Confidential Computing as an Attempt to Secure Service Provider's Confidential Client Data in a Multi-Tenant Cloud Environment

Bojan Novković, Anita Božić, Marin Golub, Stjepan Groš
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
{bojan.novkovic, anita.bozic, marin.golub, stjepan.gros}@fer.hr

***Abstract*—Cloud-oriented infrastructure posed itself as a predominant deployment paradigm in the recent decade due to its ease of provisioning and relatively low cost. However, entrusting a third party with sensitive data in a multi-tenant environment brings about increased data breach risks.**

The aim of this paper is to give an insight into challenges and threats encountered in mitigating data breach and repudiation risks for service providers utilizing cloud-based environments. Through constructing and studying a possible cloud-based service, we form a corresponding threat model with a special focus on risks originating from internal actors. We explore confidential computing as a possible solution for data confidentiality and mitigation of internal data breach risks for *Software as a Service (SaaS)* providers.

Keywords*—*data privacy, confidential computing

I. Introduction

Undeterred by the increasing awareness of data privacy, data breach risks still pose a significant threat. According to a recent security study of UK businesses, data breaches have incurred significant material costs in 19% of businesses [1], in spite of their rising security resilience. Moreover, in the first half of 2019., as many as 4.1 billion records were exposed due to data breaches [2]. One of the biggest problems is the timely reaction to the attack. Studies show that in 2019., the average data breach detection period was 206 days after the attack took place [3]. It should be noted that, in addition to the loss of reputation and trust, a company facing a data breach also has extremely high financial costs of repairing the damage.

However, analyses of breaches which took place in recent years reveal another worrying trend, a distinct rise of breaches caused by internal actors [4]. Dealing with these trending internal risks while preserving mitigations for existing risks requires a different approach in threat modelling and system design. Moreover, the rise of the cloud-based service deployment paradigm shifted the data storage and processing locus to an untrusted and unattestable environment, introducing new technical challenges and threats in preserving data confidentiality and non-repudiation.

This paper aims to systematize challenges and threats encountered in mitigating data breach and repudiation risks for service providers utilizing cloud-based infrastructure and give an overview of promising research developments in confidential computing which could help curb risks of data breaches. Taking the recent emergence of internal risks and harsh data breach sanctions into account, we take a different approach in system architecture analysis and threat modelling. The difference is that in our model the service provider does not have access to sensitive personal data or, in the worst case, it is minimized and strictly controlled. Thus, the majority of this paper focuses on exploring different options for protecting confidential client data from the service provider itself as an attempt to simultaneously protect sensitive client data and mitigate internal data breach risks and non-repudiation attempts.

The rest of this paper is organized as follows. In the second section we form and analyze an example of a possible system deployed on the cloud, describing its architecture and actors. We then proceed to build a corresponding threat model in the third section, paying close attention to risks involving internal actors. The fourth section presents a brief overview of confidential computing along with an overview of state-of-the-art approaches in securing databases. Finally, in the fifth section we give a short discussion regarding possible applications of confidential computing to our threat model and conclude the paper.

II. System architecture

To create a threat model we can later use to consider solutions to our security problems, we need to draw a system architecture diagram modelling the flow of critical data. We focus solely on cloud-based *Software as a Service (SaaS)* systems, which brings about a need for a strong distinction between two crucial and often synonymous actors: the *cloud provider* and the *service provider*. The *cloud provider* is an external actor which provides the infrastructure and the deployment platform that the *service provider* is using.

To illustrate the design of system architecture diagram we will later use for threat modelling let us take an example shown in Figure 1. By studying this example of system architecture, it is clear that the stakeholders in this system are service providers, clients, and end users. This example shows a system containing different databases owned by different stakeholders, the administration service (*A*) and applications' services, user

interfaces and external systems. These components are divided into two large groups: server side components and client side components. The service provider owns the administration service (*A*), the administrator database (*DB Admin*), as well as the database that clients have available to store their data (*DBSP*). Clients have access to Admin Centre and Main App. This leads to the conclusion that clients are both administrators of their service instance and service users. Services can also be accessed through External Systems. End users can access services only through End User Portal GUI.

Shaded databases are clients' databases which can be stored both on cloud or locally on their own servers. The service provider is in control of databases *DB Admin* and *DBSP*, the administration service (*A*) and their services (S_1 *to* S_n).

