International Journal on Optimization and Applications

IJOA. Vol. 4, Issue No. 3, Year 2024, www.ijoa.ma

Homomorphic Encryption Schemes using AES: Techniques and Applications

1st Hind BOUHEDDA
Master SSI – ENSA Kenitra, Morocco
hind.bouhedda@uit.ac.ma

2nd Salma DOUKKAR
Master SSI – ENSA Kenitra, Morocco
salma.doukkar@uit.ac.ma

3rd Otmane AMRAOUI
Master SSI – ENSA Kenitra, Morocco
otmane.amraoui@uit.ac.ma

4th Achraf AZAHOUM
Master SSI – ENSA Kenitra, Morocco
achraf.azahoum@uit.ac.ma

5th Mouad JAOUANI
Master SSI – ENSA Kenitra, Morocco
mouad.jaouani@uit.ac.ma

***Abstract*- Homomorphic encryption schemes provide a powerful mechanism for performing computations on encrypted data without decrypting it. This capability holds significant promise for enhancing the security and privacy of sensitive information in various applications. In this paper, we focus on exploring homomorphic encryption schemes using the Advanced Encryption Standard (AES). We review the fundamental principles of homomorphic encryption and discuss the potential advantages and challenges of using AES as the underlying cryptographic primitive. Furthermore, we survey recent advancements in the field and highlight key research directions for future exploration. Our analysis aims to provide researchers and practitioners with insights into the state-of-the-art techniques and opportunities for leveraging homomorphic encryption with AES in real-world applications.**

Keywords- AES, Encryption, Homomorphic

I. INTRODUCTION

In order to enable safe computation on encrypted data and protect the confidentiality and integrity of sensitive information in a variety of situations, homomorphic encryption has become a key technology. With homomorphic encryption, computations can be done directly on encrypted data, producing encrypted results that can be decrypted to produce the same result as if the computations were done on plaintext data. This is in contrast to traditional encryption schemes, which make data unreadable to unauthorized parties. This capability creates new opportunities for secure computation outsourcing, cooperative data sharing across trust boundaries, and privacy-preserving data analysis. Finding a balance between security, efficiency, and functionality is one of the main issues in the design of homomorphic encryption schemes.

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm that has gained widespread adoption due to its robust security features and seamless integration on contemporary computing platforms.

High computing performance and strong security guarantees can both be obtained by utilizing AES in homomorphic encryption schemes. However, careful consideration of AES's cryptographic properties and the creation of appropriate algebraic structures are needed to adapt it to support homomorphic operations. Homomorphic encryption (HE) [1] is a kind of public key encryption that allows computation over encrypted data with- out knowing the secret key, and has several applications such as delegated computation on cloud servers.

In this paper, we present an exploration of the combination of homomorphic encryption with AES (Advanced Encryption Standard) techniques, highlighting its significance in preserving privacy and security in data processing.

The background section provides an explanation of homomorphic encryption principles, including its different types such as partially homomorphic, somewhat homomorphic, and fully homomorphic encryption. We also provide an overview of the AES encryption algorithm, including its block cipher structure, key sizes, and cryptographic properties. Furthermore, we review previous research on homomorphic encryption schemes and their various use cases.

Moving on to the fundamentals, we delve into how homomorphic encryption principles can be applied to AES encryption. We discuss the challenges and considerations involved in adapting AES for homomorphic operations. Additionally, we provide an overview of existing techniques and approaches for combining homomorphic encryption with AES.

P. Paillier, "Public-key cryptosystems based on composite degreeresiduosity classes," EUROCRYPT 1999, LNCS, vol.1592, pp.223–238, 1999.

The subsequent section explores specific techniques and methodologies for achieving homomorphic properties with AES in detail. We discuss encryption schemes, such as partially homomorphic or fully homomorphic encryption, that utilize AES as the underlying cryptographic primitive. We also evaluate the security, efficiency, and performance characteristics of different AES-based homomorphic encryption techniques.

In the applications section, we survey real-world applications and use cases where homomorphic encryption schemes using AES can be applied. We provide examples of scenarios in data privacy, secure computation, cloud computing, and other domains that benefit from the combination of homomorphic encryption with AES. Additionally, we showcase case studies or practical implementations that demonstrate the effectiveness and feasibility of AES-based homomorphic encryption in various applications.

The article then addresses the challenges and future directions in the field of AES-based homomorphic encryption, highlighting areas for further research and development.

Finally, we conclude by summarizing the key findings and insights from the article, emphasizing the significance of combining homomorphic encryption with AES in enhancing privacy and security in data processing.

