International Journal on Cryptography and Information Security (IJCIS), Vol. 10, No.3, September 2020

# Threat Modeling of Cloud based Implementation of Homomorphic Encryption

Satish K Sreenivasaiah & Soumya Maity

REVA University, Bangalore KA 560064, India,
satish.cs01@reva.edu.in
soumya.maity@race.reva.edu.in

September 29, 2020

**Abstract**

Outsourcing of data storage and data processing to cloud-based service providers promises several advantages such as reduced maintenance overhead, elastic performance, high availability, and *security*. Cloud services offer a variety of functionalities for performing different operations on the data. However, during the processing of data in cloud, *security and privacy* may be compromised because of inadequate cryptographic implementation. Conventional encryption methods guarantee security during transport (data-in-transit) and storage (data-at-rest), but cannot prevent data leak during an operation on the data (data-in-use). Modern homomorphic encryption methods promise to solve this problem by applying different operations on encrypted data without knowing or deciphering the data. Cloud-based implementation of homomorphic cryptography has seen significant development in the recent past. However, data security, even with implemented homomorphic cryptography, is still dependant on the users and the application owners. This exposes the risk of introducing new attack surfaces. In this paper, we introduce a novel and one of the early attempts to model such new attack surfaces on the implementation of homomorphic encryption and map them to STRIDE threat model [1] which is proliferously used in the industry.

## 1 Introduction

Modern cloud services enable efficient computations on various data sets in the form of Platform or Software-as-a-Service. Data processing and Data analysis become easy and reliable due to elastic high-performance hardware used by cloud service providers (CSP). Recent data trends suggest [2] that there is an exponential increase in the growth rate of data creation. Often, this data is

DOI:10.5121/ijcis.2020.10302

shared with multiple parties, such as a CSP or a third-party organization, to store and process.

Alarmingly, users do not have control over their data and are naturally concerned about data privacy. Furthermore, data is often exposed to breaches, where sensitive customer information is accessed in an unauthorized manner. Customers often risk the privacy of their data in exchange for services from CSPs. Although these CSPs are considered as trusted business partners and are deterrent from stealing data by service level agreements, non-disclosure agreements, etc., there is a need for proven technology to prevent data to be disclosed to the cloud operators.

The CSPs can be restricted to access the user data using symmetric key cryptography like AES or 3-DES, while the data is stored (or, *data-at-rest*) in a secure datastore in the cloud [3]. The *data-in-transit* is secured during communication using public-key cryptography [4]. However, there is no well-known technology that can prevent data theft when under process (*data-in-use*). While users can encrypt data and store it on the cloud for confidentiality, this limits any kind of data processing. Therefore, the usual encryption is limited to data storage alone and does not allow for any meaningful computation. While doing different operations on the data, the cloud service provider can access the data and can technically store, share, or replay it. This security issue is very much predominant in the case of public clouds that are owned and operated by a third party.

To enable computations while guaranteeing data privacy, researchers are focusing on *privacy-enabled computations* or *confidential computations*. *Homomorphic encryption* (HE) is a promising solution towards that technical problem without compromising the robustness, scalability, and security [5][6]. As the name suggests, this is a special type of cryptosystem that has homomorphic property[7]. That means, it allows calculations to be performed on the encrypted data itself, thus the data is never decrypted even while in use.

Encrypted data is stored in a cloud. There they can be searched or processed without decrypting them. The result is sent back encrypted. The cloud provider does not know the data or the results. Though HE promises a big gain in data protection, efficiency and performance are still a major concern, at least for the early cloud-based implementations of homomorphic encryption (HE) [8] .

Homomorphic encryption, in a simple language, is a normal encryption scheme (two functions *enc* and *dec* to encrypt and decrypt) with one additional function, *eval*, such that, $eval(enc(m)) = enc(f(m))$, and, $dec(eval(enc(m))) = f(m)$

where, $m$ is a plaintext data, and user wants to compute $f(m)$.

Diagram 1 explains the operation.

