Extensive and Secure Data Management System with Vulnerable Extension Code

Xinqing Li, Iulian Sandu Popa, Nicolas Anciaux

Extensive and Secure Data Management System with Vulnerable Extension Code

Xinqing Li[1,2], Iulian Sandu Popa[1,2], and Nicolas Anciaux[1,2]

[1]PETRUS project-team, Inria de Saclay
[2]DAVID Lab., Univ. Versailles St-Q.-en-Yvelines, Univ. Paris Saclay

1 Abstract

This paper introduces a novel database management system architecture designed to address the Honest-but-Vulnerable threat model within Intel SGX environments. We propose an extensive architecture to optimize the balance between security and functionality. We outline our planned workflow and security mechanisms, including an Only-In-Never-Out policy to safeguard data confidentiality and integrity. Future work will focus on implementing and evaluating this architecture to assess its effectiveness in enhancing cloud database security.

2 Introduction

Cloud storage services provided by enterprises such as Apple, Google, Amazon, are commonly deployed in our daily life, facilitating the storage and manipulation of vast amounts of data by both individuals and enterprises. However, those centralised data management systems raise concerns about data security since even a minor software vulnerability could lead to significant social repercussions [13].

To address these growing concerns about data security, the Trusted Execution Environment (TEE) is recognized as one of the most important methods to enhance system security in a cloud context, and it can largely mitigate the problem of data leakage during data processing. Intel's Software Guard Extensions (SGX), introduced in 2014, is a prominent TEE solution. It proposes the use of enclaves — an isolated execution environment that protects data even if system-level security is compromised. Within these enclaves, data is processed at the user level (ring 3) and encrypted by the CPU (ring 0), ensuring that sensitive information remains confidential. Interactions between the enclave and the rest of the application are managed only through well-defined interfaces: ECALLs allow the external application to invoke functions within the enclave, and OCALLs permit the enclave to request services from the operating system or external applications. These mechanisms ensure that even if an attacker gains control over the operating system kernel, the integrity and confidentiality of the data (and code execution) within the enclave are maintained.

As cloud services routinely handle (large and diverse types of) data from multiple clients, ensuring data security becomes paramount. However, the complexity and variety of the code required to process these data introduce security vulnerabilities. This research seeks to identify the key challenges and vulnerabilities associated with the code used in cloud storage systems and to propose robust methods to ensure that the confidentiality of client data is preserved despite inherent vulnerabilities. The study focuses on developing a framework that enhances data security without compromising the functionality or efficiency of cloud services, examining various strategies to protect data while acknowledging the practical limitations and potential weaknesses of the code handling it.

To preserve the confidentiality of sensitive data from multiple clients, secure database outsourcing employs Application-level encryption [1]. In this model, the client generates an encryption key and uses it to encrypt their data before uploading it to the cloud. This approach has the advantage of separating encryption keys from the encrypted data stored in the cloud, as the keys never leave the application side, protecting the data from the cloud provider and any unauthorized third parties. However, applications will need to be modified to adopt this solution, as some of the database processing code will need to run on the client side. In addition, depending on the granularity of the encryption, the application may need to retrieve a larger set of data than that granted to the actual user, creating a security breach. In fact, the user (or any attacker who gains access to the machine running the application) could hack the application to access unauthorised data.

Intel SGX emerges as an alternative with numerous SGX-based databases developed since its inception. These databases typically migrate parts of classical databases into an enclave [17, 7], aiming to secure the most critical execution segments through enclave encryption. Furthermore, it is now feasible to migrate the entire database into an enclave [22, 11, 18], as SGX version 2 offers flexible large memory allocation [10] within enclaves. This approach assumes that the code within an enclave is trustworthy. However, Intel places the onus of code verification on developers, and in practice it is a challenge to verify millions of lines of code to ensure their trustworthiness. Moreover, Intel SGX does not address data leakage due to code vulnerabilities because it assumes that the enclave code is free of vulnerabilities. This oversight allows attackers to exploit these vulnerabilities to leak sensitive information [12, 16, 6]. We refer to this threat model as *Honest-but-Vulnerable*, where the code is designed for a benign purpose and respects the execution order, but where existing vulnerabilities could be exploited to leak information. Few solutions address this problem, and most focus on designing vulnerability analyzers [8, 12] that aim to align the code with Intel's assumptions. However, these analyzers cannot guarantee exhaustive detection of vulnerabilities, and their effectiveness decreases as the code base grows.

