make an SGX enclave-based snapshot composed of selected pages for fast serving the response by exploiting the intrinsic features of serverless computing.
(iii) Performance improvement with reducing startup latency: improve 10-100x startup latency by carving a future-proof working set to serve the function.

To the best of our knowledge, there is no prior research to extend snapshot-based serverless function with the enclave.

2 BACKGROUND

In this section, we discuss the characteristics and challenges of Intel SGX and serverless computing, respectively.

2.1 Intel Software Guard Extension (SGX)

Intel SGX provides a secure approach for executing a user application on an untrusted system by utilizing a secure sandbox, referred to as an enclave. During the system booting process, SGX reserves a portion of the main memory region, known as the Processor Reserved Memory (PRM). The enclave memory, called Enclave Page Cache (EPC), is allocated from the PRM and safeguarded by the strict access control model of SGX. The data and code within an EPC can only be accessed by the owning enclave and remain inaccessible and invulnerable to tampering by other enclaves or

processes, including privileged system software. When an application within an enclave requires more memory than the maximum size allowed, SGX evicts an EPC from the PRM to the untrusted memory region, but only after encrypting the EPC. The evicted EPC is reloaded back into the PRM when needed through a page fault mechanism.

While Intel SGX ensures both integrity and confidentiality of EPCs, the process of enclave creation, which involves EPC allocation and setup of necessary enclave metadata, requires time-intensive tasks (e.g., context switching and TLB flushing). Additionally, the size of the PRM is capped at 128MiB, with roughly 96MiB accessible for all enclaves. Consequently, when an enclave's memory requirement exceeds this limit, an EPC eviction process is unavoidable, thereby introducing extra overhead. To mitigate it, modern Intel processors support 512MiB EPC by giving up on integrity protection (only supporting confidentiality) [20]. The original version of SGX is referred to as SGX1, while the new one is SGX2.

2.2 Serverless Computing

Serverless computing [4] represents a new programming paradigm in cloud environments, enabling developers to construct large-scale applications by assembling a sequence of small, stateless, and ephemeral functions. A function serves as a single-purpose code segment designed to execute specific business logic. This function is triggered in response to predefined events (e.g., user requests and a message from another function) and is terminated after processing requests. While the serverless model simplifies the programming of complex applications, it may also impact application performance. The initialization of a function instance necessitates the loading of both function-specific code/data and associated function runtimes and third-party libraries. Given that this resource-intensive process repeats with each function invocation, the latency during function instance startup can potentially impede the overall performance of serverless applications. Recently, a warm start for a function has been proposed to reduce startup latency by spawning the function in advance or prolonging that activation persisting across invocations [11]. However, they cannot fully exploit the merits of serverless platforms that conserve computing resources.

Function snapshotting. In an effort to reduce the startup time of function instances, various studies [14, 31, 40, 47] have emerged in the field of FaaS. Among the myriad of studies, the snapshotting of a function [47] emerges as one of the most promising. A function snapshot essentially encapsulates the state of a function instance at a given point in time, including its code, data, and memory allocation (e.g., loaded pages). Once a snapshot is captured, it can be stored in memory or on storage, ready to be rapidly restored or replicated when the target function is invoked. This significantly reduces the function startup latency, proving particularly beneficial for functions that necessitate heavy init tasks.

Our work is inspired by this function snapshot technique. We have developed Cryonics, a novel snapshot-based SGX enclave for serverless functions. It enhances the protection of functions through the utilization of SGX, and provides fast startup time for enclaved functions by efficiently creating snapshots optimized for the enclaved functions.

Table 1: Time to spawn an enclave changing in size.

Heap Size (MiB)	4	8	16	32	64	128	256	512
Activated Pages	1779	2803	4851	8947	17139	23811	23763	23619
Loading Time (s)	0.033	0.047	0.084	0.297	0.936	7.489	16.123	32.973

3 MOTIVATION

In this section, we evaluate the feasibility of employing SGX in serverless computing for safeguarding functions by conducting experiments. We specifically measure the overhead of SGX enclave initialization and the startup time of enclaved serverless functions. The experiments are performed on an Intel Xeon E-2286M processor, which ensures the confidentiality and integrity of EPCs with a PRM size of 128MiB.

3.1 Overhead of Enclave Initialization

To safeguard a serverless function using SGX, the enclave creation procedure should precede the function's initialization. We first analyze the overhead associated with the process of enclave creation. We measure the time taken to create an enclave while incrementally increasing its memory footprint from 1MiB to 512MiB. Table 1 exhibits the result.

The time required for the creation of an enclave begins to increase linearly before observing an exponential increase at the 128MiB point, reaching an upper bound of 60 seconds. This poor performance is attributed to the frequent page evictions caused by the limited PRM size. Given the fact that the available size of the EPC is capped at 96MiB, the creation of an enclave that exceeds this limit invariably results in performance degradation due to page evictions. Serverless functions typically include an array of software components, such as language runtimes and third-party libraries, that are fundamental to their operations. As a result, their memory footprint is prone to exceed the EPC size provisioned by SGX hardware. However, an even more significant problem is the substantial overhead incurred solely by the enclave creation process, even when discounting page evictions. While the expected runtime of a serverless function is on the order of tens of milliseconds, the creation of a 64MiB enclave, even one that does not necessitate EPC evictions, consumes approximately 0.3 seconds. This is primarily because the initialization process of an enclave demands about 100K cycles

Table 2: Startup time of enclave-based function instance on various application types.

	Lighttpd	Nginx	Helloworld	Numpy
Enclave Size (MiB)	128-256			
Heap Size (MiB)	64-128			
Activated Pages	Saturated ≈ 23800			
Startup Time (s)	3.482	8.620	12.480	13.029

per page [26] due to the overhead of encrypting and verifying EPC pages. Consequently, the existing design of SGX is ill-suited to deployment in a serverless environment.

3.2 Startup Latency of Enclave-based Server

Subsequently, we evaluate the startup latency of SGX-enclaved serverless functions. Given the absence of serverless platforms that explicitly support SGX, we examine the startup latency when initializing applications (e.g., web servers and Python applications) akin to serverless functions within a library OS-based enclave (e.g., Gramine-SGX). Existing research [38] suggests that typical serverless functions have a memory footprint ranging between 170 and 400MiB. Consequently, for this experiment, we configure the enclave size for the applications at 256MiB. The experimental results are shown in Table 2. HTTP servers, namely Lighttpd and Nginx, exhibit a startup time of 3.3 sec and 8.6 sec, respectively. Python-based applications display a more extended startup latency, spanning from 12 sec to 13 sec. In the context of serverless functions, the expected startup latency for efficient service operations is in the order of tens of milliseconds. Thus, these results insinuate that the prevailing SGX design remains unsuitable for integration into serverless platforms.

The root cause of this poor performance is the inclusion of time-consuming enclave initialization tasks within the function initialization process. Note that the initialization of a single EPC requires 100K CPU cycles. Moreover, the memory footprint of typical serverless functions surpasses the available EPC size of SGX hardware, thereby leading to frequent EPC evictions that impose additional latency. In conclusion, leveraging the existing SGX enclave for serverless computing poses substantial challenges, necessitating the development of novel methods for mitigating the overhead during the initialization step.

4 CRYONICS DESIGN

4.1 Threat model

In this paper, we adopt the threat model utilized by existing SGX-based systems. We place our trust in the underlying hardware (e.g., CPU) and in-enclave code and data, which includes the library OS, application code, and third-party libraries. Conversely, we regard the following components as untrusted: (i) kernel, driver, hypervisor, and other system software, and (ii) code and data external to both the Intel CPU and the PRM. Our system, Cryonics, can be broadly divided into two phases: online and offline. During the online phase, Cryonics serves SGX-enclaved functions in response to user requests. In the offline phase, when no functions are served, and user requests are not accepted, Cryonics performs initialization procedures, such as generating the necessary metadata for enclaving function instances. We assume that all components of Cryonics are trustworthy during the offline phase.