II. BACKGROUND

A. *Homomorphic Encryption Definition:*

Homomorphic comes from the Greek words for ‘same structure’. It means that I can perform operations on things, and the structure is preserved after a mapping.

The concept of homomorphic encryption was introduced in [1], of which two of the authors are Ronald L. Rivest and Len Alderman. The R and the A in RSA encryption.

The most popular example for the use of homomorphic encryption is where a data owner wants to send data up to the cloud for processing, but does not trust a service provider with their data. Using a homomorphic encryption scheme, the data owner encrypts their data and sends it to the server. The server performs the relevant computations on the data without ever decrypting it and sends the encrypted results to the data owner. The data owner is the only one able to decrypt the results, since they alone have the secret key.

B. *Homomorphic Encryption Types :*

- **Partially Homomorphic Encryption (PHE):** In PHE schemes, only one type of mathematical operation (either addition or multiplication) can be performed on encrypted data while preserving the homomorphic property. For example, the RSA cryptosystem is partially homomorphic with respect to multiplication.
- **Somewhat Homomorphic Encryption (SHE):** SHE schemes allow a limited number of both addition and multiplication operations to be performed on encrypted data while maintaining the homomorphic property. Examples include the Gentry-Halevi Smart (GHS) scheme and the Brakerski-Gentry-Vaikuntanathan (BGV) scheme.
- **Fully Homomorphic Encryption (FHE):** FHE schemes support an unlimited number of both addition and multiplication operations on encrypted data. In addition to addition and multiplication, fully homomorphic encryption schemes can be used to perform a wide range of operations, including subtraction, division, comparison, boolean operations (AND, OR, NOT), and more. This makes FHE schemes Turing complete, meaning that any computable function can be evaluated on encrypted data.

C. *Overview of AES:*

The DES key length was a mere 56 bits. And it turned out that this isn’t nearly enough to keep encrypted information safe. For example, a test by distributed.net and the Electronic Frontier Foundation showed that DES can be easily cracked in a little bit more than 22 hours. Keep in mind that this was done in 1999, when computing power was far from what it is now.
Today, a powerful machine can crack a 56-bit DES key in 362 seconds.
On the other hand, cracking a 128-bit AES encryption key can take up to 36 quadrillion years.

AES is a symmetric encryption algorithm and a block cipher. The former means that it uses the same key to encrypt and decrypt data. The sender and the receiver must both know -- and use -- the same secret encryption key. This makes AES different from asymmetric algorithms, where different keys are used for data encryption and decryption. Block cipher means that AES splits a message into smaller blocks and encrypts those blocks to convert the plaintext message to an unintelligible form called ciphertext.

AES uses multiple cryptographic keys, each of which undergoes multiple rounds of encryption to better protect the data and ensure its confidentiality and integrity. All key lengths can be used to protect Confidential and Secret level information. In general, AES-128 provides adequate security and protection from brute-force attacks for most consumer applications. Information that's classified as Top Secret -- e.g., government or military information -- requires the stronger security provided by either 192- or 256-bit key lengths, which also require more processing power and can take longer to execute.

How does AES encryption work?

To understand the way AES works, you first need to learn how it transmits information between multiple steps. Since a single block is 16 bytes, a 4x4 matrix holds the data in a single block, with each cell holding a single byte of information.

The matrix shown in the image is known as a state array. Similarly, the key being used initially is expanded into (n+1)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

keys, with n being the number of rounds to be followed in the encryption process. So for a 128-bit key, the number of rounds is 16, with no. of keys to be generated being 10+1, which is a total of 11 keys.

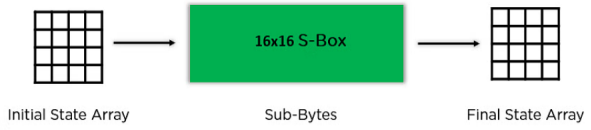
Add Round Key: You pass the block data stored in the state array through an XOR function with the first key generated (K0). It passes the resultant state array on as input to the next step.

1	2	3	4
6	7	8	5
11	12	9	10
16	13	14	15

XOR

K_0	K_1	K_2	K_3
K_4	K_5	K_6	K_7
K_8	K_9	K_{10}	K_{11}
K_{12}	K_{13}	K_{14}	K_{15}

Sub-Bytes: In this step, it converts each byte of the state array into hexadecimal, divided into two equal parts. These parts are the rows and columns, mapped with a substitution box (S-Box) to generate new values for the final state array.



