The Twilight of SGX, but Not its Tests: An In-Depth Study of Testing in SGX Projects

1st Jiapei Deng
College of Computer Science
Beijing University of Technology
Beijing, China
piepie@emails.bjut.edu.cn

2nd Yin Liu*
College of Computer Science
Beijing University of Technology
Beijing, China
yinliu@bjut.edu.cn

***Abstract*—The Software Guard Extensions (SGX) is a Trusted Execution Environment (TEE) solution that has transitioned from Intel's processor lineup for desktops and laptops to being exclusive to cloud servers. We believe one reason for this forced shift is that SGX-based applications lack comprehensive testing and may be vulnerable to security attacks. Although there have been numerous studies on SGX, research focused specifically on SGX's testing remains limited.**

In this paper, we take a first step in studying SGX-related test code, test scenarios, test oracles, and users' SGX cloud practices. Specifically, our study seeks to answer three research questions: what is the current status of the SGX test code? what are the test scenarios and oracles for SGX? how do users utilize or test SGX in the cloud? To that end, our study (1) analyzes the status quo on over 9,000 valid lines of SGX-related test code extracted from real-world projects, (2) develops 43 test scenarios identified from Intel's official documentation, along with a formalization that clearly expresses each scenario and its connected oracle, and (3) investigates into users' SGX practices in the cloud, based on 16 valid user feedback gathered from leading cloud vendors' websites and the StackExchange network.

From this work, we established three corresponding datasets: a test code set, a test scenario set, and a user feedback set. As a result, we identified three critical pieces of guidance, four testing principles, and five key findings related to SGX testing. These insights can guide and inspire software testers in creating valid test cases for SGX-based projects, while also offering suggestions for Intel, other TEE vendors, and cloud providers on how to improve their testing solutions.

***Index Terms*—Cloud Computing Security, Software Guard Extensions (SGX), Software Testing, Empirical Study.**

I. Introduction

"The moral is obvious. You can't trust code that you did not totally create yourself."
– Ken Thompson, Reflections on Trusting Trust, 1984

The Trusted Execution Environment (TEE) is an infrastructure designed to safeguard code and data against security attacks, both in cloud servers and client devices. Around 2015, Intel provided its own TEE solution, Software Guard Extensions (SGX) [19], then integrated the technique into its processor lineup [18]. During that period, SGX, as an emerging star, helped secure a vast number of desktops, laptops, and cloud servers. However, seven years later, Intel announced that SGX would focus on cloud server processors only, rather than those for desktops and laptops [32]. It is widely recognized that emergence, evolution, and obsolescence are common life cycles for technology, and the reasons behind these changes are often complex. We believe that a significant reason for the decline of SGX lies in its testing practices, especially the testing in SGX-based projects[1].

In fact, the SGX itself remains vulnerable to a series of attacks, including both system and software-targeted ones [9], [29], [36]. The former primarily focuses on address translation, cache, and DRAM [35], the latter typically exploits the SGX software interface (e.g., ECall/OCall) to compromise SGX-based projects, such as leaking secrets [37]. While application developers—those who use SGX to secure their projects—may not be able to prevent the system-targeted attacks, they can identify unsafe and improper usage of SGX that leads to the software-targeted attacks by conducting appropriate tests.

However, existing research on SGX testing is insufficient. Most of them only concentrate on measuring the execution performance of SGX projects [13], [38] or on fuzzing or monitoring the SGX's isolated execution environment (i.e., enclaves) [6], [21], [34]. These studies often overlook crucial aspects, such as test scenarios, test oracles, and relevant test code for SGX-based projects. Worse, as SGX has been shifted towards cloud environments, there is a notable lack of research addressing users' SGX practices in the cloud.

In this paper, we conduct an in-depth study of testing in SGX projects, aiming to draw a comprehensive picture of SGX-related test code, test scenarios, test oracles, and the users' SGX cloud practices. To that end, our study seeks to address the following research questions:

- *RQ-1: What is the current status of the SGX test code?*
- *RQ-2: What are the test scenarios and oracles for SGX?*
- *RQ-3: How do users utilize or test SGX in the cloud?*

To address RQ-1, we begin by using a triple-filtering approach to obtain a selection of valid SGX projects from GitHub, followed by a keyword-set based method and a human-AI pair analysis process to identify valid test code and extract its characteristics. To address RQ-2, we manually review Intel's SGX Developer Guide to extract a set of test

* Corresponding author.

[1]In this paper, we call projects using SGX components "SGX-based projects," using "SGX project" and "SGX-based project" interchangeably.

scenarios. We then analyze and formalize each scenario along with its corresponding oracle. To address RQ-3, we collect and filter valid user feedback automatically from renowned cloud vendors' websites and the StackExchange network, and then manually analyze it to gain insights into SGX usage and testing strategies within these cloud environments.

To the best of our knowledge, this is the first study to thoroughly investigate testing in SGX projects and users' SGX cloud practices. This paper makes the following contributions:

1) *A comprehensive examination of the current state of SGX-related test code in real-world projects, along with methods to identify valid test code.*
2) *An in-depth analysis of test scenarios and oracles for SGX, together with a formalization for each scenario and its corresponding oracle.*
3) *A thorough exploration of users' SGX cloud practices.*
4) *Three distinct datasets: a test code dataset, a test scenario dataset, and a cloud user feedback dataset.*
5) *Three essential pieces of guidance for practitioners, four testing principles, five key findings, and a series of insights and discussions related to SGX testing.*

We have made all of the datasets mentioned above available at: https://github.com/ppppppie/SGX-Test-Analysis

II. Background

1) TEE and SGX: A Trusted Execution Environment (TEE) [12] is a security technology designed to protect code and data from illegitimate access, damage, and tampering. The fundamental principle of TEE is "isolation," which involves placing the sensitive code and data in a secure execution environment that is difficult for attackers to breach. One prominent TEE solution is Intel's SGX. The SGX creates isolated environments called *enclaves* to provide the security features associated with TEE for a wide range of devices, including desktops, laptops, and cloud servers that use Intel processors. Recently, Intel has shifted its strategy, and SGX is now supported on cloud servers only.