We consider architectural side-channel attacks, controlled-channel attacks, and CPU vulnerabilities to be outside the scope of this paper[7, 8, 23, 50]. Additionally, denial-of-service vulnerabilities, which are common to all SGX-based approaches, are also beyond the scope of our work.

4.2 Design Requirements

In light of the challenges in adapting the existing SGX design to serverless platforms, we identify the following design requirements for our proposed system:

R1: Efficient EPC utilization: The current SGX design offers a limited number of EPCs to support memory integrity and confidentiality [12]. Consequently, the proposed system should safeguard multiple function instances in serverless workloads, which have large memory footprints, by efficiently utilizing the limited-sized EPCs.

R2: Hardware-level compatibility: The existing SGX design emphasizes the security implications of its underlying hardware. Additionally, alterations in hardware design, such as the introduction of extra instructions, can produce side effects that compromise the hardware's trustworthiness. Therefore, the proposed system does not require any changes to the existing SGX hardware and should maintain hardware-level compatibility with it while effectively protecting functions utilizing SGX hardware. In addition, the proposed system should also be able to operate on both SGX 1 and SGX 2.

R3: Application-level compatibility: Serverless computing reduces the burden of application developers by decomposing large-scale applications into modular, (i.e., functions). Requiring code-level modifications to these functions for security enhancements can undermine the benefits of serverless computing. As such, the proposed system needs to enable unmodified function instances to be protected by SGX and be quickly initialized within SGX enclaves upon invocation.

4.3 Overview

In this paper, we present Cryonics, a snapshot-based SGX enclave system for serverless computing, designed to enable

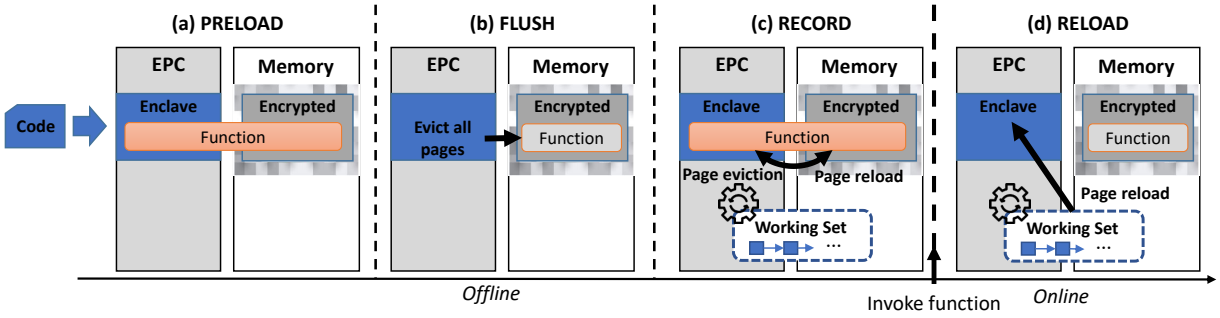

Figure 1: Overall procedure of the Cryonics system. It creates lightweight and future-proof snapshots for SGX-enclaved functions through the above four steps: (a) PRELOAD, (b) FLUSH, (c) RECORD, and (d) RELOAD.

rapid startup of enclaved serverless functions. Contrary to existing studies that perform resource-intensive SGX initialization tasks for function instances upon each invocation, Cryonics employs a preloading-based approach derived from observations of intrinsic features in serverless computing (see Section 2). Cryonics generates a snapshot of a function, which is a saved state of a function instance, including SGX-enclaved code and data, in advance and securely stores the snapshot in the regular memory area remaining encrypted. Upon function invocation, Cryonics quickly restores the saved snapshot for that function by reloading the corresponding EPC pages through a fast EPC reloading scheme, substantially reducing the startup latency of enclaved function instances. Additionally, Cryonics implements a novel page selection method that identifies only the required pages for function execution and excludes obsolete pages from the saved snapshot. This reduces the overall size of the snapshot to be allocated to enclaves, thereby enhancing the function instance's startup time and conserving the precious EPC.

Figure 1 delineates the process of constructing an SGX-based function snapshot in our system through four stages: (a) PRELOAD (b) FLUSH (c) RECORD, and (d) RELOAD. The first three stages occur during the offline phase (i.e., prior to initiating any function serving), while the final stage transpires in the online phase.

(A) PRELOAD: In this stage, Cryonics, establishes an SGX enclave to serve as the basis for the function instance. Following the creation of the enclave, Cryonics initializes the function instance using previously deployed program code from the serverless platform within the SGX enclave. The function initialization process entails the loading of numerous libraries required for the function service into the EPC. This can result in a burst of system calls requiring kernel-side handling and page eviction to overcome the limitations of the EPC space. However, as opposed to existing studies that repeat this heavy task for each function invocation, our system executes this task singularly. We achieve effective reuse of the created enclave by transforming it into a snapshot.

(B) FLUSH: This stage flushes all previously loaded pages of the function instance by evicting them to the regular memory region from the EPC area. To facilitate this, Cryonics initially halts the function operation, thereby ensuring the page set remains immutable by preventing any new entries into the enclave. Following the freezing of the function's operation, Cryonics securely evicts the page set located in the EPC to the untrusted memory region. We leverage the page eviction scheme to cleanse the EPC area during this process. As shown in Figure 1, all pages comprising the enclave are stored in an encrypted state within a regular memory area after this step.

(C) RECORD: This stage is designed to determine which of the evicted pages will be in active use during the operation of the function instance. Analogous to previous dynamic analysis, we generate a test workload and feed it to the function. During the operation of the function, the required pages are sequentially loaded in the EPC. Subsequently, Cryonics monitors the page in/out operation between the EPC area and the regular memory area to compile a list of activated pages (i.e., the working set in Figure 1). Additionally, it aims to create a lightweight working set by excluding obsolete pages, utilized solely during the function initialization, while retaining frequently accessed pages. Following the formation of the working set for the function, it flushes all the pages in the EPC area once more, awaiting function invocation with an empty EPC. Note that the created working set can be transferred to the disk to diminish memory usage, if desired.

(D) RELOAD: Upon receipt of a request by the serverless platform, which triggers the snapshotted function, Cryonics effectively restores the working set for the function using an optimized reloading mechanism (see Section 4.6). It preemptively inserts previously recorded pages (as per the RECORD stage) into the active page list maintained by the enclave instance, negating the need to raise a page-fault exception to reload the evicted pages. This significantly diminishes the overall snapshot reloading latency.

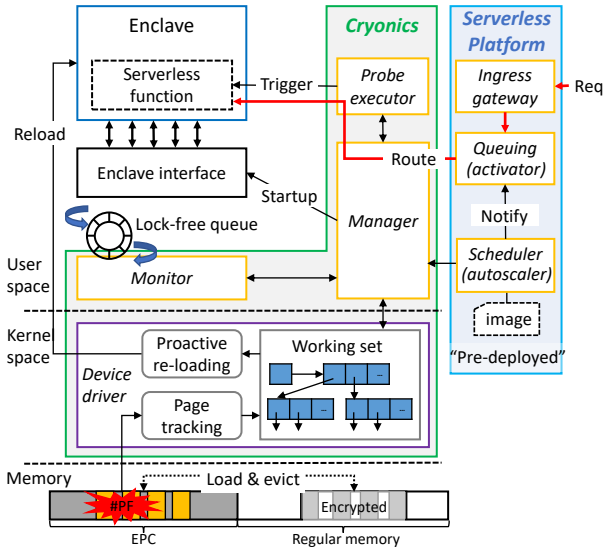

Figure 2: The system architecture of Cryonics comprises three user-level components (in orange) and one kernel-level component (in purple) that integrates with the existing serverless platform.