**Example:** A Search engine is a commonly used Software-as-a-service. When a user types a string in the text-box, the service provider finds relevant webpages from a highly dynamic data-store and responds with that list. The request and response are encrypted using RSA public-key cryptography. So, it is secured from any eavesdropper. However, the service provider can see the *search string* in plaintext. Homomorphic encryption ensures that the service provider receives

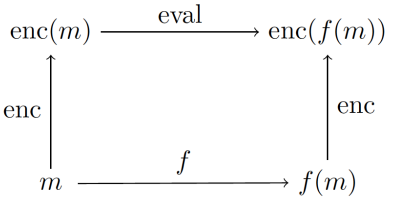


Figure 1: Homomorphic Encryption

an encrypted string, finds an encrypted list of relevant pages without even knowing the actual search string, and response. User can decrypt the response and see the list. The service provider can never know the actual search-string and the responses, but it still can provide the service seamlessly. Detail explanation of homomorphic encryption (HE) is discussed in the subsequent sections.

Microsoft's *Azure Confidential Computing* introduced in 2019 [9], followed by Google's *confidential cloud* [10], launched as a beta release in July, 2020, are the first commercial implementations of homomorphic encryption. Other cloud providers including are also extensively researching the technology to make their cloud resilient to data-breaches.

HE has been called the "Swiss Army knife of cryptography" as it is a one-stop shop solution that can be applied consistently across variety of cryptographic implementations. It is often believed to be a silver bullet for most of the problems plaguing the industry today, in terms of protection of Sensitive Personal Data or Information (SPDI) from third-party cloud providers [11]. However, challenges persist wherein the insecure implementation and inadequate security controls around HE could compromise the data and negate the whole purpose of using HE as a solution to protect SPDI from cloud vendors or third party data processors. HE, as a cryptosystem, is resilient to data breaches and attacks on privacy. But, the success of protecting the confidentiality, integrity, and availability (CIA) depends largely on the implementation and design of the system. Threat modeling is a well-accepted formal approach to find relevant threats or attack surfaces of the designed system. To identify these potential threats and possible attacks early in the life cycle of software product development, we could employ STRIDE based threat modeling [1] as an effective tool during the product design phase. Although thorough cryptanalysis would uncover these attacks or threats, it is a long drawn process and requires a high level of expertise. Hence, as a quick alternative for a rigorous cryptanalysis approach, a threat modeling methodology and tools can be adopted to identify threats and address them through appropriate mitigation techniques for a secure HE implementation.

**Research Objective**

As mentioned in the abstract, cloud-based implementation of homomorphic cryptography has seen significant development in the recent past. However, data security, even with implemented homomorphic cryptography, is still dependent on the users and the application owners. This exposes the risk of introducing new attack surfaces. In this paper, we introduce a novel and one of the early attempts to model such new attack surfaces on the implementation of homomorphic encryption and map them to Microsoft STRIDE threat model [1] which is proliferously used in the industry

**Scope**

The scope of the research is to identify threats using Microsoft STRIDE model in a cloud based homomorphic encryption implementation early in the product design phase and to plan the adoption of mitigations stated in the paper for the identified threats.

**Limitations:**

The limitation of the paper is that it does not delve deep into cryptanalysis although that is one of the right approaches to find weaknesses in cryptographic algorithms. As cryptanalysis is a time consuming activity, Threat Modeling of a HE system is suggested as a quicker alternative to identify threats and mitigations.

**Organization of the Paper**

In this paper, we model threats for Cloud-based implementation of HE using STRIDE. As per our best knowledge and literature survey, this is one of the earliest attempts for mapping attack surfaces of HE implementation with STRIDE.

We have organized this paper into three major sections. In the beginning, in section 2, we introduce the background of HE along with detail cryptanalysis. After that, we explain different attack surfaces on the implementation of cloud-based HE in section 3. How we can map the attack surfaces with the STRIDE model is explained in section 4. We conclude the paper by pointing on the merit and future scope of this work.

# 2 Background

HE is very different from other forms of cryptographic algorithms such as regular symmetric and asymmetric algorithms in a way that it can do computing on encrypted data and provide result of the computation as an encrypted output. This capability of HE is a game changer as it can now preserve the privacy or confidential data of an individual or corporates by not using plaintext data for processing.

## 2.1 Homomorphic encryption

CPA security does not prevent an attacker from tampering with the encrypted message, changing for example an encryption of the message $x$ into an encryption of $x$ with its last bit flipped. Homomorphic encryption takes this to an extreme and actually *requires* that it is possible to tamper with the encryption in an arbitrary way (while still maintaining CPA security!). The question if this is possible was first raised in 1978 by Rivest, Adleman, and Dertouzos, and over the years many conjectured that this is in fact impossible. Last year Gentry gave very strong evidence that such encryptions exist, by constructing such a scheme that is secure under relatively reasonable computational assumptions.
[**Definition**] A CPA-secure public key encryption scheme $(G, E, D)$ with one bit messages is *fully homomorphic* if,
there exists an algorithm $HEnc$ such that for every $(e, d) \leftarrow G(1^n)$, $a, b \in \{0, 1\}$, and $\hat{a} \leftarrow E_e(a)$, $\hat{b} \leftarrow E_e(b)$,

$$HEnc_e(\hat{a}, \hat{b}) \approx E_e(aHEncb)$$

where $\approx$ denotes statistical indistinguishability (i.e., $n^{-\omega(1)}$ statistical distance), and $aHEncb$ denotes $\neg(a \wedge b)$.