Shift Rows: It swaps the row elements among each other. It skips the first row. It shifts the elements in the second row, one position to the left. It also shifts the elements from the third row two consecutive positions to the left, and it shifts the last row three positions to th

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

→

1	2	3	4
6	7	8	5
11	12	9	10
16	13	14	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

X

C_0
C_1
C_2
C_3

=

NC_0
NC_1
NC_2
NC_3

Constant Matrix Old Column New Column

Mix Columns: It multiplies a constant matrix with each column in the state array to get a new column for the subsequent state array. Once all the columns are multiplied with the same constant matrix, you get your state array for the next step. This particular step is not to be done in the last rounde left.

III. PRELIMINARY

- ***Basic Definitions and Properties:***

Plaintext: Plaintext refers to the original, readable, and unencrypted data or message that is to be encrypted.

Ciphertext: Ciphertext is the encrypted form of plaintext, resulting from the application of an encryption algorithm and a secret key. It appears as unintelligible gibberish and requires the appropriate decryption key to revert it back to plaintext.

Stream cipher: A stream cipher is a symmetric encryption method where plaintext is combined with a pseudorandom keystream, typically generated from a seed value, to produce ciphertext. It encrypts data bit by bit, offering high-speed processing and lower hardware complexity compared to block ciphers, but may be vulnerable to attacks if the same seed is reused.

Block cipher: A block cipher is a symmetric encryption algorithm that operates on fixed-size blocks of data, transforming each block into ciphertext independently. It uses a cryptographic key to perform the encryption and decryption processes.

Keywords:

Gen: Generates public and secret keys based on a security parameter λ.

Enc: Encrypts a plaintext M using a public key pk, producing a ciphertext C.

Dec: Decrypts a ciphertext C using a secret key sk, resulting in either the original plaintext M or a failure symbol ⊥.

Eval: Evaluates an n-ary operation f on n ciphertexts

(C1, . . . , Cn) using the public key pk, producing either a ciphertext or a failure symbol.

- ***Symmetric Key Encryption:***

The following three PPT stands for "probabilistic polynomial- time" algorithms make up a symmetric key encryption (SKE) scheme as follows

- Gen 1λ : Given a security parameter λ, it outputs an encryption key K.

- Enc K, M : Given an encryption key K and a plaintext

M, it outputs a ciphertext C.

- Dec K, C : Given an encryption key K and a ciphertext C as input, it outputs either a plaintext or an error symbol ⊥.

We require an SKE scheme to satisfy correctness: for any K Gen 1λ , any plaintext M, and any C Enc K, M , we always have M Dec K, C .

- ***Asymmetric Key Encryption:***

Asymmetric Key Encryption, also known as public-key cryptography, operates quite differently from symmetric key encryption. Instead of using a single key for both encryption and decryption, it employs a pair of keys: a public key and a private key. The basic operations involved in an asymmetric key encryption scheme are as follows:

- **Key Generation (Gen):** Gen(1^λ): Given a security parameter λ, this algorithm generates a pair of keys: a public key (PK) and a private key (SK). The public key is intended for encryption, while the private key is kept secret and used for decryption.
- **Encryption (Enc):** Enc(PK, M): Given a public key PK and a plaintext message M, this algorithm produces a ciphertext C. The ciphertext is generated in such a way that it can only be decrypted efficiently using the corresponding private key.
- **Decryption (Dec):** Dec(SK, C): Given a private key SK and a ciphertext C, this algorithm retrieves the original plaintext message M. It's important to note that decryption is computationally feasible only with the private key corresponding to the public key used for encryption.

The fundamental property of correctness still applies in asymmetric key encryption:

- **Correctness:** For any key pair (PK, SK) generated by Gen(1^λ), and for any plaintext message M, if C = Enc(PK, M), then Dec(SK, C) = M.

  This property ensures that messages encrypted with a public key can be successfully decrypted only by the corresponding private key, thus maintaining the integrity and confidentiality of communication in asymmetric key encryption systems.

FV Scheme: The FV scheme, named after its creators Shai Halevi and Craig Gentry, is a homomorphic encryption scheme that enables computation on encrypted data without decryption. It supports both addition and multiplication operations on encrypted data, maintaining privacy throughout computations.

BGV Scheme: The BGV scheme, developed by Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, is a homomorphic encryption scheme. It focuses on efficiency improvements and flexibility in parameter choices, allowing for optimized performance and customizable security levels in privacy-preserving computations.

Additive HE: Supports only addition operation.

Linear HE: Extends additive HE to include scalar multiplication.

d-level HE: Supports operations on ciphertexts of different levels, allowing for more complex computations.

