Homomorphic Encryptions

CS6530 - Applied Cryptography Course Project, July - Nov 2025

Sri Sai Shanmukh raj Malireddi
IV year undergrad, Computer Science and Engineering
IIT Madras
CS22B029

Venkateshwara Reddy Nemali
IV year undergrad, Computer Science and Engineering
IIT Madras
CS22B027

I. Introduction

Homomorphic encryption (HE) is a form of encryption that allows computations to be carried out directly on encrypted data without requiring access to the raw, unencrypted information. The result of such computations, when decrypted, matches the outcome as if the operations were performed on the original plaintext. Formally, for an encryption function *Enc* and a decryption function *Dec*, a scheme is homomorphic if for some operation $\oplus$ on plaintexts and a corresponding operation $\boxplus$ on ciphertexts, the following holds:

$$\text{Dec}(\text{Enc}(m_1) \boxplus \text{Enc}(m_2)) = m_1 \oplus m_2.$$

Homomorphic encryption is essential in scenarios where sensitive data needs to be processed while preserving privacy. It enables secure computation over data that cannot be exposed due to regulatory, legal, or ethical constraints. For example, in healthcare, HE allows researchers to perform statistical analysis on encrypted patient records without compromising patient confidentiality. In finance, it enables secure evaluation of credit scores or risk assessments without revealing clients' private data. Additionally, HE finds applications in cloud computing, where users can outsource computation on confidential data while ensuring that the cloud provider cannot access the underlying information.

Overall, homomorphic encryption provides a strong foundation for privacy-preserving computation, making it increasingly relevant in sectors such as healthcare, finance, cloud computing, and any domain where data confidentiality is critical.

II. Homomorphic Protocols

Several cryptographic protocols implement homomorphic properties, each with different strengths, limitations, and use cases. Below, we briefly discuss some widely studied schemes

A. Paillier Cryptosystem

Background and Historical Context

Public-key cryptography traditionally allowed either encryption/decryption or digital signatures but did not directly support computations over encrypted data. Early schemes like RSA offered multiplicative properties, but an efficient additive homomorphic system was lacking. This limitation motivated the search for cryptosystems capable of supporting privacy-preserving computations such as secure voting, private aggregation of sensitive information, and confidential benchmarking of data. Against this backdrop, Paillier proposed a new encryption scheme in 1999 based on the composite residuosity class problem, which allowed secure additions of encrypted integers.

The Paillier cryptosystem was introduced by Pascal Paillier in 1999 at EUROCRYPT [1]. It represents a milestone in public-key cryptography by being one of the first practical additive homomorphic schemes. Its construction relies on the decisional composite residuosity assumption (DCRA), which ensures that distinguishing certain composite residues modulo n^2 is computationally infeasible. Since its publication, Paillier's scheme has become a foundational building block for privacy-preserving systems such as e-voting, private information retrieval, and data aggregation in distributed systems.

Basic Implementation

The Paillier cryptosystem can be implemented in four main stages: key generation, encryption, decryption, and homomorphic addition. The notation used here follows the original scheme:

- **Key Generation**
  1) Choose two large primes p and q, compute $n = pq$ and n^2.
  2) Select $g = n + 1$ (a common choice in practice).
  3) Compute $\lambda = \text{lcm}(p-1, q-1)$.
  4) Compute $\mu = L(g^\lambda \bmod n^2)^{-1} \bmod n$, where $L(x) = \frac{x-1}{n}$.
  5) Public key: (n, g); Private key: (λ, μ).
- **Encryption:** To encrypt a plaintext $m \in \mathbb{Z}_n$, choose a random $r \in \mathbb{Z}_n^*$ and compute:
$$c = g^m \cdot r^n \bmod n^2.$$
- **Decryption:** Given ciphertext c, compute:
$$m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n.$$
- **Homomorphic Addition:** For two ciphertexts c_1 and c_2, we can perform:
$$c_{\text{sum}} = c_1 \cdot c_2 \bmod n^2,$$

which decrypts to $m_1 + m_2 \bmod n$ without revealing m_1 or m_2.

This structure forms the basic implementation of the Paillier cryptosystem and illustrates how secure addition over encrypted integers is achieved. Paillier's additive homomorphism makes it ideal for secure voting, privacy-preserving data aggregation (e.g., summing encrypted sensor data), and financial protocols where summation of private values is required without exposing the underlying data.

