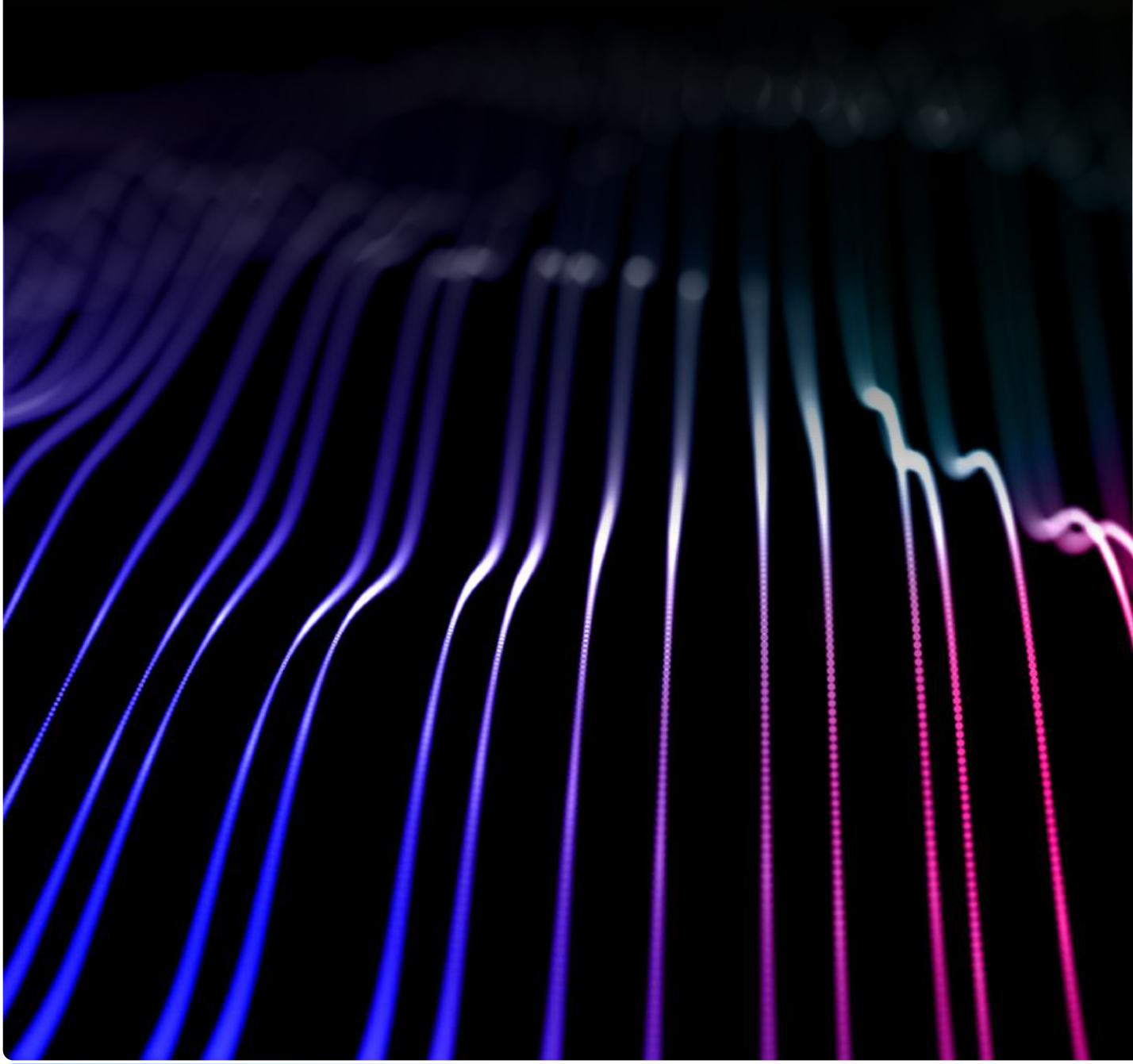
HCLTech | Supercharging Progress™

Homomorphic encryption: Exploring technology trends and future approach



Introduction

Homomorphic encryption represents a groundbreaking advancement in cryptography, enabling computations on encrypted data without decryption. This transformative capability holds immense opportunity to enhance privacy, security and data confidentiality across various domains. By allowing computations to be performed directly on encrypted data, homomorphic encryption mitigates the risks associated with data exposure during processing and transmission.

In this white paper, we delve into the technology behind homomorphic encryption, its current state-of-the-art implementations and the potential it holds for future experimentation and innovation.

Section 1: Technological advances in homomorphic encryption

In the first section, we describe the technological advances that have propelled homomorphic encryption forward. We provide a comprehensive overview of the historical timeline, tracing the evolution of homomorphic encryption from its theoretical foundations to its practical applications today. We discuss the availability and features of popular homomorphic encryption libraries.