2) SGX-based Projects and Their Inner Communication: In general, an SGX-based project consists of two components: the trusted part (i.e., the enclave) and the untrusted part (also known as the outside application). These two components communicate mainly through two types of function calls: ECall and OCall. ECall, stands for "Enclave Call," which refers to accessing an interface function inside the enclave. OCall, or "Out Call," refers to accessing external applications from inside the enclave [20]. Developers can define ECall and OCall functions using a special SGX file known as an Enclave Definition Language (EDL) file. Based on the EDL file, the SGX SDK helps generate the necessary code for facilitating communication in and out of the enclaves.

For example, if a financial application requires SGX to secure its payment functionality, the payment function and any related data need to be classified as trusted. These components should be placed within an enclave. To accomplish this, a developer can designate the payment function as an ECall function in the EDL file, implement it within the secure enclave, and call it from untrusted code (i.e., the outside application). Because the function and its associated data operate inside the enclave, SGX can ensure their security. Furthermore, the payment function within the enclave can call external functions and transmit data through OCalls.

3) SGX and Cloud: Data security and privacy concerns have long plagued cloud-based services. Before introducing SGX to their servers, leading cloud providers repeatedly assure users that their cloud services have been equipped with advanced security techniques. However, once the data was sent to the cloud server, users began to worry about the potential for their data to be leaked or abused. Indeed, cloud servers may have severe vulnerabilities that cyber attackers can exploit [1]–[4]. Moreover, sensitive user data may be accessed or exposed by careless employees, insiders, or government entities [31], [33]. Thanks to the SGX, cloud servers can have a secure space that both cloud providers and their customers trust. Although Intel has announced that its latest chips for desktops and laptops no longer support SGX, it is still available in the company's chip lineup for cloud servers. In fact, major cloud providers, such as Microsoft Azure, Alibaba Cloud, IBM Cloud, OVH Cloud, and more, have been leveraging SGX to offer enhanced security services to their customers [32].

III. Methodology

In this section, we outline our research goals and scope, followed by an overview of our solution.

A. Goal and Scope

The goal of this paper is to demystify testing in SGX projects by focusing on three subjects: (a) SGX-related test code, unveiling its current status (discussed in § IV), (b) SGX's test scenarios and oracles, including the rules and formalization (discussed in § V), and (c) users' SGX practices in the cloud, based on user feedback from dominant cloud platforms (discussed in § VI).

For (a), we investigate real-world SGX projects and their test code available on GitHub. We opt for open-source projects instead of commercial ones due to accessibility concerns, as the latter can only be accessed by their employees. Additionally, the open-sourced projects and datasets can be shared more easily with the public without encountering legal issues. Notice that, we exclude demos, sample code, tutorials, and research prototypes from our study because their test code is often immature or focuses solely on their basic functionality rather than SGX-related features. Furthermore, SGX SDKs, along with supporting libraries, testing tools, security detectors, and attack approaches targeted at SGX, are excluded as they focus on supporting or targeting SGX instead of utilizing it.

For (b), we create the scenarios based on Intel's SGX Developer Guide [20]. Although many relevant technical blogs and articles exist online, Intel's document is more comprehensive, valid, and trustworthy. For (c), we examine user feedback from notable cloud platforms, including Microsoft Azure, Alibaba Cloud, IBM Cloud, and OVH Cloud. We select these providers because they are recognized industry leaders and

have been mentioned in Intel's announcements as confirmed users of SGX for their cloud services [32]. We also collect user feedback from the StackExchange network, as it is one of the largest technical communities worldwide.

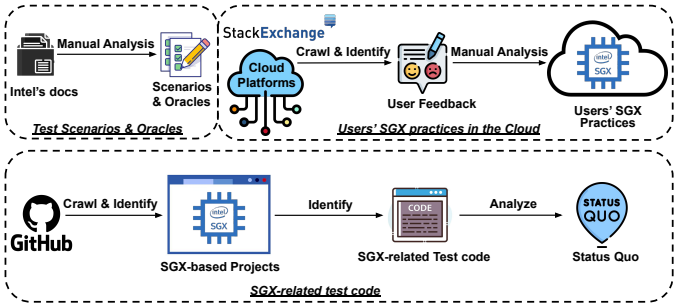

Fig. 1. The Workflow of Our Study.

B. Solution Overview

Figure 1 presents the workflow of our solution, which includes three parts: (a) our study on *SGX-related test code* (the bottom of the Figure 1) begins by collecting and filtering a number of valid SGX-based projects from GitHub. We then extract the SGX-related test code from each project, establishing a set of test code. Finally, we analyze this set to reveal the characteristics of the test code; (b) our study on *SGX's test scenarios and oracles* (the left side at the top of Figure 1) involves a manual analysis of Intel's SGX Developer Guide. This analysis extracts a comprehensive set of test scenarios and oracles specifically for SGX-based projects; (c) our study on *users' SGX practices in the clouds* (the right side of the top of Figure 1) starts by scraping and identifying a swath of user feedback and suggestions related to SGX from leading cloud platforms and the StackExchange network. Next, we manually review this information to extract important insights related to SGX usage and testing in cloud services.