We propose a novel architecture for our data management system focusing on data storage and indexing, that significantly reduces the trusted computing base (see Figure 1). The system is divided into two components: the *Core Enclave* and the *Task Enclave*. The Core Enclave contains a minimal, trusted code requiring verification, handling critical security functions. In contrast, the Task Enclave processes data using untrusted code, allowing for the incorporation of various data treatment methods without extensive verification. With this architecture, we only need to verify a minimum part of the code inside enclaves (i.e., the Core part), and we can attach various untrusted data treatment methods to our system without code verification, enhancing flexibility and scalability while maintaining security.

In this paper, we aim to contribute in two significant ways: First, we define the threat model "Honest-but-Vulnerable," distinguishing it from other attacks on SGX. This model addresses a gap in existing literature by closely aligning with real-world development scenarios where code is intended to be secure but may still contain exploitable vulnerabilities. Second, we propose a novel architecture for a data management system that balances security with extensibility. Since the system is divided into two parts, the Core will be the only part that needs code trustworthiness, and the Core in our design is minimized so that it is easy to verify. Another part consists of the Task enclaves; this section is extensive enough to incorporate various untrusted codes into our system without code verification. The Core establishes the sole communication channel with the client, ensuring that all results from the Task enclaves are examined and not sent directly to clients. This design enables enhanced protection for cloud storage services while supporting a flexible integration of various data processing methods.

3 State of Art & Related Work

3.1 Attacks on SGX

Although Intel officially states that SGX can thwart kernel-level attacks, several works have proven otherwise. According to Zhang [23], these attacks fall into four categories: side-channel attacks, hardware vulnerabilities, Denial of Service (DoS) attacks, and code vulnerabilities. Side-channel attacks exploit structural elements like page-tables in untrusted memory to observe enclave behavior, a view supported by Nilsson [15] who details various techniques. Hardware vulnerabilities, notably the Foreshadow Attack (also known as L1TF) [3], bypass the hardware barriers SGX aims to establish, allowing attackers to access sensitive data. DoS attacks exploit SGX's integrity checks to overload and disrupt enclave operations. These categories represent intrinsic SGX vulnerabilities rather than deviations from Intel's official threat model. Consequently, this paper will not address these types of attacks.

SGX is also susceptible to software-level vulnerabilities. As outlined by COIN [12], if an attacker gains control of the kernel, they can launch multiple ECALLs of the same enclave concurrently, leading to execution corruption. Additionally, launching enclave functions in an arbitrary order can disrupt data workflow and system operations. These issues stem from SGX's lack of verification mechanisms for function calls within untrusted memory, operating under the assumption that these cannot be manipulated by an attacker in the host application. This type of vulnerability does not necessarily lead to data leakage but to a system dump that affects system functionality. Therefore, in this paper, we will not include this type of vulnerability.

SGX could also leak sensitive data inside an enclave by code vulnerabilities. Intel promises confidentiality through SGX if the code inside enclaves is entirely trustworthy. However, verifying code integrity in real-world applications, especially for large projects with complex algorithms, is highly challenging. This unmet assumption opens a pathway for attackers to exploit code vulnerabilities, such as memory corruption [2] and

use-after-free [12], to leak sensitive data from enclaves. Such vulnerabilities can lead to memory alterations via malicious code injections, notably through Return-Oriented Programming (ROP) attacks. For instance, in SnakeSGX [19], attackers exploit known code flaws and the layout of enclave memory to inject malicious code that leaves minimal traces yet can repeatedly leak data. This exploitation is possible because SGX does not verify the integrity of ORET, which contains pointers that return control to the enclave, allowing attackers to alter execution order. Additionally, techniques like those seen in SmashSGX [9] and SGXDump [21] utilize exception handling mechanisms and page-table observations to deduce enclave states, facilitating data leakage. Similarly, Dark-ROP [14] and 'A Tale of Two World'[20] combine ROP attacks with side-channel observations to extract data comprehensively from enclaves, demonstrating the critical interplay of these vulnerabilities.