Section 2: Experiments with open-source homomorphic encryption library

The second section of the paper focuses on practical experimentation with homomorphic encryption using open-source libraries. With the help of hands-on experiments, we demonstrate the efficiency of the encryption-decryption process using Fully Homomorphic Encryption (FHE) in terms of time taken and storage consumed.

Section 3: The future of homomorphic encryption

In the final section, we turn our attention to the future of homomorphic encryption and the ecosystem surrounding it. We discuss ongoing research efforts aimed at advancing homomorphic encryption techniques and addressing challenges such as performance optimization, security enhancements and standardization. Emerging trends and applications of homomorphic encryption in domains such as healthcare, finance, cloud computing and machine learning are explored, along with the role of industry collaborations, regulatory frameworks and community engagement in shaping the future landscape of homomorphic encryption.

Section 1: Technological advances in homomorphic encryption

In this section, we cover the basic definition of homomorphic encryption, its types, the encryption schemes associated with FHE and the chronology of technological advances in this area.

What is homomorphic encryption

Homomorphic encryption is a cryptographic technique enabling computations on encrypted data without decrypting it first. The fundamental idea behind homomorphic encryption is to design encryption schemes that support addition and multiplication on ciphertexts.

The resultant of this operation, when decrypted, yields results equivalent to performing the same operations on the plaintexts. Hence, facilitating a user to perform operations on the data stored on the cloud without even decrypting it, as shown in the figure below.

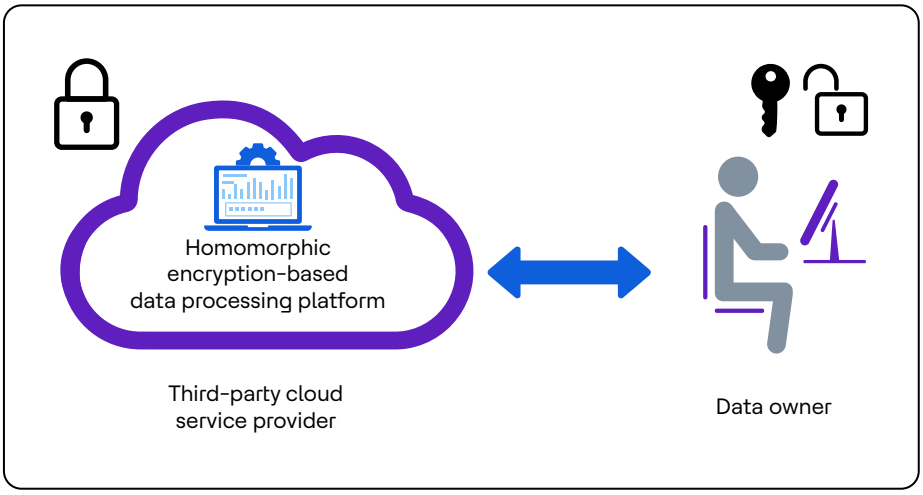


Figure 1 Homomorphic encryption example

There are various types of homomorphic encryption depending on the diverse needs and requirements of different applications and scenarios -

Partially Homomorphic Encryption (PHE):

PHE schemes support only one type of homomorphic operation, either addition or multiplication, but not both. PHE schemes are relatively simple and computationally efficient when compared to more advanced homomorphic encryption schemes, making them suitable for certain applications where simplicity and efficiency are prioritized over functionality. One such example includes the RSA cryptosystem, which supports homomorphic multiplication.

Somewhat Homomorphic Encryption (SHE):

SHE schemes support a limited number of both addition and multiplication operations on encrypted data but have constraints on the depth of computations due to noise accumulation. It strikes a balance between functionality and efficiency, supporting a limited set of computations while still preserving security. While SHE cannot support arbitrary computations like FHE, it is often more practical and efficient for applications that require basic computations on encrypted data. Some of the examples are Paillier cryptosystem and Benaloh cryptosystem.

Fully Homomorphic Encryption (FHE):

FHE schemes enable arbitrary computations on encrypted data, allowing both addition and multiplication operations, as well as more complex computations. These schemes typically involve bootstrapping techniques to refresh ciphertexts and manage noise, enabling deeper computations. Examples of such schemes include the original Gentry FHE scheme and its variants, such as BGV and BFV. Despite being the most advanced and versatile form of homomorphic encryption, offering maximum functionality and flexibility, FHE schemes tend to be more computationally intensive and have higher overhead compared to PHE and SHE, making them less practical for some applications where efficiency is critical.

Since FHE is the most desirable and secure encryption algorithm, this white paper will focus on FHE implementation only. Henceforth, in this paper, terms like homomorphic encryption and FHE will be used interchangeably. Depending on the mathematical principles and data types that are supported, there are three prominent encryption techniques for FHE implementation -

- Brakerski-Gentry-Vaikuntanathan (BGV).
- Brakerski-Fan-Vercauteren (BFV).
- Cheon-Kim-Kim-Song (CKKS) schemes.