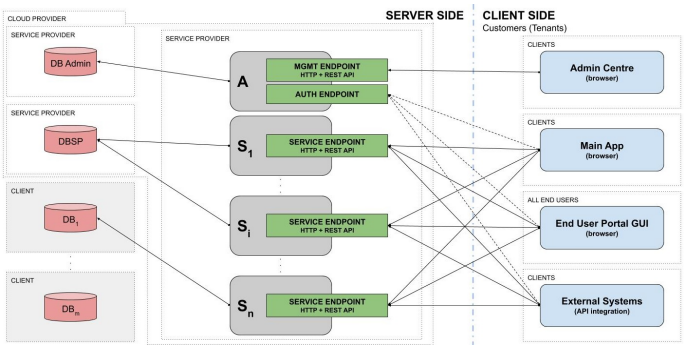

Fig. 1: An example of a system architecture diagram

Administrators connect to the administrator service's (*A*) management endpoint (*MGMT ENDPOINT*) using browser (*Admin Centre*). Both management endpoint and authentication endpoint (*AUTH ENDPOINT*) are using the administrator database (*DB Admin*) to read, as well as store, credentials and authorization data. To provide end users a particular service, clients need to store service data in a database of their choice. Clients can either use service provider's database (*DBSP*) where a larger number of different clients store data or they can use their own database (DB_1 *to* DB_m) that stores only their data.

End users and clients use a service provider's service (S_1 *to* S_n) by first logging in to a service using authentication endpoint (*AUTH ENDPOINT*). Clients connect to their service instance (S_i) using the *Main App*, while end users connect using the *End User Portal GUI*. Services are accessed using browser. Client's applications use the service provider's service in the form of an application programming interface (API).

III. Threat model

Threat modelling is a systematic approach to identify possible security threats to a system in order to protect it [5]. While it is not a guarantee that all possible threats will be identified, it represents a good start towards building a secure and stable system. There are many threat modelling methods available, the most well-known of which are STRIDE [6] and PASTA [7]. In this paper Microsoft's STRIDE method will be used. The acronym STRIDE stands for spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privileges. Definitions for all of these threats can be found in Table I.

TABLE I: STRIDE model acronym description [8]

Category	Description
Spoofing	Impersonating another person/process
Tampering	Unauthorized alterations
Repudiation	Denying claims/unproven actions
Information disclosure	Exposure to unauthorized person/process
Denial of service	Service unavailability
Elevation of privileges	Increasing person/process access level

Thread modelling process consists of a few steps [8]:

1) profiling the application (defining users, data elements, used technologies, security mechanisms etc.),
2) decomposing the application (trust boundaries, entry points, exit points, data flows),
3) identifying threats (creating the threat list),
4) identifying vulnerabilities
5) and ranking the threats.

In the next subsections we will build the threat model for the example system using the system architecture shown in Figure 1. In the first subsection we will focus on profiling and decomposing the system. This subsection is of great importance as we are especially trying to protect the system from malicious internal actors. We are after that proceeding to defining security requirements of our example system in the third subsection. Finally, in the fourth subsection, we identify possible threats and vulnerabilities in the system.

A. Building a threat model of the example system

To begin with, it is necessary to identify the stakeholders and data elements in the system and what operations the stakeholders will be able to perform on the data [8]. According to the system architecture shown in Figure 1, the stakeholders are:

- the service provider,
- clients,
- end users
- and external hackers.

The next step is to determine the trust boundaries that indicate the change in the trust level, the entry and exit points of the system, and the data flows [8]. A lot of this information can be displayed through threat model data flow diagrams that show the communication of different stakeholders in the system in a simple and visual way. The threat model data flow diagram helps to further identify possible threats according to the STRIDE model.

Figure 2 shows a threat model data flow diagram created based on system architecture shown in Figure 1. Threat model data flow diagram contains several groups of elements. Rectangles represent stakeholders. Figure 2 shows browser as a stakeholder since users access services through browser. Circles represent processes, in this case processes are authentication API and a client's service. Two parallel lines represent data stores which would include databases, cache memory etc. Data stores shown in Figure 2 are browser cache, administrator database (*DB Admin*), service provider's

database (*DBSP*) and client's database (DB_i). Arrows show data flows between processes, data stores and stakeholders. Tick dashed line crossing represents a trust boundary between stakeholder's local network (client side) and cloud service (server side). Here the client and the server side refer to client-server architecture. To the left of the trust boundary are clients, end users and external hackers whereas to the right is the service provider. Data crossing this boundary could be exposed to attacks.