- ***How does HE works:***

In HE, operations on ciphertexts are designed to correspond to operations on plaintexts.

When performing operations on ciphertexts, the result is encrypted and can be decrypted to obtain the result of the corresponding operation on plaintexts [].[2]

For example, in additive HE, adding two ciphertexts encrypted with the same public key corresponds to adding the plaintexts they represent.

Similarly, in linear HE, scalar multiplication of a ciphertext corresponds to scalar multiplication of the plaintext it represents.

In d-level HE, operations are defined based on the levels of ciphertexts, allowing for more flexibility in computations while maintaining security properties. The ciphertext level ensures that operations are performed correctly and securely.

IV. HOMOMORPHIC ENCRYPTION WITH AES: FUNDAMENTALS

Homomorphic encryption applied to AES involves implementing mathematical operations on ciphertexts in such a way that when these operations are performed, []3 they produce results that are consistent with the operations performed on the plaintext before encryption. In other words, the operations performed on encrypted data yield the same results as if they were performed on the plaintext data directly.

[2] T. El Gamal, "A public key cryptosystem and a signature schemebased on discrete logarithms," IEEE Trans. Inf. Theory, vol.31, no.4,pp.469–472, 1985.

[3] R. Canetti, S. Raghuraman, S. Richelson, and V. Vaikuntanathan, "Chosen-ciphertext secure fully homomorphic encryption," PKC 2017, pp.213–240, 2017.

One common approach to achieve homomorphic properties with AES is to use fully homomorphic encryption (FHE) schemes built on top of AES. FHE schemes, such as the Brakerski-Gentry-Vaikuntanathan (BGV) scheme or the Fan-Vercauteren (FV) scheme, allow for arbitrary computations on encrypted data. These schemes enable addition and multiplication operations on ciphertexts, which correspond to addition and multiplication operations on the plaintexts.

Under this framework, encryption involves converting plaintexts into ciphertexts using AES encryption. Then, using homomorphic properties, mathematical operations such as addition and multiplication can be performed directly on the ciphertexts. These operations are executed in such a way that they preserve the desired properties of the plaintext data.

For instance, in a scenario where two parties wish to compute the sum of their AES-encrypted data, they can use homomorphic addition to perform this operation directly on the ciphertexts. Similarly, if they need to perform multiplication operations on the encrypted data, homomorphic multiplication techniques can be applied.

This capability is invaluable in scenarios where data privacy is critical, such as secure computation in cloud environments or collaborative data analysis. It allows organizations to securely outsource computations to untrusted servers without compromising the confidentiality of their sensitive data. By leveraging homomorphic encryption with AES, organizations can ensure that their data remains encrypted throughout computations, mitigating the risks associated with exposing plaintext data to potential adversaries and enhancing overall data privacy and security.

V. TECHNIQUES FOR AES-BASED HOMOMORPHIC ENCRYPTION APPLICATIONS AND CHALLENGES

Achieving homomorphic encryption directly with AES (Advanced Encryption Standard) is challenging due to AES's symmetric nature, lacking inherent homomorphic properties. However, various techniques have been explored to integrate AES within a homomorphic encryption framework or to achieve functionalities akin to homomorphic encryption using AES. Here are some strategies:

A. Secure Multiparty Computation (SMC):

Secure Multiparty Computation (SMC) is a cryptographic technique that enables multiple parties to jointly compute a function over their private inputs without revealing those inputs to each other. While AES itself doesn't directly support SMC, it can be used within an SMC framework to provide encryption of data involved in the computation. Here's how SMC can be applied in an AES-based homomorphic encryption setting:

Overview:

Secure Multiparty Computation (SMC): SMC allows multiple parties to compute a function on their private inputs while keeping those inputs confidential.

AES-based Homomorphic Encryption: AES is a symmetric encryption algorithm that can be used to encrypt data within an SMC framework, enabling secure computation on encrypted inputs.

Working Principle:

Data Encryption:

Each party encrypts its private input using AES encryption before sharing it with the other parties involved in the computation. This ensures that the inputs remain confidential during the computation.

Secure Computation:

The parties perform the desired computation on the encrypted inputs within the SMC framework. This computation could involve arithmetic operations (e.g., addition, multiplication) or more complex functions.

Result Decryption:

After the computation is completed, the parties jointly decrypt the result using a secure protocol. Since AES is symmetric, all parties must agree on the decryption key to decrypt the result.