B. RSA (Homomorphic Variant)

Background and Historical Context

RSA, introduced by Rivest, Shamir, and Adleman in 1977 [2], is one of the most widely used public-key cryptosystems. Originally designed for secure communication and digital signatures, RSA is based on the mathematical difficulty of factoring large composite integers. Although RSA was not explicitly developed as a homomorphic encryption scheme, it naturally exhibits a *multiplicative* homomorphic property. This means that the product of two ciphertexts corresponds to the encryption of the product of the underlying plaintexts. This property makes RSA suitable for specific privacy-preserving computations, such as delegated multiplications or certain secure protocols, though it lacks additive homomorphism and full homomorphism like Paillier or Gentry's scheme.

Basic Implementation

The homomorphic RSA cryptosystem can be described using four stages similar to Paillier: key generation, encryption, decryption, and homomorphic multiplication.

- **Key Generation**
  1) Choose two large primes p and q, compute $n = pq$.
  2) Compute $\phi(n) = (p-1)(q-1)$.
  3) Select a public exponent e such that $\gcd(e, \phi(n)) = 1$.
  4) Compute the private exponent d such that $ed \equiv 1 \pmod{\phi(n)}$.
  5) Public key: (n, e); Private key: (n, d).
- **Encryption:** To encrypt a plaintext $m \in \mathbb{Z}_n$, compute:

$$c = m^e \bmod n.$$

- **Decryption:** Given ciphertext c, recover the plaintext:

$$m = c^d \bmod n.$$

- **Homomorphic Multiplication:** For two ciphertexts c_1 and c_2, perform:

$$c_{\text{prod}} = c_1 \cdot c_2 \bmod n,$$

which decrypts to $m_1 \cdot m_2 \bmod n$ without revealing m_1 or m_2.

This structure shows how RSA's multiplicative homomorphism can be leveraged. While it does not support additions like Paillier, it remains useful for protocols where multiplying encrypted values is required. Because RSA lacks semantic security under chosen-plaintext attacks when used naively, padding schemes such as OAEP are normally used in practice to strengthen its security.

C. Gentry's Fully Homomorphic Encryption (FHE) Scheme

Background and Historical Context

In 2009, Craig Gentry introduced the first construction of a *fully homomorphic encryption* (FHE) scheme [3]. Before Gentry's work, cryptosystems like Paillier or RSA only supported *partial* homomorphisms (additive or multiplicative). Gentry's breakthrough enabled both addition and multiplication on ciphertexts, allowing arbitrary-depth computations on encrypted data. This capability opened the door to performing secure computations in untrusted environments, such as cloud computing, without ever revealing sensitive plaintexts. Gentry's scheme relies on the hardness of lattice-based problems and introduces the concept of *bootstrapping* to manage the noise that accumulates during homomorphic operations.

Basic Implementation Concept

Although Gentry's FHE is mathematically complex, its core workflow can be described in four conceptual stages:

- **Key Generation:**
  1) Generate a secret key sk for encryption and decryption.
  2) Generate a corresponding public key pk that allows encryption of plaintexts.
  3) Construct additional keys for *bootstrapping*, enabling noise reduction in ciphertexts.
- **Encryption:** Given a plaintext m, compute a ciphertext $c = \text{Enc}_{pk}(m)$ such that decryption with sk recovers m. The encryption includes random noise to ensure semantic security.
- **Homomorphic Operations:** - Addition: $c_{\text{sum}} = c_1 + c_2$ - Multiplication: $c_{\text{prod}} = c_1 \cdot c_2$ Each operation increases the noise in the ciphertext. When noise grows too large, decryption may fail, so *bootstrapping* is applied to reduce noise and refresh ciphertexts.
- **Decryption:** Using the secret key sk, compute $m = \text{Dec}_{sk}(c)$. Bootstrapped ciphertexts maintain correctness, allowing arbitrarily many homomorphic operations.

Gentry's FHE scheme laid the foundation for later practical FHE constructions such as BFV and CKKS. It is particularly useful in privacy-preserving cloud computing, encrypted machine learning, and secure multi-party computation, where arbitrary operations on encrypted data are required without revealing sensitive information. By allowing both addition and multiplication operations on ciphertexts, Gentry's scheme enables the evaluation of any computable function over encrypted inputs, a capability that was previously impossible with traditional partially homomorphic cryptosystems. This property opens the door to fully secure outsourced computations, such as encrypted database queries, confidential medical data analysis, and private financial modeling, without exposing the underlying sensitive data. Furthermore, the

concept of *bootstrapping* introduced by Gentry not only solved the problem of noise growth in ciphertexts but also inspired numerous optimizations in subsequent FHE schemes, making fully homomorphic encryption increasingly practical for real-world applications. Despite the initial computational overhead, Gentry's FHE serves as a theoretical and practical cornerstone for secure computation research, and its principles continue to guide advancements in efficient, privacy-preserving cryptography for cloud services, federated learning, and blockchain-based confidential computation.