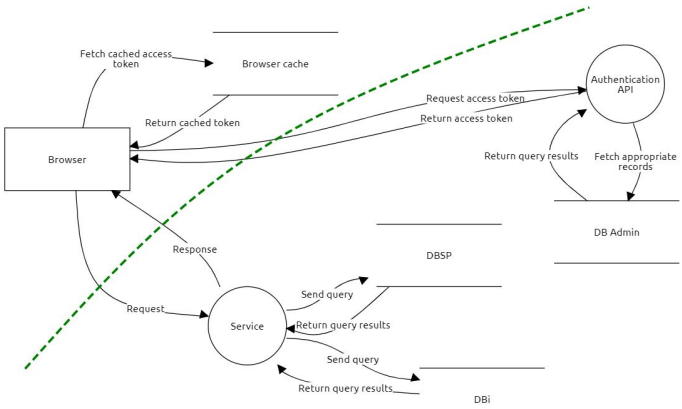

Fig. 2: Threat model data flow diagram based on system architecture shown in Figure 1

After the threat model data flow diagram shown in Figure 2 is created, it is necessary to identify the threats in the system using STRIDE model.

1) Spoofing: When it comes to spoofing, it is possible to point out several cases of impersonating another person/process:

- the service provider impersonates a client - the service provider can directly manage the data of that client's end users,
- the service provider impersonates an end user - the service provider can read data that is confidential and thus unavailable to both the service provider and a client,
- the service provider impersonates an external hacker - the service provider can get rid of any guilt for unauthorized actions, blaming, in that case, a non-existent external hacker,
- an external hacker impersonates the service provider - an external hacker can gain access to the data using stolen administrator credentials or compromised service.

There are of course many other cases and examples of threats which are not listed here as the listed examples only show threats related to the service provider.

2) Tampering: If the service provider changes the contents of the databases, makes a malicious change within the service or changes the data that is being transmitted within the application, that would be considered tampering.

3) Repudiation: The service provider is at fault if he performs actions related to sensitive data on behalf of a client who did not request it. Likewise, the client may, either intentionally or accidentally, expose the data and blame the service provider.

4) Information disclosure: Information disclosure is a problem that occurs if an unauthorized person has access to data to which they should not have access. The service provider in the role of administrator can directly view the data that they should not have or make a malicious modification to the application in such a way as to gain access to the data.

5) Denial of service: Denial of service can occur if an external hacker performs a DoS attack or if someone deletes data making the service incomplete or unavailable.

6) Elevation of privileges: Elevation of privileges attack enables the regular user to perform actions that demand administrative rights which the regular user does not have.

B. Security requirements

In order to prevent possible threats to the system, it is necessary to implement security mechanisms. Spoofing is resolved by implementing authentication and data tampering is eliminated by preserving integrity. Repudiation is prevented by digitally signing messages to prove that the message was sent from one stakeholder to another, but proof of message reception requires the use of another mechanism. The solution to information disclosure is access control, to denial of service is availability and to elevation of privileges is removing vulnerabilities in the system that lead to elevation of privileges.

C. Examples of vulnerabilities and possible attacks

After profiling and decomposing the application, identifying threats and vulnerabilities needs to be done. Table II shows threats and vulnerabilities with their respective stakeholders and loci grouped by STRIDE categories. This table of course does not show all possible threats but just gives examples to better describe the threat modelling process.

TABLE II: Examples of vulnerabilities categorized using STRIDE model

STRIDE Category	Threat	Stakeholder	Locus
Spoofing	Cross site request forgery	Third party	Browser
	Clickjacking	Third party	Browser
	Domain spoofing	Third party	Browser
	HTML content injection	Third party	Browser
	Imitation of users due to theft and / or inadequate management of session data	Third party	Browser
Tampering	Unauthorized modification of database data	Administrator (service provider)	Cloud instance
	SQL injection	Third party	Browser
	Modifying log files	Administrator (service provider)	Cloud instance
	Inserting content using stored XSS vulnerabilities	Third party/ Malicious client	Browser
	Inserting content into log files	Third party/ Malicious client	Browser