AES-based Homomorphic Encryption within the SMC framework allows multiple parties to compute a function on their private inputs while preserving the confidentiality of those inputs, thereby enabling secure computation on encrypted data.

Applications :

-Secure Auctions: SMC can facilitate secure auctions where bidders can submit their bids without revealing them to other participants until the end of the auction price.

This prevents bid manipulation and collusion.

-Privacy-preserving data analytics: SMC allows multiple parties to jointly analyze sensitive data without revealing their individual inputs.This is useful in situations such as healthcare research, financial analysis, and market research.

-Voting system: SMC can be applied to design a secure electronic voting system where voters can vote anonymously and maintain the integrity of the election process without revealing the votes of each individual.

Challenges : Secure Multiparty Computation (SMC) faces challenges in efficiency, scalability, communication overhead, trust assumptions, and key management. Efficiency concerns arise due to the computational intensity of SMC protocols, while scalability issues emerge with the growing number of parties involved. Communication overhead is a challenge due to multiple rounds of communication; trust assumptions require careful consideration in adversarial environments, and key management presents difficulties in distribution, revocation, and storage. Addressing these challenges is crucial for practical deployment of SMC in secure and privacy-preserving computation.

B. Hybrid Cryptosystems:

Hybrid cryptosystems in an AES-based homomorphic encryption context involve combining the features of symmetric and asymmetric encryption schemes within a homomorphic encryption framework. This approach leverages AES for efficient symmetric encryption of data and incorporates asymmetric encryption for secure key exchange and other cryptographic functionalities. Here's how hybrid cryptosystems can be applied in an AES-based homomorphic encryption setting:

Overview:

Hybrid Cryptosystems: Hybrid cryptosystems combine the efficiency of symmetric encryption with the security benefits of asymmetric encryption, offering a balanced approach to encryption.

Working Principle:

Symmetric Encryption (AES):

The data owner encrypts their data using AES symmetric encryption, generating ciphertexts that are efficiently processed.

Asymmetric Encryption:

The data owner encrypts the symmetric encryption key (DEK) used in AES with the public key of the intended recipient, ensuring secure key exchange.

Alternatively, asymmetric encryption can be used for other cryptographic functionalities such as digital signatures or secure communication.

Homomorphic Operations:

The encrypted data and keys can be processed within a homomorphic encryption framework,allowing computations to be performed on the ciphertexts without decryption.

Homomorphic operations such as addition and multiplication can be applied to the ciphertexts, enabling privacy-preserving data analysis and secure collaborative computation.

Decryption:

The recipient decrypts the symmetric encryption key using their private key, allowing them to
decrypt the data encrypted with AES and perform further computations or analysis.

Applications :

-Secure Communication: Hybrid cryptosystems are widely used to secure communication channels, such as SSL/TLS for securing web traffic. Asymmetric encryption is used for key exchange and authentication, while symmetric encryption is used for bulk data transmission.

- Data Storage: Hybrid cryptosystems are employed to secure stored data in databases, file systems, and cloud storage services. Asymmetric encryption can be used to encrypt symmetric keys, which in turn encrypt the actual data.

Challenges : Hybrid cryptographic systems face challenges in key management, algorithm selection, performance overhead, integration complexity, and security risks.Effectively addressing these challenges is critical to ensuring the robustness and effectiveness of hybrid cryptographic systems in securing communication channels, data storage, and digital signatures, along with other applications.

C. Proxy Re-Encryption:

Proxy Re-Encryption (PRE) is a [4] cryptographic technique that allows a semi-trusted proxy to transform ciphertexts encrypted under one key into ciphertexts that can be decrypted under another key, without the need to decrypt and re-encrypt the data. While AES itself doesn't directly support PRE, it can be used within a PRE framework to provide encryption and decryption capabilities.

Here's how PRE can be applied in an AES-based homomorphic encryption setting:

Overview:

Proxy Re-Encryption (PRE): PRE enables a proxy entity to transform ciphertexts from one encryption key to another, facilitating secure data sharing and delegation of access rights. AES-based Homomorphic Encryption: AES is a symmetric encryption algorithm that can be used for data encryption and decryption within a PRE framework.

Working Principle:

Initial Encryption: The data owner encrypts their data using AES encryption with their own secret key, generating ciphertexts that only they can decrypt.

Proxy Re-Encryption: The data owner delegates access rights to specific recipients by providing them with re encryption keys. The proxy entity, armed with the re-encryption keys, transforms the ciphertexts encrypted under the data owner's key into ciphertexts that can be decrypted by the recipients' keys using a proxy re-encryption algorithm.