Here's an overview of each, including their underlying mathematical principles and differences.

Brakerski-Gentry-Vaikuntanathan (BGV):

The BGV scheme is one of the first FHE schemes proposed and is formed by lattice-based cryptography, particularly the Learning with Errors (LWE) problem. It leverages bootstrapping to perform FHE, allowing arbitrary computations on encrypted data. It provides strong security guarantees, but can be computationally expensive, especially for complex operations and large ciphertext sizes.

Brakerski-Fan-Vercauteren (BFV):

Similar to BGV, the BFV scheme is based on lattice-based cryptography, specifically the Ring Learning with Errors (Ring-LWE) problem. Furthermore, it supports fully homomorphic encryption and employs e-modulus switching and noise management to improve efficiency. BFV builds upon BGV by introducing optimizations to reduce the computational overhead associated with homomorphic operations, making it more practical for certain applications like text manipulations.

Cheon-Kim-Kim-Song (CKKS):

The CKKS scheme is designed specifically for homomorphic encryption of real and complex numbers. It is based on the Approximate GCD problem over rings of polynomials and leverages approximate homomorphic encryption and modulus switching techniques. CKKS is tailored for applications that involve computations on real or complex numbers, such as machine learning and signal processing. It supports approximate computations with floating-point numbers, facilitating more efficient and accurate operations on encrypted data compared to schemes designed for integer arithmetic.

In summary, BGV, BFV and CKKS are all FHE schemes with different underlying mathematical principles and optimizations. BGV and BFV utilizes lattice-based cryptography, while CKKS is tailored for real and complex number computations. Each scheme has its own strengths and limitations, making them suitable for different use cases and performance requirements.

Homomorphic encryption – Chronology

The concept of homomorphic encryption was first introduced in the 1970s, but significant advancements in recent years have made it more practical and feasible for real-world applications. Here's a simplified timeline of key developments in the field of homomorphic encryption:

- 1978: Rivest, Adleman and Dertouzos propose the RSA cryptosystem, laying the basis for modern public-key cryptography. While not strictly homomorphic, RSA is partially homomorphic with respect to multiplication.
- 1994: Josh Benaloh introduced a cryptosystem based on the difficulty of computing roots in finite fields, providing a practical example of a PHE scheme.
- 1999: Pascal Paillier introduces a probabilistic encryption scheme based on the difficulty of the decisional composite residuosity problem, which supports homomorphic addition.
- 2009: Craig Gentry published his groundbreaking paper 'A Fully Homomorphic Encryption Scheme', outlining the first FHE scheme, demonstrating theoretical possibility to perform arbitrary computations on encrypted data without decryption. This work initiates a new era in cryptography.
- 2010s: Gentry, Sahai and Waters presented improvements to the original FHE scheme, making it more practical and efficient.
- 2011: Brakerski and Vaikuntanathan introduced the first SHE scheme capable of both addition and multiplication operations, providing a more efficient alternative to FHE.
- 2013: Brakerski and Vaikuntanathan devised a bootstrapping technique for FHE, significantly improving its efficiency and practicality.
- 2016: The Microsoft Research team released the Simple Encrypted Arithmetic Library (SEAL), an open-source library for homomorphic encryption, making it accessible to developers and researchers.
- 2016: The Microsoft Research team released the Simple Encrypted Arithmetic Library (SEAL), an open-source library for homomorphic encryption, making it accessible to developers and researchers.
- 2017: The PALISADE library is introduced, offering a comprehensive suite of lattice-based cryptography tools, including homomorphic encryption implementations.
- 2018: The HomomorphicEncryption.org community is established to promote collaboration and standardization efforts in the field of homomorphic encryption.
- 2019: ISO/IEC WD 18033-8, the International Organization for Standardization and the International Electrotechnical Commission initiated an official project to establish FHE standards.
- 2020: The National Institute of Standards and Technology (NIST) launched a standardization process for post-quantum cryptography, including lattice-based cryptography, which underpins many homomorphic encryption schemes.

- 2021: DARPA initiated the DPRIVE program, aiming to create a hardware accelerator for FHE computations. This accelerator is expected to significantly decrease the compute runtime overhead in comparison to software-based FHE approaches.
- 2024: Homomorphic encryption remains an active area of research, with ongoing efforts to address practical challenges and promote its adoption in real-world scenarios. The 3rd Annual FHE.org Conference held on 24 March 2024 in Toronto on FHE.

Open-source library

The advancements in both the logical foundations and the practical implementation of FHE have been closely intertwined with the evolution of technology and the enrichment of open-source libraries. As the theoretical foundations of FHE have matured, simultaneous efforts have been made to translate these advancements into practical implementations in software libraries. Some of the prominent open-source libraries for FHE are listed below.