Apart from ROP attacks, other vulnerabilities like memory corruption can also lead to data leakage within enclaves without altering the execution order. In SGX, the input verification mechanism is inadequate, enabling attackers to exploit input manipulation attacks to extract data from enclaves [12]. These attacks are particularly challenging for SGX to prevent, as the inputs to enclaves are customized and therefore difficult to verify comprehensively.

In conclusion, Intel SGX exhibits inherent vulnerabilities at both the hardware and software levels. This paper specifically addresses software vulnerabilities within SGX enclaves, particularly those that do not involve alterations in execution order. By focusing on these types of attacks, we aim to explore vulnerabilities that can occur even without changing the flow of program execution, reflecting more subtle and potentially overlooked security threats. These vulnerabilities represent a critical area of concern, as they can lead to data leakage through manipulations that bypass typical security checks without disrupting the system's operational integrity.

3.2 Databases in SGX

A diverse array of databases has been deployed within SGX, each designed with specific goals and suited for various scenarios. For example, EnclaveDB [17] exemplifies a classical database implementation in SGX, fully trusting the SGX infrastructure and client side, and pre-treating data queries in untrusted memory to minimize the code within the enclave. Similarly, Zhang [22] utilizes the same threat model to develop an ORAM database in SGX. StealthDB [11], based on PostgreSQL, achieves a minimal trusted computing base (TCB) by maintaining most of the system in untrusted memory. Enclage [18] introduces an efficient B^+-tree model with page-level encryption and reduced enclave interaction, enhancing performance over other encrypted databases. Additionally, the multi-tenant EnclaveCache [7] secures client encryption key management within an enclave, preventing key misappropriation among clients.

The existing databases above make assumptions on code trustworthiness inside enclave and demonstrate a significant trade-off between code trustworthiness and system extensibility. Many of these systems operate under the assumption that the code inside enclaves is entirely trustworthy. For instance, databases like EnclaveDB [17] and StealthDB [11] limit the size of their trusted computing base (TCB) to enhance code trustworthiness, subsequently restricting their functionality and adaptability. This design choice constrains the database's ability to process or incorporate new types of data efficiently. On the other hand, some databases do not explicitly address this assumption, potentially compromising the security hypotheses considered by Intel SGX. Thus, while aiming to minimize the enclave's codebase enhances security by adhering threato Intel's guidelines, it significantly hampers the database's ability to extend its functionalities and accommodate innovative data processing algorithms. This inherent conflict underscores a critical paradox in database management within SGX environments: securing the system while trying to maintain its adaptability and functionality. ES-PDMS[5, 4] tackles this issue by assuming the code is developed with implemented attacks, which is different to our threat model. Although it prominently mitigates the risk of data leakage, it does not eliminate the threat posed by malicious code within enclaves.

3.3 Countermeasures against attacks

In this paper, we focus on software-based code vulnerability within SGX, where attackers exploit these vulnerabilities to steal sensitive data protected by enclaves. Several countermeasures have been developed to address these attacks, primarily through vulnerability analyzers [8, 12]. These analyzers implement various examination policies and produce reports to help developers enhance their code. However, they do not guarantee the detection of all vulnerabilities, and their efficiency decreases as the amount of code increases.

In a different approach, Park [16] introduced NestedEnclave, in which SGX is modified at microcode level and enables using new SGX instructions to create hierarchical enclaves. This system allows developers to integrate untrusted library code within an enclave without accessing the trusted code in higher-level enclaves. While this

reduces some risks, it still necessitates substantial code verification efforts for the trusted sections, maintaining a significant burden on developers.

4 Threat Model

A typical cloud database architecture comprises a server hosted on the cloud and multiple clients interacting with it. The server runs a secure enclave handling all data computations, such as queries, insertions and deletions. Upon receiving a request from a client, the server authenticates the client's identity and uses a unique encryption key to decrypt the required data, which is securely stored for each client. After data processing, the server encrypts and sends the results back to the client. Access to the enclave is strictly controlled via a fixed interface, preventing unauthorized client access.

