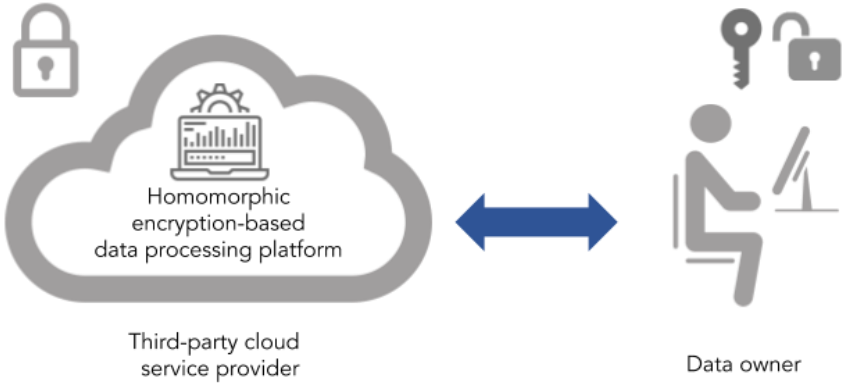
Homomorphic Encryption

Faculty of Mathematics and Computer Science
Babeș-Bolyai University

Team

Petec Răzvan-Gabriel
Stejeroiu Oana-Maria
Zărnescu Bogdan-Gabriel

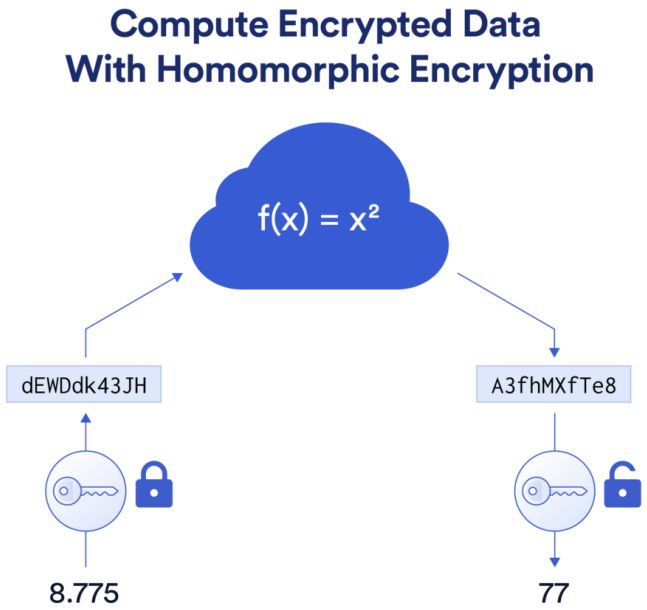
Content



Introduction

Homomorphic Encryption (HE) is a technique that keeps data encrypted during certain computations. This can be useful for:

- Protecting sensitive financial transactions
- Ensuring data privacy in Cloud Computing
- Secure collaboration in Data Analysis
- Preserving privacy in AI/ML
- …



Definition

Two monoids $(M, *)$ and $(N, \bullet)$ are **homomorphic** if there exists $f : M \rightarrow N$ with the property:

$$f(x * y) = f(x) \bullet f(y)$$

for any $x, y \in M$ [2].

Homomorphic encryption is a form of **encryption** with an additional evaluation capability for computing over encrypted data without access to the **secret key** [4].

This technique eliminates the need for processing non-encrypted data, thereby it prevents attacks that would enable an attacker to access that data while it is being processed [4].

Definition

If we define our encryption technique by K (input space), C (encrypted space), $e : K \rightarrow C$ (encryption function), $e^{-1} : C \rightarrow K$ (decryption function), then the function e should be homomorphic between the K and the C for some pre-defined operations (usually addition and / or multiplication).

An example of how this can be used:

1. Input data from K is encrypted: $x_e = e(x)$, $y_e = e(y)$.
2. Some (compatible) operations are performed: $r_e = x_e + y_e + x_e * y_e$.
3. The result is decrypted: $r = e^{-1}(r_e)$.

We can observe that the result is equal to:

$$r = e^{-1}(r_e) = e^{-1}(x_e + y_e + x_e * y_e) = e^{-1}(e(x) + e(y) + e(x) * e(y)) =$$
$$\textit{(from homomorphism properties)} = e^{-1}(e(x + y + x * y)) = x + y + x * y$$

We can observe that the result is the same as the one done if we were doing the operations in the input space, without encrypting the data.