Microsoft Simple Encrypted Arithmetic Library (SEAL):

Developed by Microsoft Research, SEAL is an open-source library for homomorphic encryption offering implementations of several homomorphic encryption schemes, including the BFV scheme for fully homomorphic encryption and the CKKS scheme for approximate homomorphic encryption. SEAL is designed to be efficient and flexible, supporting various encryption parameters and optimizations for performance.

PALISADE:

PALISADE is an open-source library for lattice-based cryptography, including homomorphic encryption. It is developed by researchers from several institutions. PALISADE provides implementations of various homomorphic encryption schemes, such as the BGV scheme for fully homomorphic encryption supporting functionalities like key generation, encryption, decryption and homomorphic operations, with a focus on security and performance.

Homomorphic Encryption Library (HElib):

HElib is a C++ library developed by researchers at IBM to implement the BGV fully homomorphic encryption scheme, offering tools for key generation, encryption, decryption and homomorphic operations, along with optimizations for performance. HElib is widely used in research and education for experimenting with homomorphic encryption and developing applications.

Standardization consortium (HESC) libraries:

The HESC is working on standardizing homomorphic encryption and developing reference implementations of standardized schemes. While specific libraries may not yet be available, the consortium's efforts aim to provide standardized APIs and implementations for homomorphic encryption, promoting interoperability and adoption.

SEAL-Python:

SEAL-Python is a Python wrapper for Microsoft SEAL, allowing users to access SEAL's functionalities from Python scripts empowering researchers and developers to experiment with homomorphic encryption in a Python environment, leveraging SEAL's features for encrypted computation.

Here is a comparison of the functionality of these libraries:

Name	Developer	BGV	CKKS	BFV	CKKS bootstrapping	Description
HElib	IBM	Yes	Yes	No	No	BGV scheme with the GHS optimizations.
Microsoft SEAL	Microsoft	Yes	Yes	Yes	No	
OpenFHE	Duality Technologies, Samsung Advanced Institute of Technology [kr], Intel, MIT, University of California, San Diego and others.	Yes	Yes	Yes	Yes	Successor to PALISADE.
PALISADE	New Jersey Institute of Technology, Duality Technologies, Raytheon BBN Technologies, MIT, University of California, San Diego and others.	Yes	Yes	Yes	No	General-purpose lattice cryptography library. Predecessor of OpenFHE

Figure 2 FHE library comparison

To support basic functionality in FHE libraries, it's essential to implement key operations that enable encryption, decryption and homomorphic computations on encrypted data.

Here are the key functions that each library should ideally support:

- Key generation: This function generates the public and private keys used for encryption and decryption of data respectively.
- Encryption: The encryption function takes plaintext data and the public key as input and produces ciphertext, which is the encrypted form of the data which can be used for secure transmission or storage.
- Decryption: The decryption function takes ciphertext and the private key as input and produces the original plaintext data. However, only entities with the correct private key can decrypt the ciphertext and recover the original data.
- Addition: Homomorphic encryption schemes that support addition allow for summation of two ciphertexts together to perform addition on the plaintexts they represent. The result is a new ciphertext that, when decrypted, yields the sum of the plaintexts.
- Multiplication: Homomorphic encryption schemes that support multiplication allow multiplying two ciphertexts together to perform multiplication on the plaintexts they represent. The result is a new ciphertext that, when decrypted, yields the product of the plaintexts.
- Bootstrapping: Bootstrapping is a technique used in fully homomorphic encryption schemes to reduce noise accumulation in ciphertexts, allowing for deeper and more complex computations. It involves refreshing ciphertexts without revealing their plaintexts, thereby maintaining data confidentiality.

- Validity check: This function is used to check the validity of ciphertexts or keys to ensure that they have not been tampered with or corrupted. It is an essential security measure in homomorphic encryption to prevent attacks and ensure data integrity.

Section 2: Experimental implementation

This experiment showcases the practical benefits of homomorphic encryption in real-world cloud computing scenarios. To start, the user encrypts their data using a homomorphic encryption scheme, generating ciphertexts. These ciphertexts are then uploaded to the cloud storage. The cloud service provider has no access to the plaintext data. When the user needs to perform operations or computations on the encrypted data, they do so directly within the cloud environment using homomorphic encryption. Once the desired operations have been completed on the encrypted data in the cloud, the user retrieves the resulting ciphertexts from the cloud storage. These ciphertexts are then decrypted locally on the user's device using the corresponding decryption key.

Here, we have used two approaches: one is a complete homomorphic encryption approach and the other is a hybrid approach where a combination of homomorphic and Advanced Encryption Standard (AES) encryption is used.

Homomorphic encryption experiment setup:

User personal data encryption:

User personal data, in the form of text strings, is encrypted using a homomorphic encryption scheme before being uploaded to the cloud.