Decryption: The recipients decrypt the transformed ciphertexts using their own secret keys, obtaining the original plaintext data.

Applications :

-Content Distribution: PRE can be used for secure content distribution, allowing content providers to encrypt data once and delegate re-encryption to proxies for distribution to different users or devices, without compromising data confidentiality.

-Secure Messaging: PRE can enhance the privacy and security of messaging applications by allowing messages to be encrypted once by the sender and re-encrypted for different recipients by proxies, ensuring end-to-end encryption without the need for the sender to manage multiple keys.

Challenges : Proxy re-encryption (PRE) faces challenges in key management, proxy reliability, performance overhead, scalability, and privacy issues.Effective key management, reliable proxy assurance, performance optimization, scalability solutions, and privacy protection mechanisms are

[4] Efficient Homomorphic Proxy Re-Encryption for Arithmetic Circuit Evaluation" by Zhoujun Li, Wenjing Lou, and Y. Thomas Hou. (Reference: https://ieeexplore.ieee.org/document/6562705

essential to successfully deploying PRE to enable access control and share data securely.

D. Homomorphic Properties of AES-Like Ciphers:

Homomorphic properties of AES-like ciphers in AES-based homomorphic encryption refer to the ability of these ciphers to preserve certain algebraic operations on encrypted data, allowing computations to be performed on ciphertexts directly without decryption. While AES itself lacks inherent homomorphic properties, researchers have explored the development of AES-like ciphers with homomorphic capabilities within a homomorphic encryption framework. Here's a brief overview:

Overview:

AES-Like Ciphers: These are encryption algorithms designed to mimic the structure and security properties of AES while incorporating homomorphic properties. Homomorphic Encryption Framework: AES-like ciphers with homomorphic properties operate within a homomorphic encryption framework, enabling computations on encrypted data without decryption. Homomorphic Operations: Homomorphic encryption schemes support operations such as addition and multiplication on encrypted data, allowing mathematical computations to be performed on ciphertexts. Homomorphic Properties:

1. Additive Homomorphism: AES-like ciphers with additive homomorphic properties preserve addition operations on ciphertexts. When two ciphertexts encrypted under the same key are added together, the result decrypts to the sum of the corresponding plaintexts.
2. Multiplicative Homomorphism: Some AES-like ciphers exhibit multiplicative homomorphic properties, preserving multiplication operations on ciphertexts. When two ciphertexts encrypted under the same key are multiplied together, the result decrypts to the product of the corresponding plaintexts.

Key Components:

1. AES Encryption: Utilize the AES algorithm for encrypting data or intermediate values within the homomorphic encryption scheme. AES provides efficient and secure encryption of data blocks.
2. Homomorphic Encryption Scheme: Incorporate a homomorphic encryption scheme that supports the desired homomorphic operations, such as addition and multiplication, on the encrypted data. 3. Key Management: Implement secure key management practices to ensure the confidentiality and integrity of encryption keys used in both AES and the homomorphic encryption scheme.

Applications :

-Secure Outsourcing: Organizations can outsource computational tasks to untrusted servers while safeguarding data privacy using homomorphic AES-like ciphers. This allows for secure cloud computing and data processing without exposing sensitive information.

-Secure Messaging: Homomorphic properties of AES-like ciphers empower secure messaging applications to perform operations on encrypted messages without decryption. This enhances privacy and confidentiality in communication channels.

-Privacy-Preserving Machine Learning: Homomorphic AES-like ciphers enable secure computation on encrypted machine learning models and data. Multiple parties can collaborate on machine learning tasks while preserving the privacy of their sensitive information.

Challenges : Developing homomorphic properties in AES-like ciphers presents challenges in security assurance, computational efficiency, key management, and algorithmic complexity. Balancing security with computational overhead, securely managing cryptographic keys, and validating complex algorithms are essential for realizing the potential of homomorphic AES-like ciphers in enabling secure and privacy-preserving computation.

VI. REAL-WORLD APPLICATIONS AND USE CASES

Homomorphic encryption schemes using AES can be applied in various real-world scenarios across different domains. Here are some examples:

Secure Outsourcing of Data Processing: Homomorphic encryption allows computations to be performed on encrypted data without decrypting it first. This is particularly useful in scenarios where sensitive data needs to be processed by untrusted third parties, such as cloud service providers. For instance, a company could outsource data analytics tasks to a cloud provider while keeping the data encrypted. The cloud provider can perform computations on the encrypted data using homomorphic encryption, preserving data privacy.