4.4 System Architecture

Figure 2 shows the Cryonics system architecture, consisting of three user-level and one kernel-level component. These components are specifically designed for the creation of an enclaved function snapshot and its efficient reloading in response to a function invocation.

The manager operates as the orchestrator of the other system components, controlling the processes of snapshot creation and reloading. In the snapshot creation phase, it initiates an enclave for the function, loading the function's code and data into the enclave area (PRELOAD). The monitor, responsible for overseeing the function's lifecycle, communicates a signal to the manager once the function is completely loaded and is ready to process requests. Upon receipt of this signal, the manager evicts all pages pertaining to the function from the enclave, moving them to the main memory (FLUSH). Subsequently, the probe executor generates a test workload for the function based on user specifications and feeds this created workload to the serverless platform, triggering the function that has been flushed. Now, the device driver, operating in the kernel space, logs the EPC page in/out operations of the function and tracks activated pages, generating a working set for the function (RECORD). Following this, the manager terminates the function and archives the created working set as a snapshot. Upon real invocation of the function, the manager directs the device driver to efficiently reload the saved snapshot (RELOAD). More detailed explanations of each component follow.

Manager. The manager serves as the central component of the Cryonics system, operating in the user space and directing the processes of snapshot creation and reloading. During the enclave creation for a serverless function, it utilizes the Gramine-SGX [46] loader to maintain compatibility with the function binary. This capability ensures that our system can load and run unmodified function binaries within an enclave, thereby allowing Cryonics to enhance the security of a function in a transparent way. Following the enclave's creation, the manager builds a snapshot for the function by interfacing with the monitor, the probe executor, and the device driver as previously outlined.

In addition, the manager can be integrated into the existing serverless platform to facilitate the reloading of a snapshotted function upon its invocation. Specifically, it can receive the metadata (e.g., function name) of the triggered function from the serverless platform. Upon receipt, it checks whether a snapshot exists for the function. If one does, the manager proceeds to reload the stored snapshot for the function, eliminating the need to launch the function from scratch. Conversely, if no snapshot is found, it reverts the function launch process back to the serverless platform, enabling the function's initiation without the assistance of Cryonics.

Monitor. The monitor is designed to infer the status of a function encapsulated within an enclave. For the execution of the FLUSH and RELOAD stages, Cryonics requires confirmation that the enclaved function is fully initialized and ready to process requests. However, even the SGX loader, responsible for managing the enclave's lifecycle, lacks the capability to monitor the state of an application (function binary) within the enclave.

One naive method for detecting the function's state is to allow ample time for the function to complete initialization, such as loading the requisite libraries. Nevertheless, this approach can prolong the function's startup latency. As an alternative, the monitor tracks system calls invoked by the enclaved function to infer the function's state. Gramine-SGX, the platform selected for the enclaving of serverless functions, manages system calls instigated by a function internally via a library OS wrapper. Additionally, it provides methods for addressing system calls by delegating them to another application thread through an RPC-based service, all without necessitating an enclave existence. This configuration allows the monitor to accumulate time series data relating to the system call invocation of the function from the RPC service. During the collecting process, the monitor uses a lock-free ring buffer to efficiently interact with the SGX loader and reduce the data transfer overhead. Once it determines that the function has fully initialized, it signals the manager to initiate subsequent Cryonics workflows.

Probe executor. The probe executor, following the FLUSH stage, generates a test workload for a function. This enables

Cryonics to identify pages that are active and frequently reused during the function's operation. To this end, the probe executor mutates parameters within the user-provided sample requests for a function, thereby generating a set of test workloads. It then feeds these generated workloads to the function and repeats the entire trial process a predefined number of times. This iterative function trial process allows Cryonics to record the function's activated pages and create a working set (i.e., the RECORD stage). Note that this function trial task is conducted only once during the offline phase (before the function serving) and does not affect the performance of the online phase (function serving).

Device driver. The device driver runs on the kernel space and is responsible for the RECORD and the RELOAD stages. It interacts with the Manager via IOCTL interfaces. During the RECORD stage, it collects activated pages for the enclaved function by tracking page in/out transitions between the EPC and the regular memory region. Pages required for the function's execution are repeatedly loaded into and out of the EPC area. Given that page transfers between the EPC and regular memory rely on memory copying, the device driver is required to trace all page transitions, preserving their addresses. To efficiently trace these transitions and collects activated pages, Cryonics employs XArray [49], a data structure widely utilized for page caching. Note that the substantial task of recording pages during the offline phase, thereby preventing performance degradation during serving. When it comes to the RELOAD step, the device driver uses the proactive working set reloading (see Section 4.6), instead of relying on the inefficient page fault handler, significantly reducing page loading time.

System integration. Cryonics provides a design that incorporates methods for seamless integration with existing serverless computing platforms. The platforms deploy a container-based image containing a serverless function during the offline phase. Subsequently, this image is utilized to serve function invocations. When the system receives a function invocation, it queues the request and creates a server based on the previously deployed image. Once server creation is complete, the queued request is delivered to the server for processing. Cryonics is also fully compatible with the existing serverless platform, adhering to the serverless function life cycle.

Cryonics initializes SGX enclave-based serverless functions using pre-deployed images from the existing serverless platform during the offline phase. The monitor oversees the progress of function creation, and once it's complete, the manager collects page access information from the container process using the device driver in the (c) RECORD stage. The manager furnishes a metric report to the scheduler within the serverless platform. This report aids in the seamless delivery of function invocations to the SGX enclave serverless function. Following this, upon receiving a request from the ingress gateway, the scheduler provisions the serverless function. An SGX enclave-based container is then established using the previously provided node information and a predeployed image. This process culminates in the successful creation of the function. Subsequently, a notification is sent to the queuing component containing the function invocation. This notification streamlines the routing to the server where the function invocation originated.

4.5 Making a Working Set

The size of a function snapshot significantly influences the time required for reloading it into the EPC area upon function invocation. Consequently, minimizing snapshot size is crucial for achieving fast function reloading. To accomplish this, it is essential to keep the pages within the snapshot (i.e., a working set) both lightweight and future-proof by identifying and eliminating unnecessary pages. In order to facilitate this, Cryonics employs the following two working set optimization strategies, which are designed based on the characteristics of serverless functions.

(i) Referenced page tracking. A typical serverless function instance consists of two components: a reverse proxy (i.e., a sidecar [24]) and a function handler. The reverse proxy deserializes a request into a predefined format (e.g., protobuf [19]) and conveys it to the function handler. The handler then executes business logic based on the content within the request. These functions are triggered according to predefined types of requests, and their business logic is typically streamlined and stateless [10, 38]. The static nature of a function can influence its page locality. Prior research [47] shows that the page locality of a common serverless function exhibits about 97% page-by-page similarity across the same types of invocation. This result indicates that a function's memory layout comprises a larger immutable region as compared to a mutable region. Consequently, pages that have been referenced once are likely to be deterministically re-referenced in subsequent function invocations. Capitalizing on this characteristic, Cryonics traces reference counts of each page during the RECORD stage. These counts are then employed to construct a future-proof working set.

(ii) Obsolete pages elimination. Given that, a function instance includes language runtimes and third-party libraries, its memory footprint often exceeds the PRM size available on commodity SGX hardware. This not only increases the snapshot size but also results in frequent EPC page evictions, which, in turn, contribute to a degradation in function startup latency. However, most pages loaded into the EPC during the initialization phase of a function are unlikely to be used during the function's operational phase. Indeed, according to a previous study [47], between 61% and 96% of total function

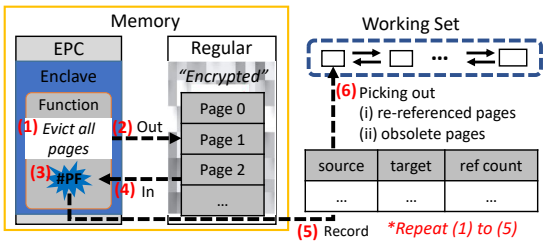

Figure 3: Working set recording procedure of Cryonics.