IV. Test Code in the Wild

This section outlines the challenges and our approach to studying SGX-related test code, followed by our findings.

A. Challenges

Empirically studying test code is a complex task. Identifying valid test code alone is particularly challenging, and our research compounds this difficulty by thoroughly examining SGX-related testing. This effort to provide a complete picture of SGX test code presented us with three main challenges.

Challenge-1: How can we obtain valid SGX-based projects? To analyze SGX-related test code, the first step is to acquire the source code for SGX-based projects, as test code is typically tied to its corresponding source code. However, closed-source projects that use SGX are not accessible to us. On the other hand, although we can access open-source projects on GitHub, identifying valid SGX-based projects among the over 420 million repositories on the world's largest developer platform can be challenging.

Challenge-2: How can we identify valid test code? Given an SGX-based project, identifying test code, particularly the portions related to SGX, can be rather tough. A straightforward way is to check if files or function names contain the keyword "test" (e.g., "testSGXCloudService"). In fact, this approach has been applied in several papers published on premier research conferences [10], [22]. Despite its effectiveness, the method has its limitations, particularly in false negatives. That is, test code may not always contain the word "test." In other words, developers can use alternative naming conventions such as "assertion," "verify," "validate," or "benchmark," among others. Additionally, the test code related to SGX contains specific keywords that go beyond the standard ones.

Challenge-3: How can we cope with a code snippet or project that involves unfamiliar domains? When reviewing a code snippet and its associated project, we may encounter areas that are unfamiliar to us. It is difficult to determine if the code is meant to perform a test, and whether it specifically assesses any SGX-related functionality. Similarly, as we analyze the dataset of identified SGX-related test code, we may again come across unfamiliar code and projects while extracting characteristics and understanding the current status.

B. Methodology

1) A Triple-Filtering Approach for Obtaining Valid SGX Projects: To overcome Challenge-1, we take a three-step filtering process: our first step starts with filtering projects from GitHub. Using the GitHub API [11], we developed a crawler and utilized a commonly used keyword, "sgx," to filter a wide range of projects potentially related to SGX. In the second step, we filter out invalid projects by checking for the presence of specific files: the EDL files and the "sgx_urts.h" header file[2]. Additionally, we apply a date filter, considering only the projects created after 01/01/2016 as valid, since Intel rolled out SGX around that time.

In the third step, we filter projects based on their categories. Although we have a more reliable dataset of SGX-based projects after the previous steps, we found that many projects in this group are sample code, demos, research prototypes, or others that do not qualify as containing valid test code. Hence, we manually analyzed and categorized all the projects, excluding those that fall into categories such as sample code, demos, and research prototypes, and so on. As a result, we identified a number of valid SGX-based projects for further examination of the SGX-related test code.

2) Two Expanded Keyword Sets for Identifying Valid Test Code: To overcome Challenge-2, we create two keyword sets to identify valid test code. The first one is an expanded keyword set of testing, which encompasses 23 test code-related terms across seven different categories, including basic words, assertions, test frameworks, test doubles, verification, performance, and debug & coverage. Unlike prior studies, which only used the keyword "test", we use the whole set as a filter. By filtering for files that include words from the keyword set, we obtain a collection of source code files most

[2]Based on Intel's official documents, any project using SGX must include the EDL file, a file that is used for defining ECalls/OCalls, as well as the "sgx_urts.h" file, a part of the SGX SDK that facilitates necessary interactions between inside and outside enclaves [17].

likely to contain test code. The second keyword set consists of over 40 terms related to SGX and its enclaves. We use this set as a reference. In the subsequent analysis, we examine each test code file closely, paying particular attention to those that contain any of the terms from this keyword set (details are available in our open-sourced datasets).

3) A Human-AI Pair Analysis Process: To overcome Challenge-3, we perform a human-AI pair analysis, where AI interprets code and humans verify it. That is, given a code snippet that falls outside our knowledge, we leverage large language models (LLMs) to interpret or, if necessary, summarize the related project. We then manually check if the code snippet is a valid test code and relevant to SGX. In particular, we use simple prompts for AI queries, like "Explain the execution logic of this code." In this process, we did not limit ourselves to a specific model; the models we used included DeepSeek-R1, ChatGPT-4, and KIMI K1.5/K2. Please note that we use LLMs only when necessary and as an assistant, not for decision-making. In other words, regardless of the AI's interpretation, we will do our part to verify the following criteria: (a) the inclusion of testing elements in the source code, such as test discovery, performance measurement, and feature verification. If these elements are present, we will consider it a valid test code. (b) that the test code calls SGX interfaces or that its logic is relevant to SGX modules. If all the above criteria are met, we will consider the code a valid SGX-related test code.

C. Dataset

Following the approaches above, our dataset comprises 217 SGX projects, which include 54 valid projects and 163 invalid ones. As stated earlier, the invalid projects consist of SGX SDKs and supporting libraries, demos, sample code, tutorials, testing tools, security detectors for SGX, SGX-targeted attacks, and research prototypes. Additionally, we have excluded projects that lack README files and those that were archived during this study. Out of the 54 valid projects, 31 include testing code, while 9 of these further involve testing SGX-related components. In total, we have gathered 9,139 lines of testing code related to SGX.