STRIDE Category	Threat	Stakeholder	Locus
Repudiation	Manipulating log files	Administrator (service provider)	Cloud instance
Information disclosure	Theft of sensitive data using XSS vulnerabilities	Third party/ Malicious client	Browser
	Storing and processing sensitive information in plain text	Service provider	Cloud instance
	Data leakage through error pages	Service provider	Browser
	Theft of cryptographic keys	Administrator (service provider)	Cloud instance / Browser
	Traffic monitoring in a multi-tenant cloud environment	Third party/ Cloud service provider	Cloud instance
	Inadequate protection on the transport layer	Service provider	Web server
	Data leakage from the database using SQL injection attacks	Third party	Browser
	Data sharing with external services (third party API)	Service provider	Application
Denial of service	DDoS of authentication endpoint	Third party/ Malicious client	Web server
	DDoS of central portal	Third party/ Malicious client	Web server
	Filling the storage space as a consequence of inadequate log management	Third party/ Malicious client	Cloud instance
Elevation of privileges	Inadequate access control for administrative interfaces	Service provider	Application
	Inadequate access control at endpoints for certain types of HTTP requests	Service provider	Application
	Deficient application logic	Service provider	Application

IV. Confidential computing

Although the client's data confidentiality problem is seemingly insurmountable in cloud environments, recent research offers a new approach in solving this issue. The ever present risk of data breaches, whose consequences were exacerbated by recent strict data privacy regulations, gave rise to a new security model which leverages hardware support to protect data in use. Dubbed *confidential computing*, it combines *trusted execution environments (TEE)* and cryptography to provide data integrity and confidentiality as well as application and memory isolation during runtime [9]. Confidential computing can also be viewed as a natural extension of the *trusted computing* model [10] as it uses a hardware root of trust to provide a tamper-resistant execution environment. It has seen increasing adoption in recent years as major cloud hosting providers introduced *confidential computing* capable virtual machines [11] backed with mature TEE technologies.

A. Trusted execution environment

As Sabt *et al.* [12] have noted, formal definitions and use of the term *trusted execution environment (TEE)* tend to vary and fail to capture various aspects that the term entails. They proceed to define TEEs as a tamper-resistant processing environment which guarantee authenticity, confidentiality and integrity of code, data and runtime states, optionally offering remote attestation [12]. Such an all-encompassing definition further illustrates the TEE's role as a fundamental building block of the *confidential computing* model.

One of the main appeals of applying TEEs to cloud-based workloads is the reduction of the *trusted computing base (TCB)* achieved by removing the cloud provider from the chain of trust. By combining the TCB reduction with offloading computationally expensive cryptographic operations to their hardware implementation, TEEs are able to offer increased security with an acceptable computational overhead.

B. Current technologies

Major hardware vendors have released a variety of hardware-assisted TEE technologies in recent years. Although most existing ISAs were fitted with such technologies [10], we focus on TEE technologies for the `x86-64` ISA since it is predominant in cloud-based environments.

McKeen *et al.* [13] announced Intel's SGX, a novel hardware-assisted TEE technology, in 2013., with first SGX-capable CPUs launching in 2015 [14]. SGX is a hardware feature that extends the ISA with a new set of instructions backed by hardware-assisted TEEs (referred to as *enclaves*), designed to provide integrity and confidentiality for sensitive computations in user applications [10], [15]. Its security model drastically reduces the TCB to only the CPU. Moreover, the security model includes physical attacks on memory as the SGX technology is capable of completely subverting physical memory attacks such as cold-boot or bus tapping [15].

Secure execution is achieved by reserving a protected memory region, called the *Processor Reserved Memory*, which stores encrypted pages containing protected code and data. These contents, called the *Enclave Page Cache*, are protected from all accesses which do not originate from within the enclave. All hardware exceptions are handled outside the enclave to avoid data leaks [15]. After all the pages are loaded into the EPC, a remote party can initiate an attestation process to ensure that the enclave was properly initialized. A cryptographic hash of EPC contents is formed and presented as evidence to the attesting party.

Kaplan *et al.* [16] presented AMD's security extensions in 2016., primarily focused on cloud computing. Their main features include *Secure Memory Encryption (SME)* and *Secure Encrypted Virtualization (SEV)*. The former addresses physical attacks and the latter addresses software based attacks. All extensions use a secure coprocessor for encryption key management and a separate AES hardware encryption engine [16] [14].