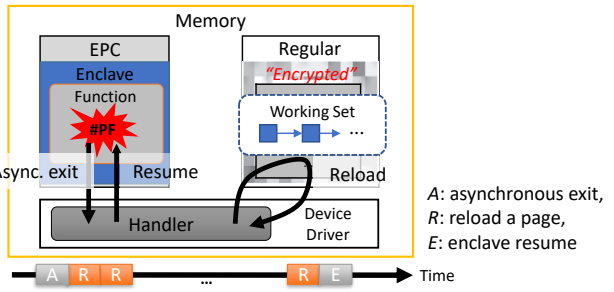

Figure 4: Proactive working set reloading. This loads multiple pages preemptively into the EPC area, avoiding the need for repeated enclave entrances and exits.

pages are utilized only during the function's initialization phase and are rarely referenced during the operational phase, in which the function processes requests. This finding suggests that these obsoleted pages can be removed from the working set, reducing the number of EPC page evictions and minimizing the snapshot size. Cryonics achieves this by identifying obsoleted pages based on reference counts and timestamps during the RECORD stage.

Based on the above two strategies, Cryonics initially categorizes the memory region of an enclaved function into immutable and mutable regions. It then identifies obsoleted pages as well as frequently referenced pages within the mutable region to create an optimized working set. This process is depicted in Figure 3. During the RECORD step, a page fault arises whenever the function attempts to access a page that isn't currently present within the EPC. Cryonics intercepts the page fault handler, extracts the address from the Control Register 2 (CR2) along with the destination page information to be reloaded into the EPC. This information is used to create a key-value pair which is then stored within the XArray located in the EPC area (key being the address). When doing so, Cryonics first checks whether the extracted address already exists within the XArray. If it does not, it denotes that the page has not been reloaded before. For such pages, Cryonics inserts a new key-value pair along with a timestamp and sets the reference counter of the page to one. If the extracted address is present within the XArray, it means a re-reference of that page. In this case, Cryonics increments the reference counter and updates the timestamp. This process of recording page faults continues until all test workloads generated by the probe executor have been processed by the function instance.

The stored information is utilized to create an optimized working set. Pages that are loaded only once at an early time are not likely to be re-referenced during function execution. These obsoleted pages are excluded from the working set. Conversely, pages that are frequently referenced for a long period of time are likely to be referenced in subsequent invocations. Cryonics assembles a working set based on pages that exceed a predefined reference threshold (default: 2). These optimization techniques equip our system with the ability to build lightweight, future-proof working sets for enclosed functions.

4.6 Proactive Working Set Reloading

During the process of reloading a function snapshot, it is essential that multiple pages required to process the request are concurrently loaded into the EPC area. The SGX typically depends on the page fault handler to reload these pages, a process involving an Asynchronous Exit (AEX). The page located in regular memory is copied to the EPC and then decrypted using the ELDU instruction. Following this, the SGX attempts to re-enter the enclave via the ERESUME instruction. The completion of the AEX, ELDU, and ERESUME steps requires approximately 10K, 44K, and 10K clock cycles, respectively [48]. These operations, while costly and time-intensive, are repeated on a per-page basis, thereby contributing significantly to the increase in the function's startup latency.

To address this problem, Cryonics employs a proactive working set reloading, which preloads all pages within the working set to avoid the repetitive and costly process of re-entering the enclave. Upon receipt of a function invocation, the enclaved function instance is initiated. A page fault then arises because the activated page of the enclave remains empty, prompting the AEX occurrence. Subsequently, the *async_exit_pointer* is triggered to perform a fall-through, and to resume once the target page has been reloaded. In this phase, Cryonics uses a customized handler for the ERESUME instruction as shown in Figure 4. This handler traverses all pages within the working set for the invoked function, attempting to preload all of these pages prior to re-entering the enclave. Following this, the operation of the function resumes, permitting the enclave to load new pages. The serverless platform then completes the initialization of the function instance as it does. After the processing of the request, Cryonics restricts new page reloading and flushes all pages within the EPC area. During the serving process, if the enclave requires a page not included within the working

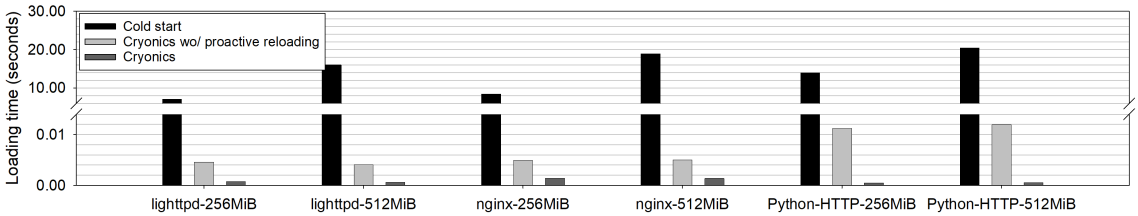

Figure 5: Function startup time of Cryonics and a cold-start-based approach for different function types.

set, the typical page reloading scheme is implemented as follows: AEX → page reload → resume. As a result, Cryonics effectively eliminates the repetitive process of exiting and reentering the enclave during the RELOAD step, substantially reducing the overhead and enhancing page loading time.

5 EVALUATION

This section outlines our evaluation environment and assesses the performance of Cryonics with regard to the startup time of SGX-enclaved serverless functions, in comparison to the traditional approach. Furthermore, we investigate the page reuse ratio of SGX-enclaved serverless functions on various real-world workloads, thereby indicating page locality. Finally, we identify obsolete pages and substantiate the feasibility of the proposed working set optimization technique by examining the proportion of a function instance's memory pages employed during execution.

5.1 Evaluation Environment

Our evaluation of Cryonics was carried out on a commodity server equipped with an Intel Xeon E-2286M CPU (Coffeelake microarchitecture), operating at 2.4GHz, featuring a 16MiB L3 shared cache, and 64 GiB of RAM. The server machine was running the Linux kernel 5.11. To measure the startup time of the SGX-enclaved function instance, we implemented a Cryonics prototype on top of Gramine-SGX (formerly known as Graphene-SGX) and the Linux SGX driver, facilitating interoperability with the PRELOAD, FLUSH, RECORD, and RELOAD stages. We utilized FunctionBench to measure the function loading time and page reuse utility under various workloads [21].

5.2 Overall Function Startup Time

In this experiment, we initially measure the time required to establish various applications commonly employed in serverless functions to handle requests within an SGX-enclaved function. For this purpose, we ported the following applications to SGX: nginx, lighttpd, and a Python-based web server. A client submits arbitrary requests to these applications to trigger the enclaved function. Here, we assess the loading time of the function while manipulating the memory footprint in three distinct function loading methodologies: conventional cold start, Cryonics without proactive reloading, and our Cryonics incorporating coalesced reloading. According to [38], the memory footprints of most serverless functions span between 256MiB and 512MiB. Consequently, we conducted experiments with two memory footprint sizes: 256MiB and 512MiB. Additionally, we included an extra scenario for memory-intensive conditions, such as training and inference in deep learning models, necessitating a minimum memory of 1024MiB.

We compare the function startup times of three distinct methods throughout the entire experiments in this section. The first methodology, cold start, necessitates both the creation of the enclave and the initialization of the server application. The second method is Cryonics without proactive working set reloading. It loads a snapshot taken by the Cryonics system rather than re-initializing the enclave and function instance. However, it relies on the traditional page fault handler for snapshot reloading. The final method is Cryonics, which proactively reloads the created snapshot.