D. Results and Findings

Finding 1: *Few projects test their SGX-related components.* As previously mentioned, we have excluded a series of projects that are generally recognized as unlikely to provide valid test code. Nevertheless, among the 54 filtered valid projects, only 9 (16.7%) contain SGX-related testing code. While we obtained more than 9,000 lines of valid testing code, these lines only represent 4.6% of the total testing code from the 9 projects.

Further, we calculated the ratio of SGX-related testing code to the total testing code within these projects. Six of the projects have SGX-related testing code that constitutes more than half of their total testing code. However, all six of these projects are relatively small, each with fewer than 10,000 lines of testing code. In contrast, the two larger projects, each with over 10,000 lines of testing code, have SGX-related testing code accounting for only 0.1% (220 out of 178,921) and 32.6% (3,412 out of 10,461) of their total testing code, respectively.

Finding 2: *Projects often test their SGX-related components in a manner that is too simplistic and focused too narrowly.* As shown on the left side of Figure 2, only 4 out of the 9 projects include unit tests to verify their SGX-related components. In contrast, more projects use smoke testing, which wraps up a series of testing functions as a whole to ensure that basic functionalities are working correctly. Furthermore, in our analysis of the test focus, we discovered that all the test code solely targets the project's functionality, without including any security or performance test cases (as shown on the right side of Figure 2). This indicates that these projects focus only on whether their SGX components work well rather than on security or performance concerns.

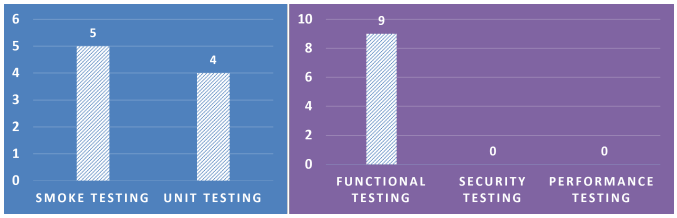

Fig. 2. Testing Strategies in SGX-related Testing Code.

Finding 3: *Projects test their ECall and OCall, frequently focusing on cryptography and the enclave's lifecycle.* In our analysis of the test subjects, we found that projects typically test either their ECalls, OCalls, or both. As illustrated on the left side of Figure 3, 8 out of the 9 projects evaluate ECalls, while 6 assess OCalls, with 5 of them testing both types.

Regarding functionalities, the right side of Figure 3 indicates that the most frequently tested features include cryptography components (e.g., key generation, encryption/decryption), and the enclave's lifecycle (e.g., an enclave's creation/destruction). Other commonly tested functionalities include printing and debugging information, sealing/unsealing, attestation, multiple-precision arithmetic, file operations, and socket functions.

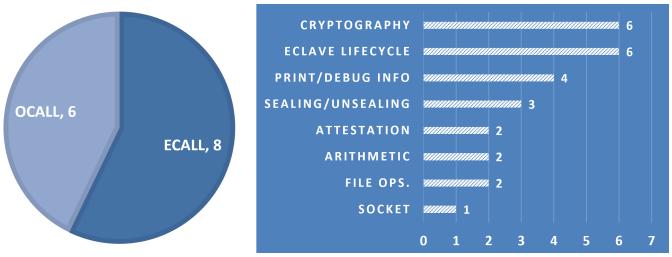

Fig. 3. Subjects and Functionalities targeted by SGX-related Testing Code.

E. Guidance for SGX practitioners

Given the findings above, we recommend that SGX practitioners take a step back to reconsider their testing processes.

a) Adding more testing code in SGX projects: Findings 1 and 2 illustrate the noticeable lack of testing for SGX components and the need for various testing strategies. With only simple smoke tests, or even no testing code at all, SGX issues—especially vulnerabilities in code within enclaves—are difficult to detect. This often leads to problems affecting the entire SGX system.

TABLE I
TEST SCENARIOS OF SGX (PARTIAL).

Test Domain	Test Goal	Test Subjects	Prerequisites	Condition	Importance
Security	To safeguard all secrets assigned to production enclaves	Enclave's debug flag	Code outside the enclave passes secret data into the enclave	Be configured as false	Must
Performance	To improve data clearing performance	Variables do not contain secrets	Some variables do not hold secrets that need clearing	Be cleared by the memset function	Recommendation
Execution Bugs	To ensure that the RDRAND instruction has been successfully performed	Code used the RDRAND instruction.	The RDRAND instruction has failed	Be attempted again up to 10 times	Recommendation

b) Focusing not only on functionality but also on security and performance: Finding 2 shows that the analyzed projects focus solely on testing functionalities. Numerous prior studies indicate that SGX may be compromised by crafted attacks or incur performance costs for the entire system. Therefore, necessary test cases on security and performance need to be developed and executed. Especially, improper usage of SGX should be tested, as we found that it may cause security and performance issues (as discussed in § V).

c) Paying more attention to SGX's cryptographic support and its lifecycle operations: Finding 3 indicates that the most frequently used or tested functionalities of SGX are its cryptographic support and lifecycle operations. Developers of SGX need to prioritize maintaining these functionalities. Furthermore, project developers should follow the correct usage of these functions (as discussed in § V) and stay informed about patches regarding these features, ensuring timely updates to the new versions of SGX.

V. SGX TEST SCENARIOS AND ORACLES

This section presents our method for creating and formalizing test scenarios and discusses our findings.

A. Methodology & Dataset

We review Intel's SGX Developer Guide [20] sentence by sentence, labeling key descriptions, requirements, and recommendations related to SGX testing. We then extract essential elements for defining test scenarios, such as goals, significance, test subjects, prerequisites, and pass conditions. Our approach identified a total of 43 test scenarios. To the best of our knowledge, it is the first dataset for SGX test scenarios. Due to space constraints, Table I shows 3 test scenarios, omitting the "Reason" column that explains their necessity (details are available in our open-sourced datasets).