The *Secure Memory Encryption* extension uses a 128-bit key for encrypting memory contents. An unused bit in the page table entry layout, later referred to as the *C bit*, was used to distinguish between encrypted and plain pages, as well as to provide seamless access to encrypted pages [16]. The SME extension is thus able to offer a flexible encryption scheme,

ranging from full to partial memory encryption. However, this scheme requires support from the underlying operating system as a result of the aforementioned modifications of the page table entry. An additional encryption scheme, *Transparent Secure Memory Encryption*, avoids this problem by transparently encrypting all memory pages at boot [16] [14].

The SEV extension aims to harden virtual machine isolation in multi-tenant cloud systems through an additional, cryptographically aided protection layer. With the threat model assuming a malicious hypervisor, each virtual machine's memory space is encrypted and communication channels used for interacting with the hypervisor are limited, thus further protecting the guest virtual machine's data and code integrity [16], [17]. This extension has recently been expanded with the introduction of *Secure Encrypted Virtualization - Encrypted State (SEV-ES)* [18] and *Secure Encrypted Virtualization - Secure Nested Paging (SEV-SNP)* [17]. These additions enable virtual machine encryption with a guest specific key, cryptographic protection of the guest virtual machine's CPU register state [18], more memory privilege levels inside the virtual machine and enhanced memory page integrity [17].

C. Current research

Confidential computing has received a significant amount of research due to the widespread use of cloud computing environments for workloads with sensitive data. A number of systems demonstrating feasible applications of TEEs in securing traditional workloads have been developed in recent years, ranging from systems which provide a fully shielded execution environment for existing applications [19], to adaptations of specific applications [20].

The work of Baumann *et al.* [19] showed that it is possible to create a fully trusted execution environment inside TEEs. The resulting system, named *Haven*, offers shielded execution of unmodified binary applications using the Intel SGX enclave. It represents a significant change in the use of TEEs, diverging from the common pattern of protecting specific parts of an application. The authors solved a series of technical challenges by emulating traditional OS mechanisms such as system calls, exceptions and memory allocations inside the enclave, using a combination of library operating systems and native implementations of specific mechanisms.

Although this approach poses itself as an elegant and convenient solution for securing existing applications without modifying them, it carries a risk of undermining security guarantees provided by the TEEs. In a recent work on exploiting memory corruption vulnerabilities in TEEs, Lee *et al.* [21] have shown that it is possible to leverage existing vulnerabilities inside an application running in an enclave to completely bypass all security guarantees. This sobering and alarming discovery demonstrates that even with state-of-the-art protection mechanisms in place, a large TCB still carries a security risk. While Lee *et al.* note that their research results strongly suggest that future research should focus on traditional security mitigations [21], determining an appropriate trade-off between security and convenience still remains an open question.

Following a more common TEE usage pattern, Gribov *et al.* [22] designed an encrypted database system with full SQL query support. A small TCB in the form of an extension for an existing DBMS, PostgreSQL, performs queries over encrypted data types. The evaluation results stated in the paper claim an average throughput decrease of 30% [22]. Seeing as the authors published the system's source code, we re-evaluated the performance claims and found the results to be comparable to the ones stated in the paper.

Priebe *et al.* [20] proposed a database engine with even stronger data confidentiality and integrity guarantees. They achieved server-side security by modifying an existing DBMS to create and host sensitive data inside an enclave, using a query engine which executes pre-compiled queries. The client defines a set of queries which will be executed on sensitive data and packages them with query engine. In addition to achieving a significant performance gain, this approach allowed the authors to leverage the remote attestation mechanism of Intel's SGX enclave to guarantee that no undefined queries can be executed on sensitive data. It should be noted that hosting data inside enclaves relies on the assumption of future support for enclaves with large memory spaces [20], which has not come to fruition yet.

V. Conclusion

Ensuring the privacy of confidential client data from a service provider's perspective is not a straightforward process, especially in multi-tenant cloud environments. In addition to traditional risks found in preserving data confidentiality, this multifaceted process must also take potential data breach sanctions and client non-repudiation attempts into account.

Data breach risks originating from third-party actors are usually mitigated by utilizing some form of *encryption at rest*, applied to the underlying filesystem or to specific files, preventing data breaches even if a database containing sensitive data was compromised. However, as noted in our threat model, malicious internal actors pose a significant threat in this setting. Administrative accounts can completely invalidate the confidentiality provided by *encryption at rest*. Therefore, transparent data handling and in-use data protection is paramount for providing privacy guarantees.