Figure 5 presents the results of the experiment. The startup time under the cold start approach demonstrates a correlation with the memory footprint occupied by the enclave. For example, the loading time for lighttpd-256MiB is roughly half that of lighttpd-512MiB. As elaborated in Section 2, the bulk of time during a cold start is expended on stages such as page copying, encryption/decryption and measurement operations, and modifications to page permissions. These processes are time-intensive, and their durations linearly increase with the number of pages participating in these stages. Contrarily, Cryonics without proactive reloading solely entails copying the page between the EPC and the untrusted memory region, based on the demands managed by the page fault handler. As merely a subset of the initialized pages is accessed, this procedure is quicker than reloading the entire enclave. It signifies a significant enhancement in loading time, demonstrating an increase in speed by more than an order of magnitude compared to the cold start method under identical workloads. However, the loading time for Cryonics without proactive reloading can be influenced by various factors, such as workload attributes, the number of accessed pages, and a page eviction ratio. In comparison, Cryonics shows greater consistency in loading time, owing to its proactive reloading of the page working set, which accounts for

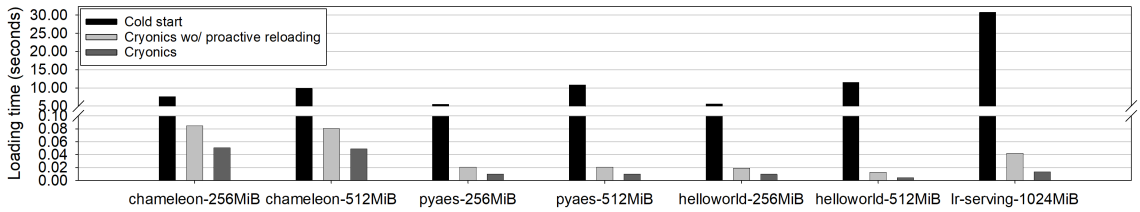

Figure 6: Function startup time of Cryonics and the cold-start-based approach on various Python runtime-based serverless functions with randomized workloads.

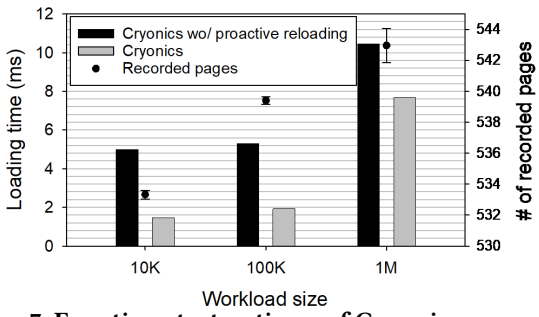

Figure 7: Function startup times of Cryonics on various workload sizes from 1KiB to 1MiB for Nginx.

page locality ahead of time, whenever a function spawning request is received from a serverless platform. This strategy alleviates the performance dependency associated with the sequence and patterns of reloaded pages.

5.3 Startup Time on Various Environments

In this section, we evaluate Cryonics in various workload settings. Figure 7 presents the function startup time required to process random workload (an HTTP request packet) of varying sizes (10KiB, 100KiB, and 1MiB) and the number of pages recorded during execution for Nginx. As the workload size increases, both Cryonics without proactive reloading and Cryonics exhibit a linear increase in load time. However, the increase in the number of activated pages is notably more gradual. Cryonics without proactive reloading correlates directly with workload size, while the proactive reloading method fits proportionally with the number of recorded pages. These results confirm that as the recycling of pages reduces the number of page evictions, the memory locality characteristic enhances, and subsequently, loading time improves due to decreased overhead in encryption/decryption stages during the page loading process. Moreover, the variations in the number of recorded pages also explain the non-uniform changes in loading time, as the volatility escalates with increasing workload size.

Subsequently, we measure the startup times of an enclaved function with different types of applications. We used four application types proposed by *FunctionBench* operating on a Python runtime [21, 47]. Firstly, *chameleon*, generates and responds to arbitrary HTML/XML templates with internal processing. The second application, *pyaes*, conducts a series of Python-based AES encryption and decryption operations, returning the results upon request. The third application, *Helloworld*, is a simple web service that generates arbitrary responses contingent upon client requests. All message exchanges are handled based on Google's gRPC protocol [19]. We assigned the memory footprint sizes of 256 and 512 MiB to these three workload types, reflecting the most prevalent memory sizes in serverless functions [38]. The final application, *lr – serving*, performs logistic regression, extensively utilized in machine learning, designed to operate in a serverless fashion within an SGX-based container. Given that *lr – serving* requires significantly more computational resources compared to other applications, we conducted this experiment in a 1024MiB environment. These services address all requests from a client, returning a dynamically altered result each time, thereby processing variables and data analogous to real services.

Similar to the previous experiment, we measured the function startup time for the cold start, Cryonics without proactive reloading, and Cryonics on the above four application types while altering their memory footprint. Figure 6 shows the experimental results. The cold start, characterized by a linear increase proportional to the memory footprint, primarily consists of enclave creation and initialization processes. Conversely, both Cryonics without proactive reloading and Cryonics exhibit much faster startup time capabilities than the cold start-based approach. In the prior environment, page evictions, contributing to performance degradation, rarely occurred during function serving since it did not encompass a language runtime heavily burdening the EPC area. Contrary, this experiment was executed on the Python runtime with *FunctionBench* applications [21], leading to frequent page eviction occurrences. This situation may affect the performance improvement of our system because the language runtime increases the memory footprint and demands a considerable number of new pages throughout the function execution. Indeed, when compared to the results of the experiment involving functions running in native environments, the performance improvement offered by our

system appears reduced. Nevertheless, both settings of Cryonics continue to deliver a function startup time that falls below 50ms.

5.4 Working Set Optimization

In this section, we assess the efficiency of the working set optimization techniques of Cryonics. Specifically, we first evaluate the system's capacity to effectively reduce the size of the working set. Then, we measure the extent to which the created working set encompasses the pages utilized during the execution of the function.

Working set size reduction. Frequent page evictions within the EPC area have the potential to prolong the startup times of functions. To circumvent this, Cryonics strategically minimizes the function snapshot size by purging obsolete pages from the working set. This evaluation aims to illuminate the extent of page obsolescence within the created working set to substantiate Cryonics' ability to effectively minimize the working set size through the elimination of such pages.

We initially created working sets for each function without excluding any pages. After that, we measured the percentage of pages referenced from this working set during the function's request processing. The experimental results are illustrated in Figure 8. The proportion of pages referenced during the function's execution ranged from a minimal 1% to 13%. Remarkably, the volume of referenced pages was diminutive, distributed within a range of 2%, in the case of Nginx and lighttpd serverless functions, which are operational in a native environment. Even for functions within the Python runtime, which require extensive page allocations for the runtime, the page reference ratio stood between 4-13% of the total working set. These findings suggest that a mere subset of pages within the working set is actively employed during actual function execution. Consequently, our system is equipped to efficiently discard obsolete, unreferenced pages, thereby inducing a significant reduction in the function snapshot size.

Consistency of working set size. Next, we delve into the fluctuations of the created working set size. Large swings in working set sizes can adversely affect page loading times when reloading these sets via our proactive reloading mechanism. Therefore, the consistency of the pages recorded within the working set emerges as a crucial metric for startup time. Figure 9 shows the number of recorded pages (i.e., the size of the working set) during the RECORD with box plots. Upon examination of various application types, we found the box plots to be closely aligned with a minimal presence of significant outliers. These results point towards a uniform startup time, which subsequently fosters a more consistent response time. This consistency gives Cryonics a performance edge over other methods such as the traditional cold startup.

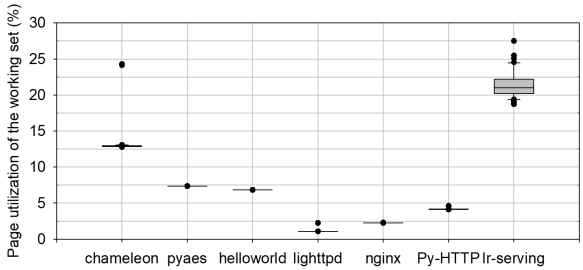

Figure 8: The page utilization ratio of the working set (prior to obsolete page reduction). The result tends to be low due to the inherent inclusion of various runtimes and third-party libraries, leading to functions using only a fraction of the entire set during execution.