We stress that the algorithm HEnc does *not* get the secret key as input. Otherwise it would be trivial: just decrypt $\hat{a}, \hat{b}$, compute $aHEncb$ and re-encrypt.
[**Universality of HEnc**] It's straightforward to show that every log gate can be expressed using few HEncs, and so obtain the following claim (left as exercise): If $(G, E, D)$ is a homomorphic encryption then there is an algorithm $EVAL$ that for every $(e, d) \leftarrow G(1^n)$, $x_1, \ldots, x_m \in \{0, 1\}$, if $\hat{x}_i = E_e(x_i)$ and $C$ is a Boolean circuit mapping $\{0, 1\}^m$ to $\{0, 1\}$, then

$$EVAL_e(C, \hat{x}_1, \ldots, \hat{x}_m) \approx_{|C|\mu(n)} E_e(C(x_1, \ldots, x_n))$$

where we say that $D \approx_\epsilon D'$ if their statistical distance is at most $\epsilon$, $\mu$ is some negligible function, and $|C|$ denotes the number of gates of $C$. In particular if $C$ is polynomial size then these two distributions are statistically indistinguishable.

## 2.2 Usefulness of homomorphic encryption

Canonical application is "cloud computing": Alice wants to store her file $x \in \{0, 1\}^m$ on Bob's server. So she sends Bob $E_e(x_1) \cdots E_e(x_m)$. Then she wants to do computation on this file. For example, if the file is a database of people she may want to find out how many of them bought something in the last month. One way to do so would be for Alice to retrieve the entire file and do the computation on her own, but if she was able to handle this amount of communication and computation, perhaps she wouldn't have needed to use cloud computing in the first place.

Instead, Alice will ask Bob to perform this operation on the encrypted data, giving her an encryption of the answer, which she can of course decrypt. There is an issue of how Alice maintains integrity in this case, this is left as an exercise.

## 2.3 Zero Knowledge from Homomorphic Encryption

We've seen zero knowledge protocols for specific statements, but now we'll see such an encryption scheme for *any* statement, specifically for a public input circuit $C : \{0,1\}^m \to \{0,1\}$, we'll show a zero knowledge proof system (in fact even proof of knowledge) for the statement "there exists $x$ such that $C(x) = 1$".

Note that this in some sense a tremendous overkill, since zero knowledge proofs for every statement can be based on just one-way functions, and the construction is not even terribly complicated, given basic NP-completeness results. But this protocol will give some intuition on homomorphic encryption, and will also be more communication efficient than the standard protocols.

### cryptanalysis

We'll describe the protocol in steps, starting with a simplified version that is not secure and tweaking it as we go along to ensure security.

**Public Input:** Boolean circuit $C : \{0,1\}^n \to \{0,1\}$.

**Prover's private input:** $x \in \{0,1\}^n$ such that $C(x) = 1$.

**Step 1** Prover runs $(e, d) \leftarrow G(1^n)$, sends $e$ to verifier.

**Step 2** Prover sends $\hat{x} = E_e(x_1) \cdots E_e(x_n)$ to verifier.

**Step 3** Verifier computes $\hat{c} = EVAL(C, \hat{x})$, sends $\hat{c}$ to prover.

**Step 4** Prover sends $d = D_d(\hat{c})$ to verifier. Verifier accepts if $d = 1$.

### Security

This protocol is obviously not sound. We change it by having the verifier toss a coin $b \leftarrow_R \{0,1\}$ in Step 3. If $b = 1$ then the verifier proceeds as before. If $b = 0$ then the verifier sends $E_e(b)$ to the prover. The verifier checks in Step 4 that $b = d$.

### Soundness

We can now prove soundness of the new protocol though we will need a strengthening of the homomorphic encryption scheme, we require that it is possible to efficiently test that a public key $e$ is in the range of the generation algorithm and a ciphertext $\hat{a}$ is in the range of the encryption algorithm. This can be fixed by adding another check by the verifier, though we'll defer details to the exercise.