Brief History

The problem of constructing a fully homomorphic encryption scheme (allowing the evaluation of arbitrary operations) was first proposed in 1978, within a year of the RSA scheme publishing [5][4].

During this period, partial results included schemes such as RSA and ElGamal cryptosystems (compatible with multiplications) and Benaloh and Pallier (compatible with additions) [4].

Fully homomorphic encryption schemes first appeared in 2009 with Craig Gentry's groundbreaking lattice-based construction [6], marking a major milestone in cryptography. Since then, researchers have focused on improving the efficiency and practicality of these schemes, significantly reducing computational overhead and making them increasingly viable for real-world applications like secure cloud computing and privacy-preserving AI [4].

Partially Homomorphic Encryption. Definition

Partial Homomorphic Encryption (PHE) enables specific mathematical operations (e.g., addition or multiplication) on ciphertexts, producing encrypted results consistent with operations on plaintexts.

Importance:
Maintains data confidentiality during computations.

Limitations:

- Restricted to a single operation.
- Computational overhead compared to non-homomorphic methods.

Security:
Based on standard cryptographic assumptions like the hardness of factoring or discrete logarithms.

Scalability:
PHE is better suited for applications requiring limited homomorphic operations, reducing computational overhead.

Partially Homomorphic Encryption. RSA

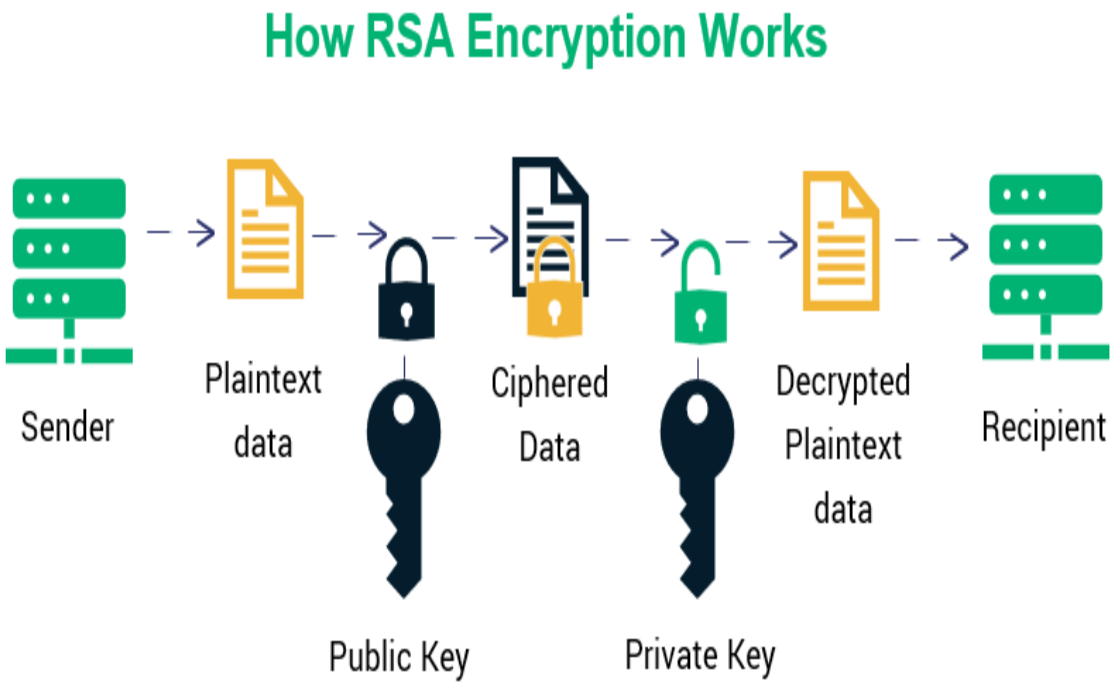
RSA Overview:
A cryptographic algorithm based on the difficulty of factoring large integers. Introduced by Ron Rivest et al. in 1977 [7].

Encryption Function:
Encrypt(x) = m^e (mod n)

Homomorphic Property:
Supports **multiplication** of ciphertexts.

Example:
Encrypt(m1) × Encrypt(m2) = Encrypt(m1 × m2).

Applications:
- Secure digital signatures.
- Confidential data transmission.