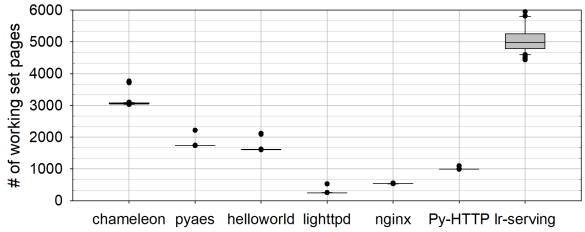

Figure 9: Variation of working set sizes. Cryonics ensures the consistency of the working set size, thereby improving the efficiency during the page reload.

Additional page loading ratio. In situations where an enclaved function accesses pages absent from its working set, page faults occur, prompting the inefficient page fault handler to load the requested pages into the EPC area. As discussed previously, this process necessitates repeated entrances and exits from the enclave, and thus frequent page faults within a working set can hinder the performance of the enclaved function. In this experiment, we assess how comprehensively the generated working set encompasses the pages required for the function's operation. More specifically, we measure the number of loaded pages not included in the working set, during a request processing across various application types. The experimental results are depicted in Figure 10.

Across all three function types evaluated, the extra page loading fell under 100 pages in the 80-90th percentile. Given the size of the working sets for these functions, as depicted in Figure 9, the number of extra pages equates to a mere 5% of the total working set. One noteworthy observation is that functions with a larger memory footprint necessitate a higher volume of extra page loading. This phenomenon is attributable to the Python runtime within the function possessing sparsely located pages, thereby diminishing locality during the creation of a working set. Despite this, Cryonics

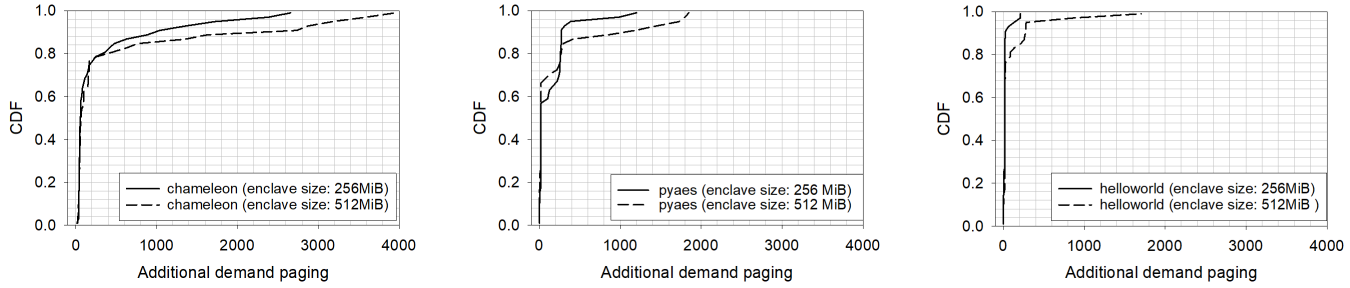

Figure 10: Cumulative Distribution Function (CDF) graphs depicting the number of loaded pages without the working set during the execution of functions: (a) chameleon, (b) pyaes, and (c) helloworld.

effectively generates future-proof working sets that encapsulate the majority of pages required for function operation across all given functions.

Accuracy of preloading mechanism. The accuracy of our proposed preloading method significantly impacts overall performance during execution. When accuracy is low, the frequency of additional page fetches on demand increases, leading to not only page reloads but also page evictions, incurring significant overhead when EPC space is insufficient. To assess the accuracy of this method, we must determine the utilization rate of the working set and the number of pages fetched on demand when serving a serverless function based on the reconstructed working set. Our experiments (see Figure 10) show that, for all three types of functions tested, the majority of cases (80-90th percentile) required fewer than 200 out-of-working-set pages during execution. As the memory footprint of the serverless function increases, the number of additional demand paging fetches also increases. The memory size of the serverless function for the real-world scenario, as used in [38], is smaller than our experimental setup of 512 MiB, which suggests the potential for better performance. Furthermore, less than 5% of pages within the working set underwent alterations throughout the execution. In summary, Cryonics operates based on a working set with consistent characteristics and a highly accurate pre-loading mechanism, indicating that the working set is future-proof.

5.5 Scalability

The objective of Cryonics is to restructure the working set, which comprises the SGX enclave-based function, into pages that are identified as reusable, as opposed to the current approach. This process eliminates pages of existing functions (which will not be used again) from being loaded into the space-limited EPC, enabling more functions to coexist on a single node while minimizing performance degradation. Even if it withstands the bursts of EPC page evictions, co-locating a large number of functions is not ideal as it significantly slows down the execution time of the function. From this perspective, we believe that the proposed method offers a more scalable design.

To assess scalability quantitatively, we conducted experiments with various workloads to determine the number of pages in the working set for each workload and the additional pages fetched. This enabled us to verify the number of functions that can be co-located. Based on this information, Cryonics determines the number of function instances that can coexist when functions possess the same workload capabilities within a limited EPC size (i.e., SGX1).

As depicted in Figure 9, the working set size utilized in serverless functions varies depending on the specific workload, which shows the consistent working set size. Figure 10 confirms that the number of additionally fetched pages consistently falls within the 80-90th percentile, totaling approximately 100 to 200 pages. When analyzing each workload individually, we observed that cases like Chameleon, pyaes, and Helloworld can accommodate between 20 to 40 co-located functions. In contrast, for lighter workloads such as lighttpd, Nginx, and Python-HTTP, it is possible to accommodate 50 to 100 functions simultaneously.

For reference, in SGX2, our approach may exhibit limitations in scalability due to relaxed EPC size constraints. However, viewed from another perspective, particularly in a cold-start scenario, Cryonics has demonstrated superior performance compared to existing approaches.

5.6 End-to-end Cost-benefit

The end-to-end process of our method consists of two parts: creating a working set in the offline phase and serving the serverless function online. Our Cryonics initiates the creation of a working set during the offline phase using predeployed code within the serverless computing environment. While creating a working set (i.e., (a) PRELOAD), the system measures every page within the SGX enclave instance. The

results are crucial for conducting local or remote attestation of the SGX enclave. However, since it is calculated sequentially in units of 256 bytes, it can be of concern in the case of rapidly spawning functions. This process accounts for most of the approximately 100K cycles it takes to load a page. When a lot of pages are loaded during the preload process, it takes time to flush them in proportion to the number of pages. This is one of the reasons why Table 1 shows that it takes a long time to load.

Fortunately, as we handle this during the pre-processing offline phase, it does not lead to performance degradation during serverless function invocation (i.e., online phase). The proposed method incurs additional overhead as the Cryonics snapshot utilizes cloud infrastructure resources, and extra tasks must be carried out beforehand during the offline phase, specifically in the pre-deployment process. Therefore, preparation for this is necessary. Nevertheless, we have designed Cryonics with a focus on designing methods for models to ensure a safer and more efficient serving of serverless functions during the online phase.

6 SECURITY ANALYSIS

This section analyzes the security implications of Cryonics throughout the entire procedure, from snapshot creation to restoration.

In the PRELOAD stage, a serverless function-based enclave is created, using the same procedure as that for creating a user-level SGX enclave. Within this stage, Cryonics relies on the assumption of trust in all code residing within the enclave, including the language runtime, third-party libraries, and business logic, thereby ensuring the security of the serverless function. Furthermore, the untrusted system software and co-located neighbor enclave are incapable of compromising the enclave-based function. This assurance is grounded in the fact that these entities are unable to access the memory contents and execution metadata, thanks to the protection afforded by the SGX enclave.