**Step 4** The prover only sends a *commitment* to $d$ (for example $f(x), r, \langle x, r\rangle \oplus d$, where $f$ is a one-way permutation).

**Step 5** Verifier sends all randomness it used in producing the ciphertext of Step 3. The prover verifies this is indeed the case, and otherwise aborts.

**Step 6** The prover sends $d$ and also the randomness used in producing the commitment.

This can be shown to preserve soundness, since soundness held even for *computationally unbounded* provers, and the commitment scheme is *perfectly binding*.

**Constructing homomorphic encryption**

Homomorphic encryptions can be used to do wonderful things, but the same holds for perpetual motion machines, cold fusion, unicorns, etc.

So, the question whether we can actually construct such schemes. Since the question was raised in 1978 by Rivest et al. [12], there have been no significant candidate for a homomorphic encryption scheme. Almost after 30 years of that work, Gentry gave the first such construction [5][13]. The construction relies on somewhat non-standard, but still rather reasonable assumptions. Also, as mentioned, it is still not practical, requiring at least $k^8$ operation to achieve $2^k$ security. Hopefully, with time we will see improved constructions, using more standard assumptions and more efficient. We will see a close variant of Gentry's scheme now. We remark that all the applications we saw (zero knowledge, multi-party computation, private information retrieval) have alternative constructions that utilize much more standard assumptions.

## 2.4 Need of HE

In the new age of regulatory compliances and the paramount importance placed on privacy of individuals across and within nations, it is an imperative need to protect SPDI from everyone except the data owner. The only person who needs to have access to the data should be the data owner and not even to the data processor or any third party processing or administering unit or individuals. This is a paradox as protecting SPDI from a data processor, be it cloud or third party on premise vendor, is hard as data needs to be decrypted prior to processing and decrypted data in the memory is accessible for the cloud provider, if he wishes to see. Hence, HE is a perfect solution to address the mentioned paradox and a timely technological intervention without which the only way to address the case was through legal and contractual obligations between the data owner and the data processor.

## 2.5 Types of HE

Now having seen the what and why of HE, we explain different types of HE in vogue today. The categories of HE are based on the number of mathematical computations that can be performed on the encrypted text. The major differences in terms of capability, is tabulated in table 1.

### 2.5.1 Partially Homomorphic Encryption (PHE):

It supports computations of mathematical operations such as addition or multiplication on the encrypted data for unlimited number of times. It allows only one type of operation to be performed. PHE allows any number of computations to be performed for only a single type of mathematical operation, be it addition or multiplication on encrypted data. Ex., it only allows computation of either additions, such as $enc(x + y)$ for a given $enc(x)$ and $enc(y)$. Similarly computation for multiplication operation alone such as $enc(x * y)$ for a given $enc(x)$ and $enc(y)$

### 2.5.2 Somewhat Homomorphic Encryption (SHE):

SHE allows a limited number of computations to be performed for both types of mathematical operations, addition and multiplication on encrypted data. Ex., it allows computation of additions, such as $enc(x + y)$ and computation of multiplications operation $enc(x * y)$, for a given $enc(x)$ and $enc(y)$.

### 2.5.3 Fully Homomorphic Encryption (FHE):

FHE allows any number of computations to be performed for both types of mathematical operations, addition and multiplication on encrypted data. FHE allows unlimited additions and multiplications.

Partially homomorphic encryption is fairly easy; eg. RSA has a multiplicative homomorphism: $encrypt(x) = X^e, encrypt(y) = y^e$,
So, $encrypt(x) * encrypt(y) = (xy)^e = encrypt(xy)$

Elliptic curves can offer similar properties with addition. Allowing both addition and multiplication is, it turns out, significantly harder.

Table 1: Categorization summary of Homomorphic Encryption

| HE Types/Operations | PHE | SHE | FHE |
|---|---|---|---|
| Operations Supported | Addition OR Multiplication | Addition AND Multiplication | Addition AND Multiplication |
| Frequency of operations | Unlimited | Limited | Unlimited |

# 3 Attacks on a HE implementation

After introducing the conceptual background of HE we introduce different types of threats or attacks that could be possible on Cloud based or non-cloud based HE implementations [14].