Partially Homomorphic Encryption. Paillier

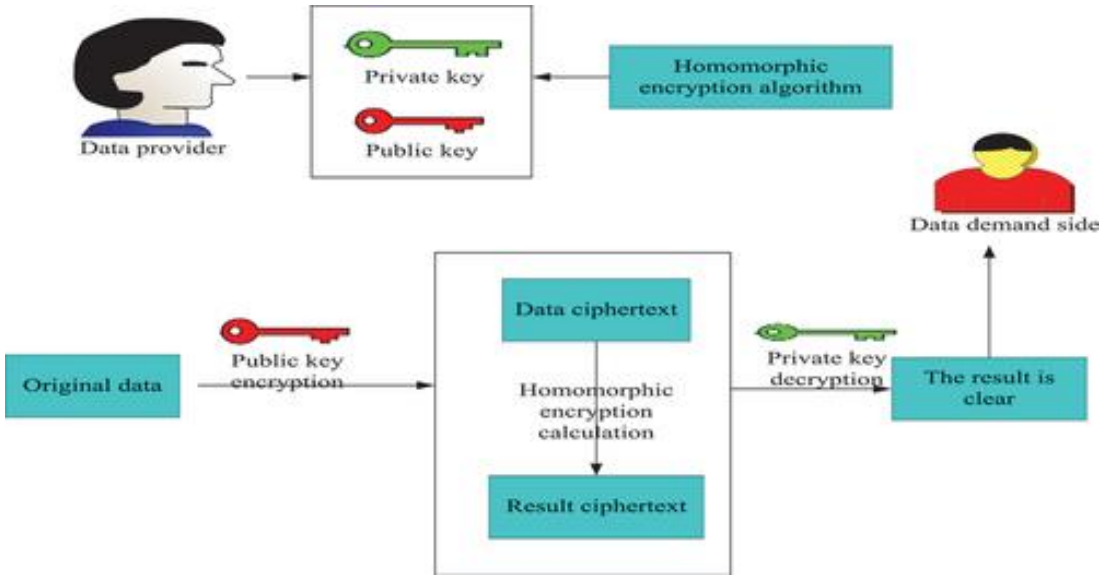
Paillier Overview:
A public-key cryptosystem relying on the difficulty of the composite residuosity class problem. Introduced by Pascal Pallier in 1999 [8].

Encrypt Function:
Encrypt(x) = $g^x \times r^n \pmod{n^2}$

Homomorphic Property:
Supports **addition** of ciphertexts.

Example:
Encrypt(m1) × Encrypt(m2) = Encrypt(m1 + m2).

Applications:
- Privacy-preserving statistics.
- Secure data aggregation in IoT and cloud computing.



Partially Homomorphic Encryption. Comparison

Advantages of RSA:

- High computational efficiency.
- Well-established and widely used.

Disadvantages of RSA:

- Supports only multiplication.
- Susceptible to certain attacks if not implemented securely.

Advantages of Paillier:

- Supports addition, useful for privacy-preserving computations.
- Robust for secure aggregation scenarios.

Disadvantages of Paillier:

- Higher computational cost.
- Larger ciphertext size, leading to storage overhead.

Somewhat Homomorphic Encryption

Definition:

- A cryptographic scheme that supports a limited number of operations (addition and multiplication) on encrypted data due to noise accumulation.

Limitations:

- Noise grows with each operation, restricting the number of computations that can be performed before decryption becomes unreliable.

Examples:

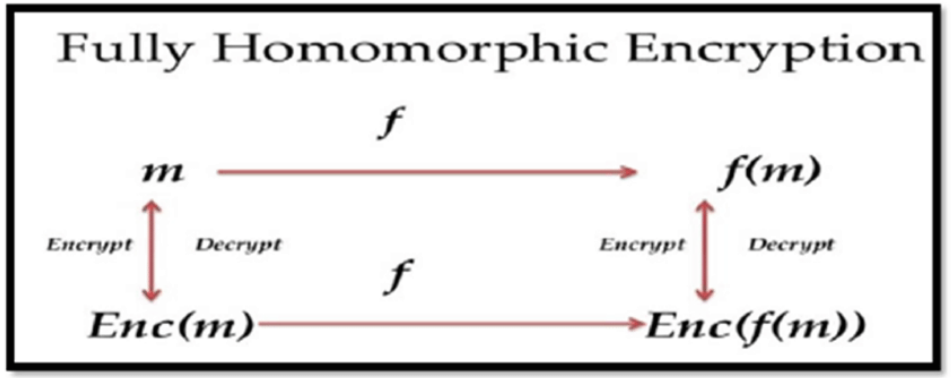
- **BGV Scheme** (Brakerski-Gentry-Vaikuntanathan, 2012 [9]): Early SHE that balances efficiency with noise control.
- **NTRU Cryptosystem** (Lopez-Alt, Tromer, and Vaikuntanathan, 2012 [10]): Lattice-based approach supporting limited homomorphic operations.