Next, in the FLUSH stage, it forcefully flushes the pages located in the EPC to the regular memory region. During this process, all flushed pages are copied to the regular memory region, rather than the PRM, while being encrypted. Once stored in the EPC, they are continuously monitored and tracked through the Enclave Page Cache Map (EPCM). Consequently, these steps are tightly bound by the SGX lifecycle, ensuring that no new attack vectors are introduced.

During the RECORD stage, a malicious page has the potential to be included in the working set by forcefully listing it. Additionally, before serving, a page that is part of the working set can be replaced with a malicious one in the RELOAD stage. This scenario is akin to an attacker remapping an arbitrary physical page to a virtual page, known as an active page remapping attack, which results in hijacking the program's control flow [15]. Fortunately, SGX diligently records each sensitive page involved in the working set to the EPCM within an array. The EPCM contains the virtual address assigned to the corresponding EPC page, enabling tracking of the page's owner and providing protection against page replacement or remapping attacks. The attacker's capability is limited to causing DoS attacks by manipulating the page since the correct pages are securely recorded in the EPCM.

Lastly, Cryonics shares an enclave-based snapshot, that is generated during the offline phase, for each function invocation. As the pages added during the online phase are not included in the working set, sharing the snapshots does not give rise to data leakage issues or introduce side channels. Furthermore, it is important to note that our security scope does not cover completely unresolved concerns, such as Return-oriented Programming (ROP) attacks and data-only attacks, which could potentially bypass Control Flow Integrity (CFI) protection.

7 RELATED WORK

Function startup optimization in serverless platforms. Several approaches have been developed to reduce the function startup latency. For instance, SOCK [32] employs an intricate three-tier caching strategy to expedite the initialization process of serverless functions. SAND [1], on the other hand, advocates for fine-grained sandboxing and a hierarchical message bus within serverless platforms to diminish the function startup time. To mitigate the startup latency issue, some approaches employ the snapshot method. Mohan et al.[31] present a method that pre-creates networking resources and dynamically reassociates them with container-based serverless functions to facilitate quick startup time. Similarly, SEUSS[9] suggests the use of a unikernel-based function snapshot to reduce startup latency, as this bypasses the time-consuming initialization stage. Additionally, REAP [47] introduces a snapshot optimization method that records the stable working set of memory pages for functions and proactively prefetches the result to spawn the function. These snapshot-based approaches inspired the foundational design principle of Cryonics. We have further developed this snapshot-based method to facilitate the operation of a serverless function within a trusted computing environment.

Enclave-based serverless function. Some pioneering studies have attempted to incorporate Trusted Execution Environment (TEE) technologies into serverless platforms. Table 3 summarizes the comparison of existing enclave-based serverless functions based on the criteria previously defined (see 4.2). This table assesses the fulfillment of requirements for accommodating different execution environments and the acceptable startup latency in real-world scenarios.

Table 3: A comparison of the previous studies on enclave-based serverless functions.

	TFaaS [6]	S-FaaS [2]	Clemmys [44]	PIE [25]	**Cryonics**
Efficient EPC usage (**R1** in 4.2)	–	–	✔	–	✔
Hardware-level compatibility (**R2**)	✔	✔	–	–	✔
Application-level compatibility (**R3**)	–	–	✔	✔	✔
Startup latency minimization	–	–	✔	✔	✔

TFaaS [6] constructs an SGX-enclaved JavaScript engine that securely executes JavaScript-based serverless functions. Similarly, S-FaaS [2] presents a method for executing general serverless functions within a system-wide SGX enclave and provides fine-grained accounting for the resource utilization of these functions. However, these studies mainly focus on securely deploying and managing serverless functions, overlooking the issue of startup latency of the SGX enclave. Also, they assume the system-wide SGX enclave remains continually active to bypass the initialization overhead.

Clemmys employs enclave dynamic memory management that only supports SGX2, with batched EPC augmentation to enhance the startup latency of enclaved functions [44]. However, it entails repeatedly adding redundant pages to the EPC to recreate the same enclave, consequently consuming hundreds of milliseconds to spawn the enclaved function. PIE adopts a memory mapping model for the enclaved function to reduce startup latency [25]. Nevertheless, this approach necessitates modifications to the underlying SGX hardware, an overhaul of the SGX lifecycle, and continuous occupation of the EPC with a preordained region for memory mapping. Moreover, both Clemmys and PIE are designed exclusively for SGX2, lacking a protection scheme against the integrity and replay attacks [20]. In contrast to these previous studies, our system necessitates no alterations to the SGX hardware and is designed to accommodate both SGX1 and SGX2. Additionally, it ensures rapid startup times for enclaved functions through the creation of optimized snapshots.

Run applications on SGX enclave. There are attempts to accommodate various applications in several specific domains to the SGX enclave. These contain a framework for MapReduce tasks [37], serverless computing platform [25], JavaScript language runtime [17], Rust-based application [13], Bitcoin applications [45], in-memory databases [34], object stores [35], key-value store [22] and middlebox [43]. Furthermore, it has been developing to port legacy applications to enclave - the library OS [5, 33, 46], the library wrapper [39], and the instruction wrapper models [3, 42]. In this paper, we adopt the Gramine-SGX Shielded Container, a deployable Docker container protected by SGX enclave through Graphene-SGX, to run the serverless function on a container.

8 DISCUSSION

Cryonics offers a strong security foundation for a serverless function execution environment, although there exists for further improvement.

Scaling-out. To ensure high availability, Cryonics utilizes the *Enclave Fork()* provided in [41] and [46] to facilitate the scaling-out of created snapshots to other nodes or within the same node. Since SGX employs distinct encryption keys for each enclave, it is unable to decrypt snapshots on another node or enclave once they are copied. Therefore, it is necessary to first establish the procedure of creating a new enclave and securely copying the memory contents of the serverless function instance, which serves as the replication target. In other words, a new enclave is created, followed by the establishment of a secure channel through key exchange between the enclaves, and finally the transfer of memory contents from one enclave to another. Although this deployment process is time-consuming, it does not impact the operation and startup performance as it is performed offline.

Snapshot versioning. Cryonics retrieves the pre-deployed code and generates a snapshot. Due to the time difference between program deployment and snapshot creation, there is a possibility of version control inconsistencies being exposed. To address this issue, it is essential to manage hash values for the separately deployed code and the code generated within the snapshot, and periodically synchronize them.

9 CONCLUSION

This paper introduced Cryonics, a snapshot-based spawning system designed for SGX-enclaved serverless functions. We identified the inherent limitations of the current SGX design in executing a serverless function instance. We addressed the limitations by creating an optimized working set of the instance and proactively reloading it. Cryonics generates a future-proof working set of the snapshot by examining the function, considering obsolete pages and locality, with the aim of reducing startup latency. Our evaluation demonstrated that Cryonics outperforms existing methods in terms of server startup time by a factor of 10 to 100 under an SGX enclave-based serverless function. The results also exhibited the stability of startup time with proactive reloading, thereby ensuring consistent service provision. While there are several avenues for further enhancement of Cryonics, such as support for node-level scalability and the integration of a file-backed snapshot, we believe that Cryonics establishes a robust foundation for the forthcoming generation of secure serverless computing.

REFERENCES

[1] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. 2018. SAND: Towards High-Performance Serverless Computing. In *Proceedings of USENIX Annual Technical Conference (ATC '18)*.

[2] Fritz Alder, N. Asokan, Arseny Kurnikov, Andrew Paverd, and Michael Steiner. 2019. S-FaaS: Trustworthy and Accountable Function-as-a-Service using Intel SGX. In *Proceedings of ACM SIGSAC Conference on Cloud Computing Security Workshop (CCSW '19)*.

[3] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, , Mark L Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

[4] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. 2017. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*.