B. Results of Test Scenarios

As shown in Table I, we categorized the 43 test scenarios into three domains: security, execution bugs, and performance (i.e., the "Test Domain" column). Overall, the scenarios that fell into the security category are the most (32 of 43), followed by the performance (8 of 43) and execution bugs (6 of 43). Additionally, three scenarios are classified under more than one domain. The "Test Goal" column explains the object of this test scenario. We typically represent the test goal as "*(in order) to something*," e.g., *To safeguard all secrets assigned to production enclaves*.

The "Test Subjects" column outlines the specific elements that each test scenario evaluates. Among our identified scenarios, 62.8% focus on code-related elements, which include ECalls, OCalls, specific functions, and other code-related features or processes. Additionally, 18.6% of scenarios concentrate on data, covering parameters, variables, keys, state information, memory regions, and more. Furthermore, 11.6% of scenarios address configurations, which involve debug flags, enclave attributes, compiler options, and other settings. Lastly, two scenarios pertain to other areas, including development platforms, enclave's size and quantity.

The "Prerequisites" column exhibits the situations or assumptions necessary to perform a test scenario, e.g., *Code outside the enclave passes secret data into the enclave*. The "Behavior" column dictates the expected behaviors for the test subjects, e.g., *Be configured as false*.

The "Importance" column indicates the significance of the test scenarios, including levels such as Must, Should, May, and Recommendation. Each importance level is derived from Intel's SGX Developer Guide, which uses these terms to describe the content related to our identified test scenarios. Out of 43 scenarios, there are 17 Must, 14 Should, 1 May, 10 Recommendations, and 1 NA. We assigned one scenario an importance level of NA, as we could not find any relevant expression of importance level for this scenario in the guide.

C. Formalizing Test Scenarios and Oracles

Based on the attributes of test scenarios (i.e., column labels in Table I), we formalize the scenarios and their oracles.

a) For Test Scenarios: Let S denote the set of test subjects, s an individual subject. Let P, B denote the prerequisites and expected behaviors of a certain test scenario, respectively. $B(s)$ denotes the predicate representing the behavior B for a given subject s. O denotes deontic operators, including must, should, may, and recommend, which correspond to the importance levels. A test scenario can be described as follows.

$$P \rightarrow \text{Verify}(\forall s \in S,\ \text{O}(B(s)))$$

That is, given that P, then verify that for all test subjects s in S, it is obligatory (must/should/may/recommend) that they exhibit expected behaviors B. For instance, the scenario in the first row of Table I can be described as "given *code outside the enclave passes secret data into the enclave*, then verify that for all *Enclave's debug flags must be configured as false*."

b) For Test Oracles: The elements above can also formalize the test oracles. Let pass(s) denote a predicate that is true if the test case for subject s passes, and fail(s) denote a predicate that is true if the test case for subject s fails. Then, a test oracle can be described as follows.

$$P \rightarrow \forall s \in S\,[(B(s) \rightarrow \text{pass}(s)) \wedge (\neg B(s) \rightarrow \text{fail}(s))]$$

That is, given that P, then for all test subjects s in the set S, if the expected behavior B occurs for subject s, the test case for that subject passes, otherwise, it fails. For instance, the above

scenario's corresponding oracle can be described as "given *code outside the enclave passes secret data into the enclave*, then for all *Enclave's debug flags*, if they *are configured as false*, then the case *passes*, otherwise, it *fails*."

D. Principles of Testing in SGX

We identified four testing principles from the above dataset.

P-1: Never trust the outside world. SGX's threat model presents that only the enclave is secure, while everything outside it can be compromised at any time. Hence, the code and data within the enclave must never trust the outside world and should always be prepared for unexpected outside behaviors. In terms of testing, an SGX project should always check the validity and reliability of information originating from the outside world. Furthermore, the project must incorporate necessary handler code to enhance resilience against unexpected behaviors executed by entities outside the enclave.

P-2: Inside code may run into issues. While the outside world should never be trusted, there are also risks associated with the code within the enclave, including security vulnerabilities, execution bugs, and performance issues. In general, the more code included within the enclave, the greater the possibility for security risks and bugs, and the larger the potential for performance cost. It is crucial for SGX projects to minimize the enclave size and ensure high code quality within the enclave. In terms of testing, an SGX project should verify the necessity of the code inside the enclave and ensure that the code adheres to proper security and performance guidelines.

P-3: SGX itself may not be entirely reliable. Nothing is perfect, nor is SGX. Despite its robust protection mechanisms, SGX has inherent limitations and vulnerabilities. For example, its enclaves do not support certain hardware features and instructions. Furthermore, SGX is not naturally immune to specific types of attacks (e.g., side-channel attacks). In terms of testing, an SGX project should contain proper handler code or mitigation strategies to protect against vulnerabilities and minimize limitations, which should be ensured through testing.

P-4: Configuration matters. Like other complex systems, SGX includes many configurable attributes, such as debug flags, enclave attributes, compiler options, and more. An improper configuration may lead to security risks or trigger execution bugs. Hence, verifying that these configurations are properly set is a crucial aspect of SGX testing.