Cloud storage

The encrypted user's personal data is stored in the cloud which provides a convenient and scalable platform for storing large volumes of encrypted data securely.

String search operation:

The string search operation involves searching for a specific substring or pattern within the encrypted data without revealing the plaintext content to the cloud service provider. The homomorphic encryption scheme used in this experiment supports the required operations for string search, such as comparison and pattern matching, while preserving the confidentiality of the encrypted data. The scheme enables the user to perform computations on the encrypted data, such as searching for specific substrings, without decrypting the data.

Search result retrieval:

After performing the string search operation on the encrypted data stored in the cloud, the user receives the encrypted blocks that match the search criteria. The user can then decrypt the retrieved encrypted blocks locally using the decryption key to obtain the corresponding plaintext results.

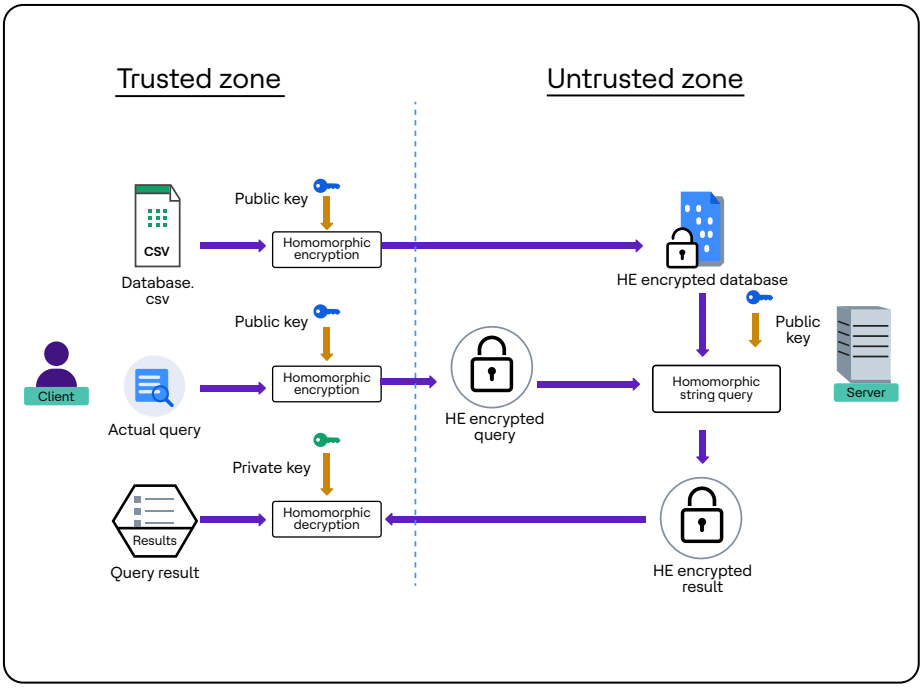


Figure 3 Experimental setup for homomorphic encryption

This experiment is implemented using Helib operating a 11th Gen Intel(R) Core (TM) i7-1185G7 @ 3.00GHz CPU with 16 GB memory and different size data set starting from five rows of user data to 50 rows of user data with a unique key. The table below showcases the result of an experiment that includes the time taken to encrypt, decrypt and search data in the setup:

Dataset count	Homomorphic encryption time (ms)	String Search	Homomorphic decryption time (ms)	Actual database file size (bytes)	Homomorphically encrypted database file size (bytes)	factors increased in size
5	152	21796	128	146	1137688	7792.38
25	472	94739	146	571	4930029	8634.03
50	820	181913	179	1114	9671254	8681.56
75	1293	288467	137	1647	14411376	8750.08
100	1357	381182	149	2193	19152885	8733.65
125	1894	376454	89	2749	23893355	8691.65
150	1594	458856	120	3319	28633165	8627.05
175	1684	547519	139	3892	33374011	8575.03
200	1987	580652	106	4441	38115551	8582.65
225	2737	742621	181	5021	42855051	8535.16
250	2717	753387	120	5598	47596328	8502.38

Figure 4 Table containing processing time and storage consumption for experiment one

Observation

The following are the highlights of the experiment:

String search time and data encryption time:

As the number of datasets increases, the string search time and data encryption time increase proportionally as the number of complex mathematical operations increases with the size of datasets, which in turn requires more computational resources and time.

Additionally, string search operations involve iterating through the dataset to find matching patterns or substrings, which becomes more time-consuming as the dataset size grows.

Decryption time:

However, the decryption time remains constant, as fetching one dataset from the cloud for decryption involves a fixed amount of computation regardless of the dataset size.

Storage requirements:

With the increase in data, the storage requirements increase significantly. Homomorphic encryption schemes result in enormous ciphertext expansion, where the size of the encrypted data is much larger than the original plaintext data. Consequently, storing larger volumes of encrypted data in the cloud requires more storage space, leading to increased storage costs and potential scalability issues.