We highlight the code runs inside the server's enclave and the attacker abilities conform to *Honest-but-Vulnerable* attack model. 1) In this model, the code inside the server's enclave is developed with good intentions and follows the prescribed execution order faithfully. 2) However, despite these precautions, the code is susceptible to vulnerabilities, such as buffer overflows and use-after-free errors. 3) These vulnerabilities are known to the attacker, who has comprehensive control over the host application and underlying operating system. While the attacker can analyze the binary files and identify weaknesses, 4) Intel SGX has no intrinsic design vulnerability which means attackers cannot establish unauthorized communication channels or directly observe enclave execution from outside.

To launch an attack, the attacker exploits these vulnerabilities via permissible interactions, such as ECALL/OCALL functions. For instance, if the server processes a query where the output length is determined by the client, the attacker could manipulate this to cause the server to return sensitive data inadvertently. This could occur through improper bounds checking in functions like *memcpy*, where the server might be tricked into copying more data than intended, mixing sensitive information with regular output.

The *Honest-but-Vulnerable* threat model presents a nuanced distinction from the typical *Honest-but-Curious* and *Malicious* models. Unlike the *Honest-but-Curious* model, where the attacker passively observes the data flow without attempting to alter the system's operation, our model involves attackers who actively introduce malicious content. However, these alterations are subtly executed to avoid detection, making them harder to identify contrasting sharply with the *Malicious* model, where attackers overtly disrupt system functionality or cause blatant security breaches. The smart point of this threat model is that, by extracting a single encryption key inside an enclave, all other sensitive data could be decrypted and thus leaked to attacker.This represents a more realistic scenario often encountered in real-world applications, where attackers aim to extract sensitive information discreetly, thereby avoiding aggressive actions that might draw immediate attention from security mechanisms. Our model addresses this middle ground, where the attack is direct and intentional but designed to remain under the radar of conventional security defenses.

5 Architecture

Our database system design, currently under development, aims to mitigate *Honest-but-Vulnerable* attacks by leveraging SGX's security frameworks. The system includes indexation operations that allow clients to PUT, GET and DELETE data with predicate. The architecture is divided into two primary components: the Core and the Task. The Core functions as a minimized trusted computing base (TCB) containing trusted code, while the Task segments handle untrusted code to expand data processing capabilities.

Query Handling: All queries from clients first reach the Core, where they are divided into a uni-format header and a function-specific sub-query tailored by diverse indexation code within Task enclaves. The Core's Simple Pre-treatment module analyzes the header, which specifies the indexation operation, algorithm, and data type, and then routes the functional sub-query to the appropriate Task, awaiting responses.

PUT Operation: For new data entries, the Core distributes a unique identity to each object and performs a hash function to ensure identity integrity and object integrity. In other words, a complete object identity is composed of a random generated identity value and related hash value. The Mapping module stores these objects by their identities and communicates with the Storage Management module, which seals the data into blocks and stores them in untrusted memory. Also, the Hash value of object is also involved to prevent alteration of the sealed blocks. Concurrently, the query is sent to the Task enclave for insertion into an efficient search module, and encrypting necessary information by using a Task-generated encryption key to store in untrusted memory. Upon receiving confirmation from the Task, the Core sends a positive response to the client.