[5] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2014. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*.

[6] Stefan Brenner and Rüdiger Kapitza. 2019. Trust more, serverless. In *Proceedings of ACM International Systems and Storage Conference (SYSTOR '19)*.

[7] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, , and R. Strackx. 2019. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of Order Execution. In *Proceedings of USENIX Annual Technical Conference (ATC '19)*.

[8] J. V. Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, , and F. Piessens. 2020. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *Proceedings of 41st IEEE Symposium on Security and Privacy (S&P '20)'*.

[9] James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, and Jonathan Appavoo. 2020. SEUSS: skip redundant paths to make serverless fast. In *Proceedings of European Conference on Computer Systems (EuroSys '20)*.

[10] Paul Castro, Vatche Isahagian, Vinod Muthusamy, and Aleksander Slominski. 2022. Hybrid Serverless Computing: Opportunities and Challenges. arXiv:2208.04213 [cs.DC]

[11] Sam Corcos. 2022. *How to Keep Your AWS Lambda Functions Warm.* Retrieved May 31, 2023 from Available:https://acloudguru.com/blog/engineering/how-to-keep-your-lambda-functions-warm

[12] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained.

[13] Yu Ding, Ran Duan, Long Li, Yueqiang Cheng, Yulong Zhang, Tanghui Chen, Tao Wei, and Huibo Wang. 2017. POSTER: Rust SGX SDK: Towards Memory Safety in Intel SGX Enclave. In *Proceedings of The ACM Conference on Computer and Communications Security (CCS '17)*.

[14] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *Proceedings of 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*.

[15] Nicolas A. Economou and Enrique E. Nissim. 2016. *Getting Physical: Extreme abuse of Intel based Paging Systems.* Retrieved June 4, 2023 from Available:https://www.coresecurity.com/sites/default/files/private-files/publications/2016/05/CSW2016%20-%20Getting%20Physical%20-%20Extended%20Version.pdf

[16] Xing Gao, Zhongshu Gu, Zhengfa Li, Hani Jamjoom, and Cong Wang. 2019. Houdini's Escape: Breaking the Resource Rein of Linux Control Groups. In *Proceedings of the 26th ACM Conference on Computer and Communications (CCS '19)*.

[17] David Goltzsche, Colin Wulf, Divya Muthukumaran, Konrad Rieck, Peter Pietzuch, and Rüdiger Kapitza. 2017. TrustJS: Trusted Client-side Execution of JavaScript. In *European Workshop on Systems Security (EuroSec '17)*.

[18] Google. 2022. *Google Cloud Functions.* Retrieved April 5, 2023 from https://cloud.google.com/functions

[19] K. Indrasiri and D. Kuruppu. 2020. *gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes.* O'Reilly Media.

[20] Simon Johnson, Raghunandan Makaram, Amy Santoni, and Vinnie Scarlet. 2022. *Supporting Intel SGX on Multi-Socket Platforms.* Retrieved April 5, 2023 from Available:https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/supporting-intel-sgx-on-mulit-socket-platforms.pdf

[21] Jeongchul Kim and Kyungyong Lee. 2019. FunctionBench: A Suite of Workloads for Serverless Cloud Function Service. In *Proceedings of IEEE International Conference on Cloud Computing (CLOUD '19)*.

[22] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. ShieldStore: Shielded In-memory Key-value Storage with SGX. In *Proceedings of the The European Conference on Computer Systems (EuroSys '19)*.

[23] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *Proceedings of 40th IEEE Symposium on Security and Privacy (S&P '19)*.

[24] Junfeng Li, Sameer G. Kulkarni, K. K. Ramakrishnan, , and Dan Li. 2019. Understanding Open Source Serverless Platforms: Design Considerations and Performance. In *Proceedings of the 5th International Workshop on Serverless Computing (WoSC '19)*.

[25] Mingyu Li, Yubin Xia, and Haibo Chen. 2021. Confidential serverless made efficient with plug-in enclaves. In *Proceedings of the International Symposium on Computer Architecture (ISCA '21)*.

[26] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*.

[27] Microsoft. 2022. *Azure Functions.* Retrieved April 5, 2023 from https://azure.microsoft.com/en-us/products/functions

[28] MITRE. 2014. *CVE-2014-9357.* Retrieved April 5, 2023 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9357

[29] MITRE. 2015. *CVE-2015-3456.* Retrieved April 5, 2023 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3456

[30] MITRE. 2015. *CVE-2015-5154.* Retrieved April 5, 2023 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5154

[31] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. 2019. Agile Cold Starts for Scalable Serverless. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '19)*.

[32] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2018. SOCK: Rapid Task Provisioning with Serverless-Optimized Containers. In *Proceedings of USENIX Annual Technical Conference (ATC '18)*.

[33] Christian Priebe, Divya Muthukumaran, Joshua Lind, Huanzhou Zhu, Shujie Cui, and Vasily A. Sartakovand Peter Pietzuch. 2019. SGX-LKL: Securing the Host OS Interface for Trusted Execution. In

arXiv:1908.11143.

[34] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In *Proceedings of 39th IEEE Symposium on Security and Privacy (S&P '18).*

[35] Anjo Vahldiek-Oberwagner Thomas Knauth Pramod Bhatotia Christof Fetzer Robert Krahn, Bohdan Trach. 2018. Pesos: Policy Enhanced Secure Object Store. In *ACM EuroSys.*

[36] Peter Sbarski and Sam Kroonenburg. 2017. *Serverless architectures on AWS: with examples using Aws Lambda.* Manning Publications, New York.

[37] Felix Schuster, Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud using SGX. In *Proceedings of 36th IEEE Symposium on Security and Privacy (S&P '15).*

[38] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloudd Provider. In *Proceedings of USENIX Annual Technical Conference (ATC '20).*

[39] Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. 2017. Panoply: Low-TCB Linux Applications With SGX Enclaves. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium (NDSS '17).*

[40] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. 2020. Prebaking Functions to Warm the Serverless Cold Start. In *Proceedings of the annual ACM/IFIP Middleware conference (Middleware '20).*

[41] ASYLO TEAM. 2019. *Real-World Applications in Enclaves.* Retrieved April 5, 2023 from https://asylo.dev/blog/2019/asylo-redis-sqlite.html

[42] Dave (Jing) Tian, Joseph Choi, Grant Hernandez, Patrick Traynor, and Kevin Butler. 2019. A practical intel sgx setting for linux containers in the cloud. In *Proceedings of ACM Conference on Data and Application Security and Privacy (CODASPY '19).*

[43] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. 2018. ShieldBox: Secure Middleboxes using Shielded Execution. In *Proceedings of Symposium on SDN Research (SOSR '18).*

[44] Bohdan Trach, Oleksii Oleksenko, Franz Gregor, Pramod Bhatotia, and Christof Fetzer. 2019. Clemmys: Towards Secure Remote Execution in FaaS. In *Proceedings of the ACM International Systems and Storage Conference (SYSTOR '19).*

[45] Muoi Tran, Loi Luu, Min Suk Kang, Iddo Bentov, and Prateek Saxena. 2018. Obscuro: A Bitcoin Mixer using Trusted Execution Environments. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC '18).*

[46] Chia-Che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *Proceedings of USENIX Annual Technical Conference (ATC '17).*

[47] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. 2021. Benchmarking, Analysis, and Optimization of Serverless Function Snapshots. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21).*

[48] Ofir Weisse, Valeria Bertacco, and Todd Austin. 2017. Regaining lost cycles with HotCalls: A fast interface for SGX secure enclaves. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17).*

[49] Matthew Wilcox. 2018. *XArray.* Retrieved April 5, 2023 from Available: https://docs.kernel.org/core-api/xarray.html

[50] Y. Xu, W. Cui, and M. Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *Proceedings of 36th IEEE Symposium on Security and Privacy (S&P '15).*