Mixed approach experimental setup:

The experiment involves encrypting the data using a hybrid approach:

User setup:

Data indices are encrypted using homomorphic encryption, supporting efficient string search operations while preserving privacy. Other data (e.g., text, documents) is encrypted using AES encryption, resulting in AES-encrypted blobs. The user generates a key for homomorphic and AES encryption and creates two encrypted databases.

Cloud setup:

The cloud storage consists of two separate storage systems.

- Storage for homomorphically encrypted data indices: This storage contains the encrypted data indices generated using the homomorphic encryption scheme.
- Storage for AES-encrypted blobs: This storage contains the actual data (e.g., text documents) encrypted using AES, with each blob associated with its corresponding index.

String query processing:

When a user submits a string query, the system performs the following steps:

The system searches the homomorphically encrypted data indices to find matching indices for the query. Once matching indices are found, the system retrieves the corresponding AES-encrypted blobs from the second storage system based on the indices. And returns the blob.

The retrieved blobs are then decrypted at the user end using the AES decryption key, allowing the user to get the plaintext data matching the user's query.

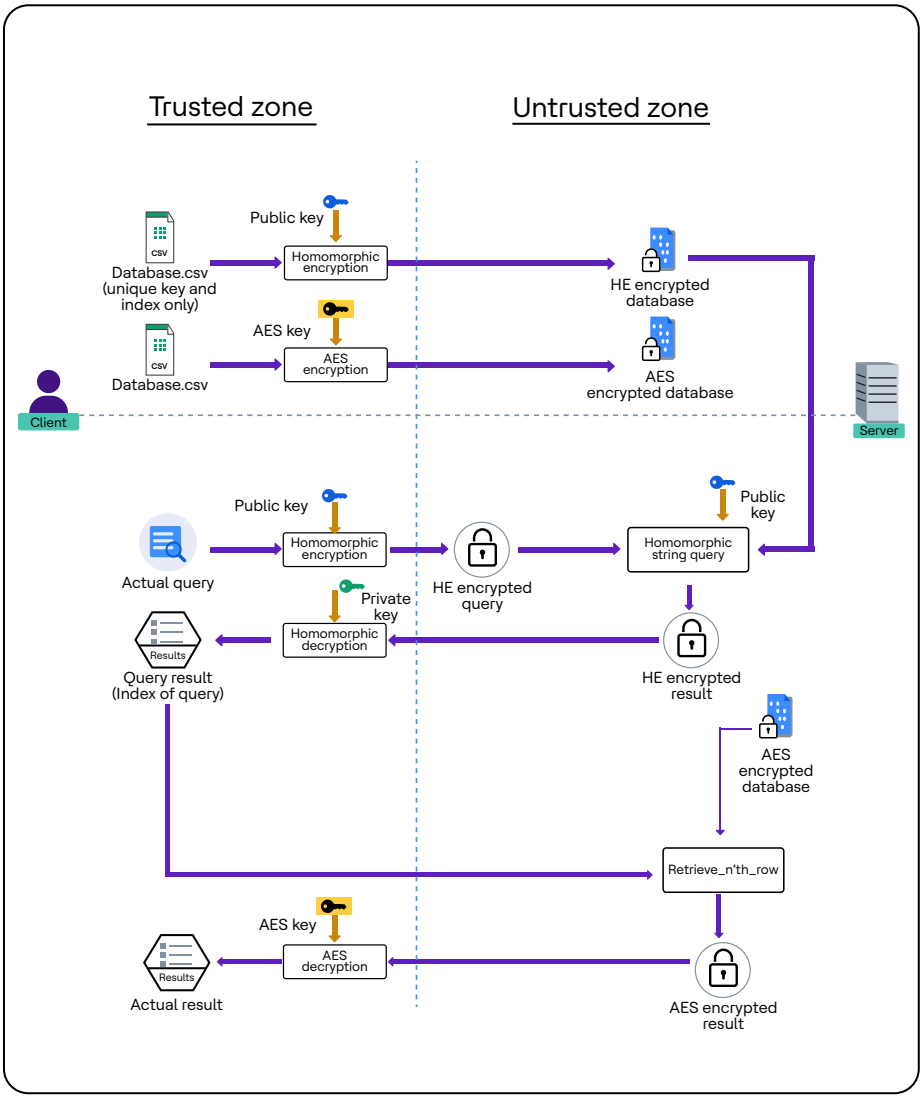


Figure 5 Mixed approach experimental setup

This experiment is implemented using Helib that utilizes 11th Gen Intel(R) Core (TM) i7-1185G7 @ 3.00GHz CPU with 16 GB memory and different size data sets starting from five rows of user data to 50 rows of user data with a unique key. The table below showcases the result of an experiment that includes the time taken to encrypt, decrypt and search data in the setup:

Dataset count	Total Encryption Time	Data query time (ms)	Total Decryption Time	Actual database file size (bytes)	Total Storage for Encryption	Memory Factor
5	129	7257	106	142	379645	2673.56
25	252	28793	195	575	1645236	2861.28
50	397	63980	125	1119	3226994	2883.82
75	464	85902	88	1653	4809042	2909.28
100	330	104190	119	2193	6391182	2914.36
125	760	162411	109	2750	7973121	2899.32
150	621	153734	118	3325	9554991	2873.68
175	763	175036	114	3888	11137280	2864.53
200	782	205486	93	4459	12718037	2852.22
225	1185	283187	107	4992	14301433	2864.87
250	1057	309984	160	5588	15882498	2842.25

Figure 6 Table containing processing time and storage consumption for experiment two

Observation

The following are the highlights of the experiment:

Processing requirements:

By using a hybrid encryption approach, the processing requirements for string search operations are slightly reduced as compared to fully homomorphic encryption.

Storage requirements:

The storage requirements are significantly reduced with the hybrid encryption approach. Only the data indices are stored using homomorphic encryption, leading to smaller ciphertext sizes compared to encrypting the entire dataset. The actual data (blobs) are stored using AES encryption, which is efficient in terms of storage space.

Query response time:

The query response time was reduced, too, but it was a very small factor. There was no significant achievement with respect to time.

Conclusion:

The hybrid encryption approach strikes a balance for storage requirements, even though it's higher than that required for AES encryption. It is imperative to perform more optimization to reduce the query response time and use it in the real world. This application is not scalable with its current performance to the real world. Hence, it is essential to look for accelerators to ensure the security, efficiency and usability of homomorphic encryption.

Section 3: Future of FHE and advancements

Homomorphic encryption holds immense promise for revolutionizing secure computation and data privacy in the digital age. By harnessing the full potential of homomorphic encryption, we can pave the way for a future where privacy-preserving technologies empower individuals and organizations to securely leverage the vast wealth of data at their disposal.

FHE: A paradigm shift

The paradigm shift enabled by FHE will lead to the emergence of innovative use cases and applications across various domains, including secure cloud computing, privacy-preserving analytics, encrypted search, secure multi-party computation and more. FHE has the potential to transform how organizations handle and analyze sensitive data, driving advancements in privacy, security and data-driven decision-making.

From secrecy to computation:

Traditional encryption schemes focus on maintaining the secrecy of data during storage and transmission. FHE extends this notion by enabling computation on encrypted data. Instead of merely protecting data from unauthorized access, FHE enables significant operations and analysis to be performed directly on encrypted data without decryption, opening new possibilities for secure computation.

Privacy-preserving outsourcing:

FHE enables secure outsourcing of computations to untrusted third parties, such as cloud service providers, without compromising data privacy. This paradigm shift allows organizations to leverage external computing resources while maintaining control over sensitive data, addressing concerns about data exposure and unauthorized access.

Shift in trust model:

FHE challenges the traditional trust model by allowing computations to be performed on encrypted data by potentially untrusted entities. Instead of relying solely on the trustworthiness of data processors, FHE shifts the focus to the security properties of the encryption scheme itself, enabling secure computation even in adversarial environments.

Privacy-enhanced collaborative computing:

FHE facilitates secure collaboration and data sharing among multiple parties by enabling computations on encrypted data from different sources. This paradigm shift enables collaborative analysis without sharing raw data, addressing privacy concerns and regulatory requirements while fostering innovation and knowledge sharing across organizational boundaries.

Privacy-first data analytics:

FHE enables privacy-first data analytics by allowing computations to be performed directly on encrypted data. This paradigm shift empowers organizations to extract valuable insights and knowledge from sensitive datasets while preserving individual privacy rights and complying with data protection regulations.

New possibilities for secure ML:

FHE opens new possibilities for secure machine learning by allowing models to be trained on encrypted data without exposing the raw data to the model training process. This paradigm shift enables privacy-preserving machine learning applications in healthcare, finance and other domains where data confidentiality is paramount.

Innovative use cases and applications:

Overall, the advent of FHE represents a fundamental shift in how we approach secure computation and data privacy, unlocking new opportunities for innovation and collaboration while ensuring the confidentiality and integrity of sensitive information in an increasingly interconnected and data-driven world.