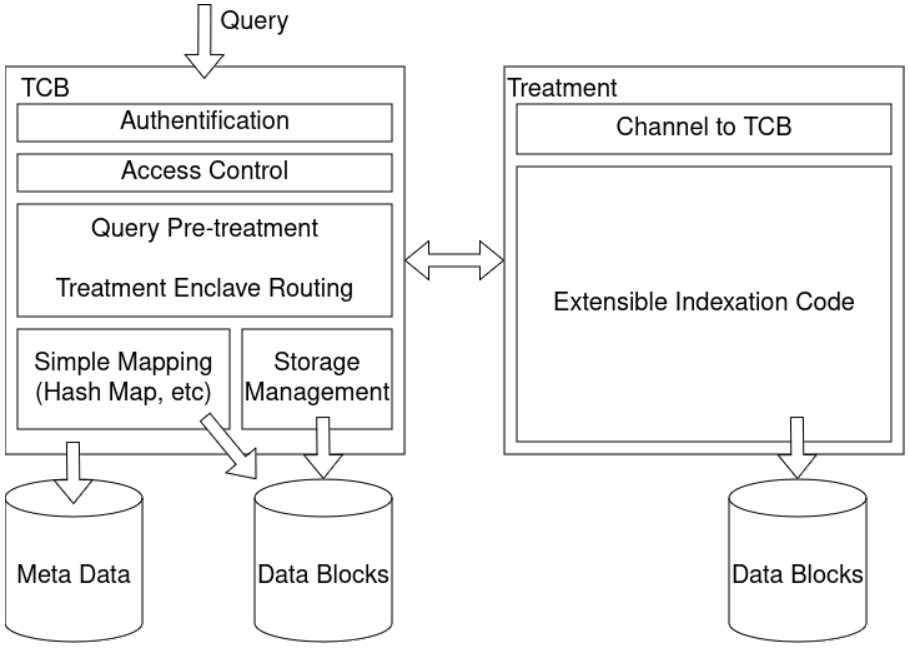

Figure 1: Architecture of database

GET Operation: Similarly, GET queries are pre-processed in the Core and routed to the designated Task enclave. The Task processes the query and sends back data object identities, which the Core uses to retrieve the original data. The Core checks the identity integrity to verify the validity of returned result. After the Access Control module verifies the client's rights, the data is sent to the client.

There are several policies that ensure data confidentiality of our system. Firstly the communication channel between the Core and the client is encrypted. After the authentication, a secret key is exchanged, securing all subsequent communications. The standard header of each query provides the Core with sufficient information for basic indexation operations and Task routing, ensuring that the database always holds a trustworthy copy of all data, and thereby prevents all vulnerability attacks because the Core is verified and trusted.

Secondly, the APIs between the Core and the Tasks are well designed. The Core maintains a API format table includes API for each Task, enabling simple verification on returned results in the Core, to prevent improper memory coping. For instance, during the GET operation, if an encryption key is leaked to the Core, it triggers a system alert due to a failed data identity integrity check.

Furthermore, to prevent potential attacks where the encryption key replaces original value of an object and be send back to a client, our system adheres to an Only-In-Never-Out policy. Only data stored in the Core will be send back to clients, eliminating invalid alteration on object.

Finally, the Access Control module rigorously verifies each client's rights to access the data. In a single enclave, memory corruptions could compromise the Access Control module since all execution inside an enclave share common memory so that a vulnerability in indexation code could also affect other modules. On the contrary, in our design, the Access Control module does not share memory with untrusted code and we could trust the execution result of this module.

Looking ahead, our ongoing work will focus on better characterising the honest-but-vulnerable attack model in terms of classic software bugs. Our ambition is to demonstrate the interest of the proposed system in the face of these vulnerabilities in terms of security and performance. In particular, we aim to evaluate the proposed system under different operational conditions to identify potential bottlenecks and areas for improvement. The challenge remains to maximize system efficiency while maintaining core credibility. Although various solutions for databases are developed in non-SGX environments, SGX development is still immature, necessitating careful selection and optimization of approaches to enhance performance and reliability. In addition, we will investigate enhancements to secure data interactions within the task enclave to mitigate some potential remaining vulnerabilities.

6 Conclusion

In conclusion, this paper has presented a preliminary architecture designed to address the Honest-but-Vulnerable threat model within the context of Intel SGX. By dividing the system into two primary components, the Core and the Task, we aim to strike a balance between maintaining a minimal amount of verified code and enhancing the system's data processing capabilities with extending untrusted code. This approach allows for dynamic extension of functionality without compromising the integrity and confidentiality of the data being processed. Our design not only adheres to stringent security protocols but also introduces a structured workflow that efficiently manages authentication, access control, query processing, and data storage.

References

[1] N. Anciaux, L. Bouganim, and Y. Guo. Database encryption. *Encyclopedia of Cryptography, Security and Privacy*, 2023.

[2] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A.-R. Sadeghi. The Guard's Dilemma: Efficient {Code-Reuse} Attacks Against Intel {SGX}. pages 1213–1227, 2018.

[3] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the Keys to the Intel {SGX} Kingdom with Transient {Out-of-Order} Execution. page 991, 2018.