Searchable encryption and confidential computing are a natural fit for this use case as they both provide data privacy in a shared computing environment. Despite a significant development seen in searchable encryption techniques, they often impose prohibitive computational and storage overhead and require heavy modifications of underlying database management systems, resulting in a plausible but practically often cumbersome data privacy solution. The *confidential computing* model has recently seen widespread adoption with novel research demonstrating effective application in the field of data privacy.

A major factor affecting the design of a system which protects data in use is the high computational overhead of performing queries on encrypted data. Design choices should thus focus on deciding which actor should bear most of the

inherent computational burden. Client-side manipulation of encrypted data is the most transparent solution, but incurs prohibitively high data transfers, depending on the size of the database. On the other hand, handling encrypted data on the remote server is faster but not sufficiently transparent, thus requiring implementation of additional attestation mechanisms. Moreover, this approach often requires heavy modification of the underlying DBMS, which is not practical. A middleware based approach offers adaptability, leaving room for balancing between the aforementioned approaches.

ACKNOWLEDGMENT

This work has been carried out within the *AIPD2, Digital platform for personal data lifecycle management protection* project, funded by the European Regional Development Fund.

REFERENCES

[1] M. —. S. UK Department for Digital Culture. (2020). "Cyber security breaches survey 2020," [Online]. Available: https://www.gov.uk/government/publications/cyber-security-breaches-survey-2020/cyber-security-breaches-survey-2020. (accessed: 13.11.2020.)

[2] I. Goddijn, "Midyear quickview data breach report," *RiskBased Security*, 2019.

[3] IBM. (2020). "Cost of a data breach report 2020," [Online]. Available: https://www.ibm.com/security/data-breach. (accessed: 16.11.2020.)

[4] V. Enterprise, "2020 data breach investigations report," 2020. [Online]. Available: https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf (visited on 11/15/2020).

[5] OWASP. (2020). "Threat modeling cheat sheet," [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html. (accessed: 05.11.2020.)

[6] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack. (2006). "Threat modeling: Uncover security design flaws using the stride approach," [Online]. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach. (accessed: 06.11.2020.)

[7] OWASP. (2013). "Application threat modeling via the pasta methodology," [Online]. Available: https://owasp.org/www-pdf-archive//APAC13_TonyUV.pdf. (accessed: 08.01.2021.)

[8] V. Jagannathan. (2020). "Threat modeling: Architecting —& designing with security in mind," [Online]. Available: https://owasp.org/www-pdf-archive/AdvancedThreatModeling.pdf. (accessed: 05.11.2020.)

[9] M. Bartock, M. Souppaya, R. Savino, T. Knoll, U. Shetty, M. Cherfaoui, R. Yeluri, and K. Scarfone, "Hardware-Enabled Security for Server Platforms: Enabling a Layered Approach to Platform Security for Cloud and Edge Computing Use Cases (Draft)," National Institute of Standards and Technology, Tech. Rep., 2020. [Online]. Available: https://doi.org/10.6028/NIST.CSWP.04282020-draft.

[10] P. Maene, J. Götzfried, R. De Clercq, T. Müller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 361–374, 2017.

[11] M. Russinovich, "Introducing Azure confidential computing," *Microsoft Azure Blog*, 2017. [Online]. Available: https://azure.microsoft.com/en-gb/blog/introducing-azure-confidential-computing/ (visited on 11/15/2020).

[12] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, IEEE, vol. 1, 2015, pp. 57–64.

[13] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution.," *Hasp@ isca*, vol. 10, no. 1, 2013.

[14] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of Intel SGX and AMD memory encryption technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2018, pp. 1–8.

[15] V. Costan and S. Devadas, "Intel SGX Explained.," *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[16] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption," *White paper*, 2016. [Online]. Available: https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf.

[17] AMD, "Strengthening VM isolation with integrity protection and more," *White Paper, January*, 2020. [Online]. Available: https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[18] D. Kaplan, "Protecting VM register state with SEV-ES," *White paper, Feb*, 2017.

[19] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with Haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, pp. 1–26, 2015.

[20] C. Priebe, K. Vaswani, and M. Costa, "Enclavedb: A secure database using SGX," in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 264–278.

[21] 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," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 523–539.

[22] A. Gribov, D. Vinayagamurthy, and S. Gorbunov, "Stealthdb: A scalable encrypted database with full sql query support," *arXiv preprint arXiv:1711.02279*, 2017.