FHE Roadblocks:

FHE holds immense promise for enabling secure computation on encrypted data, but it also faces several roadblocks that have hindered its widespread adoption. Some of the key challenges associated with FHE include:

- Computational complexity: FHE schemes typically involve complex mathematical operations, resulting in high computational overhead. Performing homomorphic operations on encrypted data can be orders of magnitude slower than equivalent operations on plaintext data.
- Large ciphertext size: Homomorphic encryption often results in ciphertext expansion, where the size of the encrypted data is much larger than the original plaintext. This increases storage requirements and communication overhead, particularly for large datasets.
- Noise accumulation: FHE schemes are susceptible to noise accumulation during homomorphic operations, which can degrade the quality of the encrypted data and affect the accuracy of computations. Managing and reducing noise while preserving data privacy is a significant challenge.
- Limited functionality: While FHE enables arbitrary computations on encrypted data, there are practical limitations on the types of operations and computations that can be efficiently performed. Certain operations, such as division and comparison, are particularly challenging to support in FHE schemes.
- Key management and distribution: FHE requires the generation and management of encryption keys, including public and private keys, which adds complexity to key management and distribution processes. Securely distributing and protecting keys while ensuring efficient access control is a non-trivial task.
- Standardization and interoperability: Lack of standardized FHE schemes and interoperable implementations hinders collaboration and adoption across different platforms and environments. Standardization efforts are essential for promoting interoperability and ensuring compatibility between FHE implementations.

Current trends and approach

To overcome these hurdles, the industry is actively pursuing various research and development efforts, including:

- Algorithmic improvements: Researchers are continuously developing more efficient algorithms and techniques to reduce the computational complexity of FHE schemes and improve their performance. This includes optimizing homomorphic operations, minimizing noise accumulation and enhancing encryption and decryption algorithms.
- Parameter selection and optimization: Selecting appropriate encryption parameters and optimizing scheme parameters are critical for achieving better performance and security in FHE implementations. Parameter selection techniques and optimization strategies aim to strike a balance between performance, security and functionality.
- Hardware acceleration: Hardware acceleration techniques, such as specialized cryptographic processors (e.g., FPGA, ASIC), are being explored to speed up homomorphic computations and reduce latency. Hardware-based solutions can significantly improve the efficiency of FHE implementations for specific applications and use cases.
- Hybrid encryption approaches: Hybrid encryption approaches, combining FHE with other encryption techniques like symmetric-key encryption (e.g., AES), aim to leverage the benefits of both schemes while mitigating their respective limitations. By encrypting only certain parts of the data using FHE, hybrid approaches reduce computational overhead and storage requirements.
- Standardization efforts: Industry consortia and standardization bodies are working towards developing common standards and protocols for FHE, facilitating interoperability and adoption across different platforms and applications. Standardization efforts promote collaboration, interoperability and compatibility among FHE implementations.

Overall, addressing the roadblocks associated with FHE requires a concerted effort from researchers, industry stakeholders and standardization bodies. FHE is the future. By advancing algorithmic techniques, optimizing parameters, exploring hardware acceleration and promoting standardization, the industry aims to overcome these challenges and unlock the full potential of FHE for secure and privacy-preserving computation on encrypted data. Once these roadblocks are cleared, FHE will disrupt the traditional cryptographic mechanisms.

References

- Gentry, C. (2009). A Fully Homomorphic Encryption Scheme.
- Brakerski, Z., & Vaikuntanathan, V. (2011). Fully Homomorphic Encryption without Bootstrapping.
- Smart, N. P., & Vercauteren, F. (2010). Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes.
- https://github.com/jonaschn/awesome-he?tab=readme-ov-file
- https://csrc.nist.gov/Projects/threshold-cryptography
- Data Protection in Virtual Environments (DPRIVE) Dr. Bryan Jacob
- https://iapp.org/news/a/the-latest-in-homomorphic-encryption-a-game-changer-shaping-up/
- https://fhe.org/conferences/conference-2024/
- https://www.semanticscholar.org/paper/Survey-on-Homomorphic-Encryption-and-Address-of-New-Alharbi-Zamzami/6468cffa6d7a1fba27d4e813a0a22531757d1d8a
- https://research.ibm.com/blog/federated-learning-homomorphic-encryption
- https://en.wikipedia.org/wiki/Homomorphic_encryption
- Homomorphic Encryption Standard by Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, Vinod Vaikuntanathan

Author information

Shivani Agarwal

Shivani Agarwal is a Solution Architect and Embedded Security evangelist working on security architecture in IOT & Embedded products. She has wide experience in Threat Assessment & Modeling, security controls implementations and Risk management. She has more than 18 years of experience in complete SDLC Implementation including Requirement gathering & Risk Assessment, System Design and Threat Modeling, development of secure code, testing and secure deployment. She has worked in various domain including consumer electronics, Medical, Automotive Industrial etc. She is experienced in developing secure solutions using PKI Infrastructure, Secure boot, Secure Execution environment, OS hardening, Network and Communication Security.

Manikandan

Manikandan is a cybersecurity engineer with a strong focus on cryptographic algorithms, including post-quantum and homomorphic encryption algorithms. With expertise in encryption-decryption, MAC, digital signature algorithms, Manikandan brings a deep understanding of secure systems to any project. His hands-on experience in handling various hardware boards allows him to bring a practical approach to implementing secure systems.