[4] R. Carpentier, S. P. Iulian, and A. Nicolas. Enabling Secure Data-Driven Applications: An Approach to Personal Data Management using Trusted Execution Environments.

[5] R. Carpentier, F. Thiant, I. S. Popa, N. Anciaux, and L. Bouganim. An Extensive and Secure Personal Data Management System Using SGX. Mar. 2022.

[6] S. Checkoway and H. Shacham. Iago attacks: why the system call API is a bad untrusted RPC interface. *ACM SIGPLAN Notices*, 48(4):253–264, Mar. 2013.

[7] L. Chen, J. Li, R. Ma, H. Guan, and H.-A. Jacobsen. EnclaveCache: A Secure and Scalable Key-value Cache in Multi-tenant Clouds using Intel SGX. In *Proceedings of the 20th International Middleware Conference*, Middleware '19, pages 14–27, New York, NY, USA, Dec. 2019. Association for Computing Machinery.

[8] T. Cloosters, M. Rodler, and L. Davi. {TeeRex}: Discovery and Exploitation of Memory Corruption Vulnerabilities in {SGX} Enclaves. pages 841–858, 2020.

[9] J. Cui, J. Z. Yu, S. Shinde, P. Saxena, and Z. Cai. SmashEx: Smashing SGX Enclaves Using Exceptions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 779–793, New York, NY, USA, Nov. 2021. Association for Computing Machinery.

[10] M. El-Hindi, T. Ziegler, M. Heinrich, A. Lutsch, Z. Zhao, and C. Binnig. Benchmarking the second generation of intel sgx hardware. In *Proceedings of the 18th International Workshop on Data Management on New Hardware*, pages 1–8, 2022.

[11] A. Gribov, D. Vinayagamurthy, and S. Gorbunov. StealthDB: a Scalable Encrypted Database with Full SQL Query Support, Apr. 2019.

[12] M. R. Khandaker, Y. Cheng, Z. Wang, and T. Wei. COIN Attacks: On Insecurity of Enclave Untrusted Interfaces in SGX. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, pages 971–985, New York, NY, USA, Mar. 2020. Association for Computing Machinery.

[13] S. Kyatam, A. Alhayajneh, and T. Hayajneh. Heartbleed attacks implementation and vulnerability. In *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–6, May 2017.

[14] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang. Hacking in Darkness: Return-oriented Programming against Secure Enclaves. pages 523–539, 2017.

[15] A. Nilsson, P. N. Bideh, and J. Brorsson. A Survey of Published Attacks on Intel SGX, June 2020. arXiv:2006.13598 [cs].

[16] J. Park, N. Kang, T. Kim, Y. Kwon, and J. Huh. Nested Enclave: Supporting Fine-grained Hierarchical Isolation with SGX. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 776–789, 2020.

[17] C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A Secure Database Using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 264–278, May 2018.

[18] Y. Sun, S. Wang, H. Li, and F. Li. Building enclave-native storage engines for practical encrypted databases. *Proceedings of the VLDB Endowment*, 14(6):1019–1032, Feb. 2021.

[19] F. Toffalini, M. Graziano, M. Conti, and J. Zhou. SnakeGX: A Sneaky Attack Against SGX Enclaves. In K. Sako and N. O. Tippenhauer, editors, *Applied Cryptography and Network Security*, Lecture Notes in Computer Science, pages 333–362, Cham, 2021. Springer International Publishing.

[20] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens. A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 1741–1758, New York, NY, USA, Nov. 2019. Association for Computing Machinery.

[21] H. Yoon and M. Lee. SGXDump: A Repeatable Code-Reuse Attack for Extracting SGX Enclave Memory. *Applied Sciences*, 12(15):7655, Jan. 2022. Number: 15 Publisher: Multidisciplinary Digital Publishing Institute.

[22] W. Zhang. A Practical Oblivious Cloud Storage System based on TEE and Client Gateway. In *2021 18th International Conference on Privacy, Security and Trust (PST)*, pages 1–6, Dec. 2021.

[23] Y. Zhang, M. Zhao, T. Li, and H. Han. Survey of Attacks and Defenses against SGX. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pages 1492–1496, June 2020.