Healthcare Data Analysis: In healthcare, patient data is highly sensitive and subject to strict privacy regulations. Homomorphic encryption can enable secure data analysis on encrypted medical records. For example, hospitals could collaborate with research institutions to perform statistical analysis on encrypted patient data without compromising patient privacy.

Financial Data Analysis: Financial institutions deal with large volumes of sensitive financial data that need to be analyzed for various purposes such as risk assessment, fraud detection, and customer profiling. Homomorphic encryption can be used to securely analyze this data while keeping it encrypted, thus ensuring confidentiality and compliance with regulations like GDPR or PCI-DSS.

Secure Multi-Party Computation (SMPC): Homomorphic encryption can facilitate secure multi-party computation where multiple parties wish to jointly compute a function over their inputs while keeping those inputs private. For example, in a scenario where several organizations want to calculate aggregate statistics from their individual datasets without revealing the raw data, homomorphic encryption enables this computation to be performed securely.

Privacy-Preserving Machine Learning: Homomorphic encryption can also be used to train machine learning models on encrypted data while preserving data privacy. This is particularly relevant in situations where data owners are concerned about sharing their sensitive data with third parties. With homomorphic encryption, data can remain encrypted throughout the training process, and only the encrypted model parameters are shared or used for prediction.

Secure IoT Data Processing: With the proliferation of Internet of Things (IoT) devices, there's a growing need to process sensitive data collected from these devices while preserving privacy. Homomorphic encryption can enable secure and privacy-preserving data processing in IoT environments, allowing for analysis and decision-making without exposing raw sensor data to unauthorized parties.

Blockchain and Cryptocurrency: Homomorphic encryption can enhance the privacy and confidentiality of transactions in blockchain networks. By encrypting transaction data homomorphically, participants can perform certain operations on the encrypted data within smart contracts while keeping the underlying transaction details confidential.[1][2][3][4][5].

VII. THE CHALLENGES AND FUTURE DIRECTIONS

Homomorphic encryption, especially when based on AES (Advanced Encryption Standard)[5], holds great promise for secure computation over encrypted data. However, several challenges and opportunities for future research and development remain in this field, however, several challenges and opportunities for future research and development persist.[6]

1. Performance Optimization: The primary challenge with AES-based homomorphic encryption is the computational overhead. AES is a symmetric encryption algorithm, and performing homomorphic operations on encrypted data often involves complex mathematical operations, which can lead to significant computational costs. Future research should focus on improving the performance of AES-based homomorphic encryption schemes, in order to make them more practical for real-world applications.

2. Security Analysis[7]: Although AES is a widely utilized encryption standard that is renowned for its security, its implementation in a homomorphic encryption context introduces additional security considerations. In the future, it is imperative to conduct comprehensive security analyses of homomorphic encryption schemes based on AES in order to guarantee that they offer the necessary levels of confidentiality, integrity, and authenticity.

3. Scalability: As the volume of data increases, scalability emerges as a crucial concern in homomorphic encryption. Future research should examine methods to enhance the scalability of AES-based homomorphic encryption schemes, thereby enabling efficient computation over vast datasets without compromising security or performance.

4. Homomorphic Operations Support: AES-based homomorphic encryption schemes typically offer a restricted range of homomorphic operations, such as addition and multiplication. Future research should aim to broaden the range of supported operations to facilitate more intricate computations on encrypted data, thereby enhancing the utility of homomorphic encryption in diverse domains.

5. Key Management[8]: An efficient key management system is essential for the secure deployment of AES-based homomorphic encryption schemes. Future research should be focused on developing robust key management mechanisms that can handle the complexities of homomorphic encryption while ensuring the confidentiality and integrity of encryption keys.[3][6]

6. Standardization and Interoperability: The establishment of standards for AES-based homomorphic encryption can facilitate interoperability and encourage adoption across diverse platforms and applications. Future research should prioritize standardization initiatives to guarantee compatibility and ease of integration with existing systems and protocols.

7. Hardware Acceleration: The utilization of specialized hardware, such as secure enclaves or hardware accelerators, can significantly enhance the performance of AES-based homomorphic encryption schemes. Future research should explore hardware-based approaches to accelerate homomorphic computations while still maintaining security guarantees.

[5] Garrison, G., Wakefield, R. L., & Kim, S. (2015). The effects of IT capabilities and delivery model on cloud computing success and firm performance for cloud supported processes and operations. International Journal of Information Management, 35, 377-393.