## 3.1 Chosen Cipher-Text Attack

In this kind of attack, the attacker might get decryptions of chosen cipher texts with restrictions [15]. CCA has all the capabilities of a CPA as detailed in the above section and also obtain decryptions of the selected encrypted message.

## 3.2 Encoder Attack

There are different Homomorphic Encryption libraries available today such as Microsoft SEAL [16] (Simple Encrypted Arithmetic Library), Palisade, HELib, NFLlib and so on. A few of the encoding methods used in SEAL to convert integers and floating point numbers are IntegerEncoder () and Batch Encoder () [17]. IntegerEncoder method is known to leak information and is suggested not to be used in real applications whereas BatchEncoder () does not seem to have this vulnerability.

## 3.3 Side Channel Attack

In this kind of attack, a malicious user does a run time monitoring of encryption operation to obtain environment details like power consumption for the encryption process and alternatively the elapsed time for key generation. It provides sufficient details about the key leading to potential attacks.

## 3.4 Active Attacks

Active attacks are wherein the adversary plants himself in between the data sender and the receiver and starts monitoring the data in transit, modifies the data in motion or at the target environment and also can inflict non availability for the target server. Active attacks include DoS, DDoS, Session Hijacking and so on.

## 3.5 Key Recovery Attack

In this kind of attack, given multiple plaintext/cipher text pairs an attacker can obtain the secret key from the provided pairs of plaintext/cipher text.

## 3.6 Network Traffic Interception

Attackers can intrude the network if not secured with the right security controls for data in transit. There are multiple OSS and COTS tools that can be used for network sniffing and interception that can impact confidentiality and potential integrity of the data.

# 4 Mapping of Attacks to STRIDE Threat Model

## 4.1 STRIDE Threat Model

Microsoft STRIDE methodology [1] stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privileges. There are different kinds of threat modeling approaches such as Attacker centric, Asset centric and Software or System centric. STRIDE is a system centric approach where it focuses on the threats that could potentially impact a software system.

STRIDE, should be applied during the software design phase of the SDLC life cycle in order to ensure the identified threats are mitigated upfront during the product development life cycle.

The table 2 provides definition for each of the STRIDE threat methodology along with examples. The last column in the table maps to the impacted Security Quality Sub Characteristics (QSC) of CIA triad, Authentication, Authorization and Non-repudiation.

Now having understood the STRIDE methodology and the quality sub characteristics it is impacting in a software system, let's map the potential attacks to threats and subsequently detail the mitigation for each of these threats or potential attacks.

## 4.2 Mapping of potential HE implementation attacks to STRIDE

As detailed in the table 3, a CCA in HE implementation, can potentially lead to Information Disclosure and Tampering threats. With this, the idea of computing on cipher text in HE that is assumed to be protected from a cloud provider takes a beating as the adversary has obtained knowledge of some part of plain-text based on CCA.

The countermeasure or mitigation for this attack is to adopt Authenticated Encryption (AE) as it withstands the CCA attacks. AE provides Confidentiality and Authentication at the same time as against a plain encryption implementation. The product team needs to ensure that they implement AE-Secure solution during the design phase to avoid threats from CCA attacks.

Encoder Vulnerability in HE library - Integer, Floating number encoding vulnerabilities in well-known HE libraries as discussed earlier, leads to Information Disclosure. The mitigation for these kind of threats is to use Safe libraries during coding such as BatchEncoder () method as compared to IntegerEncoder () method. As shown in the table 3, it impacts Confidentiality of the data and hence needs to be addressed early in the SDLC phase.

Side Channel Attack – could potentially obtain sensitive information through various parameters discussed in section 2 impacting Confidentiality of the system. The mitigation for such attacks are varied and techniques comprise of jamming the emission channel, inducing random delay in the timing, randomization of cipher text and so on.

Table 2: STRIDE Methodology

| Threat | Definition | Examples | Security Quality Sub-Characteristic |
|---|---|---|---|
| Spoofing | Claiming to be someone else or somebody else's identity | To take an identity of the sender of a packet and claim to be the packet sender to the receiver | Authentication |
| Tampering | Change the data or code in transit, while at use or at rest | Modify data that is stored in a database or a file while data is at rest. And while data is in transit, modify the contents on the wire | Integrity |
| Repudiation | Deny performing an action or a transaction | "I never did that transaction" or "I never updated that database or file" | Non Repudiation |
| Information | Disclosure Unauthorized disclosure of data | Leakage of sensitive data like SPDI or business confidential data due to vulnerabilities in the system or unauthorized access | Confidentiality |
| Denial of Service | Make system unavailable for business or make it slow to impact the users | Bombard the server with huge HTTP/TCP or any other requests so that server shuts down | Availability |
| Elevation of Privilege | Unauthorized privileges | A database user having privileges of a database administrator | Authorization |

Active Attacks – The potential attacks of DoS, DDoS, Session Hijacking could lead to all the threats of STRIDE as shown in the table 3. And mitigations for active attacks need to be handled at infra, code and configuration levels.