Use Cases:

- Suitable for computations requiring only a small number of operations, like simple encrypted analytics or secure multi-party computation.

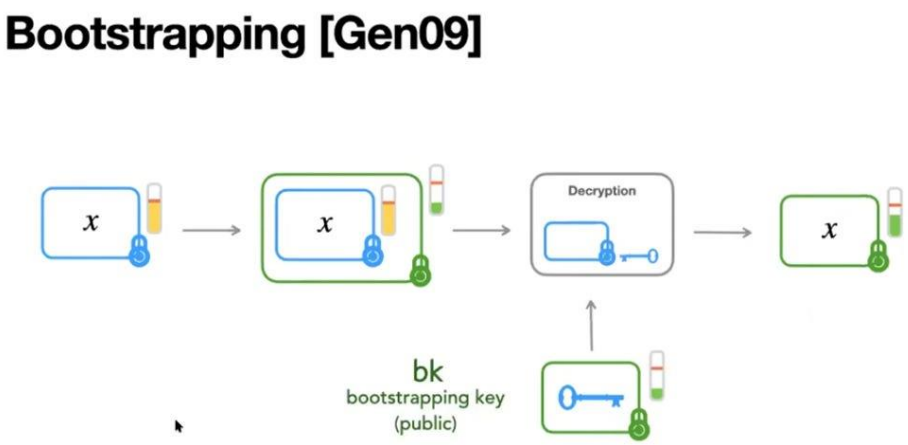
Fully Homomorphic Encryption. Definition

- **Definition:**
  - FHE is an advanced cryptographic technique that allows computations on encrypted data without needing decryption.
  - Ensures data privacy during processing in untrusted environments.
- **Key Features:**
  - Preserves confidentiality throughout computations.
  - Supports both addition and multiplication operations on ciphertexts, enabling arbitrary computations.
- **Example in Practice:**
  - Encrypt(x) + Encrypt(y) → Decrypt(Result) = x + y
  - Encrypt(x) * Encrypt(y) → Decrypt(Result) = x * y



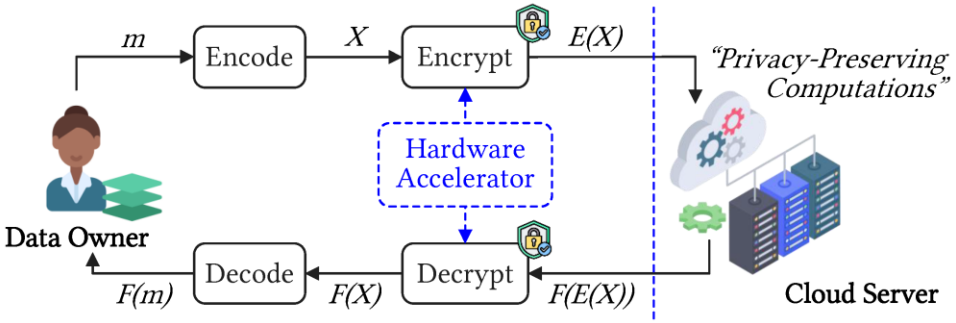
Fully Homomorphic Encryption. Gentry

- **Historical Background:**
  - Craig Gentry, 2009: First practical FHE scheme introduced as part of his PhD thesis.
- **Core Concept:**
  - Utilized a "bootstrapping" technique to reduce noise in ciphertexts during repeated operations.
- **Steps in Gentry's Scheme:**
  - Constructing a "somewhat homomorphic encryption" scheme.
  - Using bootstrapping to achieve fully homomorphic capabilities.
- **Impact:**
  - Marked a revolution in cryptography, inspiring further advancements and practical implementations.



Fully Homomorphic Encryption. CKKS

- **Introduction to CKKS:**
  - Proposed by Cheon, Kim, Kim, and Song in 2017.
  - Specifically designed for approximate arithmetic on encrypted data.
- **Key Features:**
  - Handles real-number operations efficiently.
  - Ideal for applications in machine learning, data science, and financial computations.
- **Advantages:**
  - Supports batched computations for higher efficiency.
  - Balances precision and encryption overhead.
- **Use Case Example:**
  - Securely training machine learning models on encrypted datasets.