[6] Zhang, D., Feng, G., Shi, Y., & Srinivasan, D. (2021). Physical Safety and Cyber Security Analysis of Multi-Agent Systems: A Survey of Recent Advances. IEEE/CAA Journal of Automatica Sinica, 8, 319-333.

[7] Dobraunig, C., Grassi, L., Helminger, L., Rechberger, C., Schofnegger, M., & Walch, R. (2023). Pasta: A Case for Hybrid Homomorphic Encryption. IACR Cryptology ePrint Archive, 2023, 30-73

[8] P, A., Sharma, A., Singla, A., Sharma, N., & V, D. G. (2022). IoT Group Key Management using Incremental Gaussian Mixture Model. In International Conference Electronic Systems, Signal Processing and Computing Technologies [ICESC-] (pp. 469-474).

8. Privacy-Preserving Machine Learning: Homomorphic encryption has the potential to allow privacy-preserving machine learning by allowing computations on encrypted data. Future research should focus on developing AES-based homomorphic encryption schemes for machine learning applications, which would enable secure and privacy-preserving model training and inference.[8]

9. Usability and Accessibility: The implementation and utilization of AES-based homomorphic encryption is imperative for its widespread adoption. Future research should prioritize usability and accessibility by developing user-friendly tools, libraries, and frameworks that will make it easier for developers to integrate homomorphic encryption into their applications.

10. Real-World Applications: Ultimately, it is imperative to validate the practicality and efficacy of AES-based homomorphic encryption in real-world applications in order to facilitate its adoption. Research should focus on demonstrating the feasibility and performance of homomorphic encryption in various use cases, such as secure outsourcing of computations, privacy-preserving data analytics, and secure multiparty computation.

Exploring these future directions will contribute to the advancement of AES-based homomorphic encryption and pave the way for its widespread adoption in securing sensitive data while enabling secure computation over encrypted information. Addressing these challenges and advancing AES-based homomorphic encryption techniques will be crucial for broader adoption and seamless integration into real-world scenarios.

VIII. CONCLUSION

In conclusion, this article explored the potential of Homomorphic Encryption using the established Advanced Encryption Standard (AES) algorithm. We delved into the fundamentals of this approach, examining various techniques for performing computations on encrypted data with AES. By showcasing real-world applications and use cases, we've highlighted the transformative potential of this technology in areas like cloud computing and secure data analysis. However, challenges remain, such as computational overhead and limited operation support. As research progresses, overcoming these hurdles will unlock the full potential of AES-based Homomorphic Encryption, paving the way for a future where data security and usability coexist seamlessly.

REFERENCES

[1] Gentry, C. (2009). A fully homomorphic encryption scheme. Ph.D. Dissertation, Stanford University. . (n.d.).

[2] Chen, B., Wang, Y., Zhang, Y., & Yang, J. (2014). Efficient privacy-preserving biometric identification based on homomorphic encryption. In Information Security and Cryptology - ICISC 2014 (pp. 98-112). Springer, Cham. . (n.d.).

[3] Dijk, M. Van, Gentry, C., Halevi, S., Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers. In Advances in Cryptology – EUROCRYPT 2010 (pp. 24-43). Springer Berlin Heidelberg. (n.d.).

[4] Smart, N. P. (2013). Fully homomorphic encryption with relatively small key and ciphertext sizes. In Advances in Cryptology – EUROCRYPT 2013 (pp. 340-357). Springer Berlin Heidelberg. (n.d.).

[5] Juels, A., & Ristenpart, T. (2014). Honey encryption: Security beyond the brute-force bound. In Advances in Cryptology – EUROCRYPT 2014 (pp. 293-310). Springer Berlin Heidelberg. (n.d.).

[6] Bogetoft, P., Christensen, D., Damgård, I., & Geisler, M. (2012). Secure multiparty computation goes live. In Advances in Cryptology – EUROCRYPT 2012 (pp. 325-342). Springer Berlin Heidelberg. (n.d.).

[7] Henecka, W., & Pohl, H. (2010). Privacy-preserving data analysis on grids. In Proceedings of the 2010 ACM workshop on Cloud computing security workshop (pp. 19-30). (n.d.).

[8] Bost, R., Popa, R. A., Tu, S., Goldwasser, S., & Boneh, D. (2015). Machine learning classification over encrypted data. In 2015 IEEE Symposium on Security and Privacy (pp. 1-17). IEEE. (n.d.).

[9] Goldwasser, S., & Rothblum, G. N. (2008). How to compute on encrypted data. In Advances in Cryptology – CRYPTO 2008 (pp. 276-293). Springer Berlin Heidelberg. . (n.d.).