E. Overlooked Runtime Tests

All the principles above contain runtime issues. For instance, the behavior of functions outside of the enclave can change at runtime (P-1), enclave code may fail and raise exceptions at runtime (P-2), attacks can be carried out by malicious actors at runtime (P-3), and configurations may be tampered with at runtime (P-4). However, we found no explicit runtime test scenarios or code in our datasets or the Intel SGX Developer Guide. To assist SGX testing and security engineers, we recommend focusing on two key aspects:

a) For P-1 and P-4: We recommend building a monitor that checks the following at runtime: (1) The order of functions within enclaves called by external entities to ensure it is correct; (2) OCall behaviors to verify they meet expectations, including scenarios where they are not executed as anticipated or are not executed at all; (3) The enclave loading process to confirm that correct hosts load it and that the loading proceeds as expected; (4) The configuration values to ensure they are set correctly; (5) All of the above elements should be verified and remain untouched thereafter.

b) For P-2 and P-3: Predicting code failures and effectively defending against attacks at runtime is challenging. While incorporating necessary error handling into code is important, we can also approach runtime failures and attacks as valuable test cases. This method, which we call the "Failure/Attack as a Test Case" philosophy, aims to collect information for further analysis. Hence, we recommend that the execution code include necessary logging and a log dump feature to capture useful information, even in the event of a system crash, enabling more effective subsequent testing.

VI. SGX in the Cloud

This section outlines our method for gathering user feedback on cloud-based SGX and discusses our findings.

A. Methodology & Dataset

a) A Search-based Approach: We create a dataset of user feedback on "SGX in the cloud" by searching online resources from Intel-recognized cloud vendors (Microsoft Azure [26], [27], Alibaba Cloud [5], IBM Cloud [14], [15], OVH Cloud [30]) and the StackExchange network [16]. In particular, we search for the keyword "sgx" on cloud vendors' websites. For the StackExchange network, we use a crawler to look for "cloud" and filter results for "sgx" in titles and content to ensure relevance. We also manually search for "sgx, cloud" on the StackExchange site to maximize our results. After that, we manually analyze each record to remove invalid entries. A record is considered valid only if it pertains to both SGX and cloud computing, such as remote attestation (i.e., verifying whether the cloud server executes code within enclaves) and SGX support for Golang (i.e., a language widely used in cloud services). We also remove records that are not from users, such as technical instructions from cloud providers.

b) Dataset: From the cloud vendors' websites, we identified 6 valid records out of 23 searched results using the aforementioned search method. From the StackExchange network, we initially found 235,268 records in our first round of searching for "cloud". After filtering for "sgx," we obtained 38 relevant records. Additionally, a manual search for "sgx, cloud" yielded 37 records. Among these records from the StackExchange network, we identified 10 valid ones. In total, our user-based dataset consists of 16 valid records.

B. Results and Findings

Finding 4: *From the users' perspective, "SGX in the cloud" appears to have received little attention and has not been widely adopted.* As a mature commercial technology has been rolled out for 10 years, we assumed that there would be plenty of feedback, questions, and discussions about SGX online. Surprisingly, we found only 16 valid records regarding SGX

in the cloud[3]. Similarly, for SGX-supported cloud platforms, there is a noticeable lack of user questions or community posts that explicitly reference SGX. Notably, both Alibaba Cloud and IBM Cloud, each of which maintains large, actively-moderated forums and dedicated user-feedback portals, show no trace of SGX-related discussions.

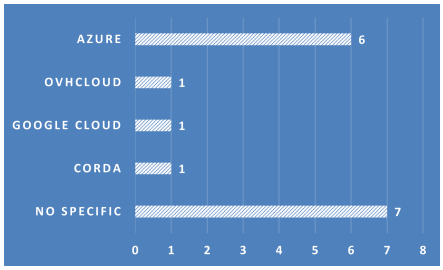

Fig. 4. Targeted Cloud Platforms for SGX.

In our dataset, as shown in Figure 4, the most frequently targeted cloud platform for SGX is Microsoft Azure (6 posts), followed by OVH Cloud, Google Cloud, and Corda (1 post for each). The remaining posts are not directed toward any specific cloud platform. One possible reason for the phenomenon is that SGX always serves as a foundational component rather than as a user-facing application. As a result, users—such as developers, testers, and end-users—might either not notice SGX's presence or have limited opportunities to use it for implementing specific features. We plan to conduct a future user survey to better understand this phenomenon.

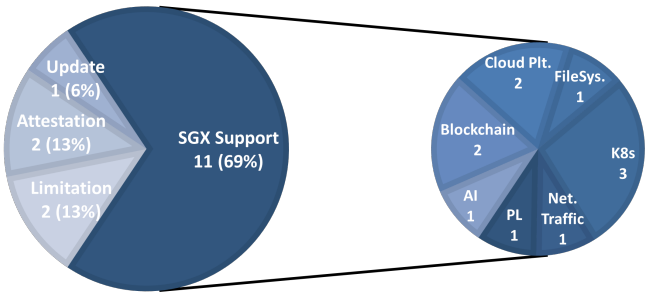

Fig. 5. User-concerned Scenarios for SGX in the Cloud

Finding 5: *The most user-concerned scenarios for SGX in the cloud is about SGX support.* As shown on the left side of Figure 5, 69% of user discussions revolve around how or if SGX can support a platform or technique. The other topics include SGX limitations (e.g., its memory capacities), remote attestation (e.g., ensure code runs in enclaves), and Intel's SGX updates (e.g., security patches).

Within the "SGX support", as illustrated on the right side of Figure 5, there are a variety of application scenarios. These include SGX support for Kubernetes (3 posts), blockchain (2 posts), cloud platforms (2 posts), file systems (1 post), network traffic data (1 post), programming languages (1 post), and AI (1 post). We have made the detailed posts available online.