Key recovery attack – As in table 3, this could happen due to availability of plain text/cipher text cipher and is a threat to Confidentiality. The mitigation of this kind of a threat is to ensure strong encryption algorithms such as AES 256. AES 128 and 256 both are known to withstand key recovery attacks

Network Sniffing Attack – This is a potential threat to Confidentiality, Integrity and Availability of a system. Mitigation is to ensure data in transit is secure with strong TLS encryption and also ensure Mutual TLS is enabled for intra server communication.

So, with the proposed mapping of potential attacks, threats and security quality sub characteristics along with mitigation following a STRIDE methodology is a sure way of ensuring security of HE implementation on Cloud.

Table 3: Categorization summary of Homomorphic Encryption

| Attacks | S | T | R | I | D | E | Impacted Security QSC |
|---|---|---|---|---|---|---|---|
| Chosen Ciphertext Attack | | ✓ | | ✓ | | | Confidentiality Integrity |
| Encoder Attack | | | | ✓ | | | Confidentiality |
| Side Channel Attack | | | | ✓ | | | Confidentiality |
| Active Attacks | ✓ | ✓ | | ✓ | ✓ | ✓ | CIA, Authorization Authentication |
| Key Recovery Attack | | | | ✓ | | | Confidentiality |
| Network Sniffing Attack | | ✓ | | ✓ | ✓ | | CIA |

# 5 Conclusion

In this paper, we introduced a novel approach for doing threat modeling for cloud-based implementation of homomorphic encryption. We translate the cryptanalysis based attack models into STRIDE threat model which is easy to understand by the practitioners. The present state of art in the domain is still very immature. We are in process of building a framework based on this approach, which is included as our future scope of work. This work should help the developers ar architects to build in-depth security implementation of homomorphic security in an effective and efficient way and without understanding the detail mathematical background of this cryptanalysis.

# References

[1] A. Shostack, "Experiences threat modeling at microsoft." *MODSEC MoDELS*, vol. 2008, 2008.

[2] S. He, G. Manns, J. Saunders, W. Wang, L. Pollock, and M. L. Soffa, "A statistics-based performance testing methodology for cloud applications," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 188–199.

[3] C. Wang, K. Ren, W. Lou, and J. Li, "Toward publicly auditable secure cloud data storage services," *IEEE network*, vol. 24, no. 4, pp. 19–24, 2010.

[4] M. E. Hellman, "An overview of public key cryptography," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 42–49, 2002.

[5] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Sanford Univeristy, 2009.

[6] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, 2011, pp. 113–124.

[7] B. Rossman, "Homomorphism preservation theorems," *Journal of the ACM (JACM)*, vol. 55, no. 3, pp. 1–53, 2008.

[8] B. Barak, "Computer science 433 - cryptography, spring 2010," Accessed on 10-August-2020. [Online]. Available: https://www.cs.princeton.edu/courses/archive/spring10/cos433/

[9] Microsoft, "Confidential computing on azure," Accessed on 10-August-2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/confidential-computing/overview

[10] Google, "Google confidential computing," Accessed on 10-August-2020. [Online]. Available: https://cloud.google.com/confidential-computing

[11] I. Žliobaitė and B. Custers, "Using sensitive personal data may be necessary for avoiding discrimination in data-driven decision models," *Artificial Intelligence and Law*, vol. 24, no. 2, pp. 183–201, 2016.

[12] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[13] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[14] M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison *et al.*, "Security of homomorphic encryption," *HomomorphicEncryption. org, Redmond WA, Tech. Rep*, 2017.

[15] D. Boneh and V. Shoup, "A graduate course in applied cryptography," *Draft V0.5*, 2020.

[16] Microsoft, "Microsoft seal," Accessed on 10-August-2020. [Online]. Available: https://github.com/Microsoft/SEAL

[17] Z. Peng, "Danger of using fully homomorphic encryption: A look at microsoft seal," *arXiv preprint arXiv:1906.07127*, 2019.