Fully Homomorphic Encryption. Comparison

Aspect	Gentry's Scheme	CKKS Scheme
Introduction	The first practical FHE scheme, proposed in 2009	Proposed in 2017 for approximate computations
Methodology	Based on "bootstrapping" to reduce noise	Operates on real numbers with controlled error
Complexity	Computationally intensive, hard to implement practically	More efficient and optimized for modern applications
Applications	Theoretical, for demonstrating the concept	Machine learning, data analysis, finance
Performance	Lower computational rate due to accumulated noise	Higher computational rate, supports batched operations
Data Preservation	Perfect operations with no precision loss	Minor precision loss (approximation)

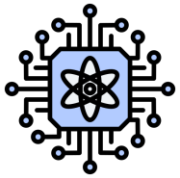
Trade-Offs

Property	Partial Homomorphic Encryption	Somewhat Homomorphic Encryption	Fully Homomorphic Encryption
Operations Supported	Single type (e.g., addition or multiplication)	Addition and multiplication (limited)	Addition and multiplication (unlimited)
Number of Operations	Unlimited for the supported operation	Limited (depends on noise growth)	Unlimited (with noise management)
Noise Accumulation	None (supports one operation only)	Increases with operations	Managed via bootstrapping
Computational Complexity	Low	Moderate	High
Practical Use Cases	Simple tasks like summation or multiplication (e.g., voting systems)	Moderate computations, e.g., small-scale ML or analytics	Complex tasks like privacy-preserving AI and secure cloud computing

Security Implications

Strong Cryptographic Assumptions:

- Security relies on problems like lattice-based cryptography (e.g., Learning With Errors problem) or integer factorization.



Resilience to Quantum Attacks:

- Many HE schemes are designed to resist quantum computing threats, making them future-proof.

Privacy by Design:

- Data remains encrypted throughout processing, reducing the risk of breaches.

Potential Vulnerabilities:

- Noise management is critical; improper handling can lead to security flaws.
- Advances in decryption attacks (e.g., side-channel attacks) require continuous updates.

Conclusions

Homomorphic Encryption's Impact

- Homomorphic encryption is a groundbreaking technique that allows secure computations on encrypted data, addressing critical privacy concerns in modern applications.

Different Levels for Different Needs

- Partial, Somewhat, and Fully Homomorphic Encryption offer varying trade-offs in functionality, efficiency, and complexity, catering to diverse use cases.

Challenges and Innovations

- While Fully Homomorphic Encryption provides unmatched flexibility, ongoing research aims to reduce its computational overhead for broader adoption.

Future Directions

- With advancements in noise management and real-world implementations, homomorphic encryption is poised to revolutionize secure data processing.

Thanks for your attention!

Bibliography

[1] Thaine P. (2020). *Homomorphic encryption for beginners: A practical guide (Part 1)*. Medium.

[2] Modoi G. (2021). *Algebra for Computer Science: Course support*.

[3] Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). *A survey on homomorphic encryption schemes: Theory and implementation*. ACM Computing Surveys (Csur), 51(4), 1-35.

[4] Wikipedia contributors. (n.d.). *Homomorphic encryption*. Wikipedia.

[5] Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). *On data banks and privacy homomorphisms*. Foundations of secure computation, 4(11), 169-180.

[6] Gentry, C. (2009). *Fully homomorphic encryption using ideal lattices*. In Proceedings of the forty-first annual ACM symposium on Theory of computing (pp. 169-178).

[7] Rivest, R. L., Shamir, A., & Adleman, L. M. (1977). *Cryptographic communications system and method* (U.S. Patent No. 4,405,829). U.S. Patent and Trademark Office.

[8] Paillier, P. (1999). *Public-key cryptosystems based on composite degree residuosity classes*. In International conference on the theory and applications of cryptographic techniques (pp. 223-238). Berlin, Heidelberg: Springer Berlin Heidelberg.

[9] Brakerski, Z. (2012). *Fully homomorphic encryption without modulus switching from classical GapSVP*. In Annual cryptology conference (pp. 868-886). Berlin, Heidelberg: Springer Berlin Heidelberg.

[10] López-Alt, A., Tromer, E., & Vaikuntanathan, V. (2012). *On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption*. In Proceedings of the forty-fourth annual ACM symposium on Theory of computing (pp. 1219-1234).