[3]Searching for "sgx" on StackExchange returns 1,352 records, while "cloud" results in 235,268. This indicates that SGX topics, even without cloud adoption, are not very popular among users.

C. Insights about Testing SGX in the Cloud

a) Importance: Finding 4 implies the significance of testing SGX in the cloud. That is, cloud developers and testers may easily overlook SGX, resulting in neglect of maintaining and executing test code. Furthermore, if the SGX-related code that has been overlooked contains errors, debugging or resolving these issues can be particularly challenging.

b) Variety: Finding 5 emphasizes the variety of testing scenarios for SGX in the cloud. That is, it is essential to create test cases for specific scenarios such as Kubernetes, file systems, blockchain, AI, and more. Additionally, SGX's inherent attributes and features, such as memory limitations and remote attestation, should also be verified. Further, we need to verify whether updates to Intel's SGX could introduce security vulnerabilities or execution bugs in cloud services.

VII. Threats to Validity

The internal validity is threatened by our manual analysis method. That is, when analyzing the SGX-related test code, test scenarios, and user feedback, our manual analysis process may lead to the loss of valid records. To mitigate this threat, we involved AI-assisted analysis for the test code, and repeatedly reviewed our scenario results and user feedback records more than four times. *The external validity* is threatened by the limitations of our data sources. That is, the collected SGX projects and test codes may not fully represent real-world testing characteristics. Additionally, our analysis of Intel's official documentation and our search of online sources may not have covered every possible test scenario or relevant user posts on cloud-based SGX. To mitigate these threats, we have made all of our datasets available online, allowing practitioners to contribute additional information and insights.

VIII. Related Work

1) Survey on SGX: Intel's SGX has attracted significant research attention as a commercial TEE solution. Zheng et al. conducted a literature review of SGX, analyzing its applications, attack approaches, and its pros and cons from over 100 papers [41]. Nilsson et al. examined the literature and extracted 24 distinct attacks against SGX, which they then organized into 7 categories [29]. Will et al. surveyed 293 papers and classified SGX applications by goals and contexts. [39]. Fei et al. surveyed existing SGX security vulnerabilities and countermeasures, and introduced the first two sets of criteria to evaluate them [9]. Van Schaik et al. surveyed and categorized publicly known SGX hardware attacks [36]. Although these studies revealed SGX's application scenarios, vulnerabilities, attacks, and defense strategies, they overlooked the status quo of testing in SGX. Additionally, they based their survey primarily on papers rather than real-world projects.

2) Testing on SGX or other TEE solutions: Prior studies of testing on SGX can be divided into two main groups: performance benchmarking of the SGX system and fuzz testing of SGX projects. In the first group, Kumar et al. proposed a benchmark suite for SGX and listed parameter configurations for ten tested workloads [23]. Hasan et al. benchmarked SGX

to evaluate which workloads are better for porting or shimming and the performance effects of each [13]. Coppolino et al. conducted performance testing and comparison of four TEE solutions under real-world conditions [8]. The second group consists of fuzzing frameworks that identify vulnerabilities in SGX projects, including EnclaveFuzz [6], FUZZSGX [21], SEnFuzzer [40], SGXFUZZ [7], etc. These research studies have improved SGX-related testing methods but lack connections to real-world test scenarios and oracles.

3) Survey on SGX/TEE in the Cloud: There is a scarcity of comprehensive surveys on SGX or TEE in the cloud, and none of them focus on the user's perspective. Miriyala et al. compared the SGX services of AWS and Azure [28]. Li et al. surveyed three types of TEE-based secure computation protocols that can be applied in cloud [25]. Lei et al. studied SGX's application in blockchain, highlighting challenges in four layers and the solutions it provides [24].

IX. Conclusion

In this paper, we conduct a comprehensive study on SGX's testing, involving three components: (1) studying real-world test code, (2) analyzing and formalizing test scenarios and oracles, and (3) exploring users' SGX practices in the cloud. Our work resulted in three datasets and a series of findings that can inform and guide both academia and industry practitioners in improving SGX and other TEE-based applications.

Acknowledgements

We thank the anonymous reviewers for their valuable comments that enhanced this paper. This research is supported by Beijing Natural Science Foundation (Grant No. 4232019).

References

[1] "CVE-2017-6131." www.cve.org/CVERecord?id=CVE-2017-6131.
[2] "CVE-2021-39014." www.cve.org/CVERecord?id=CVE-2021-39014.
[3] "CVE-2022-39397." www.cve.org/CVERecord?id=CVE-2022-39397.
[4] "CVE-2023-32990." www.cve.org/CVERecord?id=CVE-2023-32990.
[5] Alibaba, "Alibaba Cloud," 2025, https://www.aliyun.com/.
[6] L. Chen, Z. Li, Z. Ma, Y. Li, B. Chen, and C. Zhang, "Enclavefuzz: Finding vulnerabilities in sgx applications," in *Network and Distributed System Security (NDSS) Symposium*, 2024.
[7] T. Cloosters, J. Willbold, T. Holz, and L. Davi, "{SGXFuzz}: Efficiently synthesizing nested structures for {SGX} enclave fuzzing," in *31st USENIX Security Symposium (USENIX Security)*, 2022, pp. 3147–3164.
[8] L. Coppolino, S. D'Antonio, G. Mazzeo, and L. Romano, "An experimental evaluation of tee technology: Benchmarking transparent approaches based on sgx, sev, and tdx," *Computers & Security*, vol. 154, p. 104457, 2025.
[9] S. Fei, Z. Yan, W. Ding, and H. Xie, "Security vulnerabilities of sgx and countermeasures: A survey," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–36, 2021.
[10] Y. Gao, X. Hu, T. Xu, X. Xia, D. Lo, and X. Yang, "Mut: Human-in-the-loop unit test migration," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.
[11] GitHub, Inc., "GitHub REST API documentation," 2025, https://docs.github.com/en/rest?apiVersion=2022-11-28l.
[12] GlobalPlatform, "GlobalPlatform, TEE system architecture, technical report," 2022, https://globalplatform.org/specs-library/tee-system-architecture/.
[13] A. Hasan, R. Riley, and D. Ponomarev, "Port or shim? stress testing application performance on intel sgx," in *2020 IEEE International Symposium on Workload Characterization*. IEEE, 2020, pp. 123–133.
[14] IBM, "IBM - Cloud, PowerVS and Ceph aaS - Structured Ideas," 2025, https://ibmcloud.ideas.ibm.com/.
[15] ——, "IBM TechXChange Community," 2025, https://community.ibm.com/community/user/my-community.
[16] S. E. Inc, "Stack Exchange," 2025, https://stackexchange.com/.
[17] Intel, "Intel® Software Guard Extensions (Intel® SGX) SDK for Windows* OS ," 2020, https://cdrdv2-public.intel.com/671508/sgx-sdk-developer-reference-for-windows-os.pdf.
[18] ——, "Intel Processors Supporting Intel SGX," 2025, https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions-processors.html.
[19] ——, "Intel Software Guard Extensions (SGX)," 2025, https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html.
[20] ——, "Intel® Software Guard Extensions Developer Guide," 2025, https://www.intel.com/content/www/us/en/content-details/738855/intel-software-guard-extensions-developer-guide.html.
[21] A. Khan, M. Zou, K. Kim, D. Xu, A. Bianchi, and D. J. Tian, "Fuzzing sgx enclaves via host program mutations," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023.
[22] D. J. Kim, J. Yang, and T.-H. Chen, "A first look at the inheritance-induced redundant test execution," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024.
[23] S. Kumar, A. Panda, and S. R. Sarangi, "A comprehensive benchmark suite for intel sgx," *arXiv preprint arXiv:2205.06415*, 2022.
[24] H. Lei, Q. Wang, W. Shi, and Z. Bao, "A survey on the application of sgx in blockchain area," in *International Conference on Blockchain and Trustworthy Systems*. Springer, 2020, pp. 633–647.
[25] X. Li, B. Zhao, G. Yang, T. Xiang, J. Weng, and R. H. Deng, "A survey of secure computation using trusted execution environments," *arXiv preprint arXiv:2302.12150*, 2023.
[26] Microsoft, "Microsoft Azure Share your Ideas," 2025, https://feedback.azure.com/d365community/.
[27] ——, "Microsoft Community Hub," 2025, https://techcommunity.microsoft.com/.
[28] N. S. Miriyala, K. B. Macha, S. Metha, and D. Dave, "Comparative review of aws and azure confidential computing systems," 2024.
[29] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on intel sgx," *arXiv preprint arXiv:2006.13598*, 2020.
[30] OVH SAS., "OVHcloud Community," 2025, https://community.ovhcloud.com/community.
[31] L. Pryimenko, "7 examples of real-life data breaches caused by insider threats," 2024, https://www.syteca.com/en/blog/real-life-examples-insider-threat-caused-breaches.
[32] A. Rao, "Rising to the Challenge — Data Security with Intel Confidential Computing," 2022, https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Rising-to-the-Challenge-Data-Security-with-Intel-Confidential/post/1353141.
[33] The New Daily, "Federal government to force tech giants to reveal user data," 2018, https://thenewdaily.com.au/news/national/2018/08/14/tech-surveillance-laws/.
[34] F. Toffalini, M. Payer, J. Zhou, and L. Cavallaro, "Designing a provenance analysis for sgx enclaves," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 102–116.
[35] S. Van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "Sgaxe: How sgx fails in practice," 2020.
[36] S. Van Schaik, A. Seto, T. Yurek, A. Batori, B. AlBassam, D. Genkin, A. Miller, E. Ronen, Y. Yarom, and C. Garman, "Sok: SGX. Fail: How stuff gets exposed," in *2024 IEEE symposium on security and privacy (SP)*. IEEE, 2024, pp. 4143–4162.
[37] J. Wang, Y. Cheng, Q. Li, and Y. Jiang, "Interface-based side channel attack against intel sgx," *arXiv preprint arXiv:1811.05378*, 2018.
[38] N. Weichbrodt, P.-L. Aublin, and R. Kapitza, "sgx-perf: A performance analysis tool for intel sgx enclaves," in *Proceedings of the 19th International Middleware Conference*, 2018, pp. 201–213.
[39] N. C. Will and C. A. Maziero, "Intel software guard extensions applications: A survey," *ACM Computing Surveys*, vol. 55, no. 14s, 2023.
[40] D. Yu, J. Wang, H. Fang, Y. Fang, and Y. Zhang, "Senfuzzer: Detecting sgx memory corruption via information feedback and tailored interface analysis," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, pp. 485–498.
[41] W. Zheng, Y. Wu, X. Wu, C. Feng, Y. Sui, X. Luo, and Y. Zhou, "A survey of intel sgx and its applications," *Frontiers of Computer Science*, vol. 15, no. 3, p. 153808, 2021.