D. BFV (Brakerski/Fan-Vercauteren)

Background and Historical Context

The BFV scheme, independently proposed by Brakerski [4] and Fan and Vercauteren [5], is a lattice-based fully homomorphic encryption scheme that improves the efficiency of Gentry's original FHE construction. BFV supports both addition and multiplication on encrypted integers modulo a plaintext modulus, while managing the noise growth efficiently without frequent bootstrapping. It is based on the Ring Learning With Errors (RLWE) problem, which is believed to be hard even for quantum computers. BFV is widely used in privacy-preserving computations, such as secure aggregation, encrypted machine learning, and confidential data analysis in cloud environments.

Basic Implementation Concept

The BFV scheme can be described in four main conceptual stages:

- **Key Generation:**
  1) Generate a secret key sk in a polynomial ring.
  2) Generate a public key pk derived from sk with added noise for semantic security.
  3) Optionally generate evaluation keys for performing homomorphic multiplications efficiently.
- **Encryption:** Encrypt plaintext m (a polynomial) using the public key pk, producing ciphertext c. Random noise is included to ensure semantic security.
- **Homomorphic Operations:** - Addition: $c_{\text{sum}} = c_1 + c_2$ - Multiplication: $c_{\text{prod}} = c_1 \cdot c_2$ Noise increases during operations; BFV manages it using modulus switching to allow multiple operations without bootstrapping.
- **Decryption:** Decrypt ciphertext c using the secret key sk to recover the plaintext m. Proper management of noise ensures correct decryption after several homomorphic operations.

BFV is particularly suited for applications requiring exact arithmetic on encrypted integers, including encrypted voting, privacy-preserving statistics, and secure financial computations. It is also used in secure data aggregation from IoT devices and confidential benchmarking of organizational data, where maintaining exact numerical integrity is crucial. Moreover, BFV supports efficient batch operations using SIMD techniques, allowing multiple encrypted values to be processed simultaneously, which significantly improves computational efficiency in large-scale privacy-preserving systems.

E. CKKS (Cheon-Kim-Kim-Song)

Background and Historical Context

The CKKS scheme, proposed by Cheon, Kim, Kim, and Song in 2017 [6], is a lattice-based homomorphic encryption scheme designed for approximate arithmetic on real or complex numbers. Unlike BFV, which handles exact integers, CKKS allows efficient computation on encrypted floating-point data with controlled approximation errors. It leverages the Ring Learning With Errors (RLWE) problem for security and uses rescaling techniques to manage noise growth during repeated homomorphic multiplications. CKKS is widely applied in privacy-preserving machine learning, encrypted signal processing, and computations on sensitive numerical data in cloud environments.

Basic Implementation Concept

CKKS can be described using four conceptual stages:

- **Key Generation:**
  1) Generate a secret key sk and a corresponding public key pk in a polynomial ring with RLWE noise.
  2) Generate evaluation keys to support homomorphic multiplication and rescaling.
- **Encryption:** Encrypt a plaintext vector of real/complex numbers using pk, producing ciphertext c. The encryption introduces small noise for security.
- **Homomorphic Operations:** - Addition: $c_{\text{sum}} = c_1 + c_2$ - Multiplication: $c_{\text{prod}} = c_1 \cdot c_2$ After multiplications, ciphertexts are rescaled to control noise and maintain correct approximate values.
- **Decryption:** Decrypt ciphertext c with the secret key sk to recover an approximate plaintext vector. CKKS trades exact precision for efficiency, making it suitable for computations where small approximation errors are acceptable.

CKKS has become the standard for encrypted machine learning and data analytics, enabling operations on sensitive real-world data without exposing the raw values.

REFERENCES

[1] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT 1999*, pp. 223–238, Springer, 1999.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC '09: Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pp. 169–178, ACM, 2009.

[4] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology — CRYPTO 2012*, pp. 868–886, Springer, 2012.

[5] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," in *IACR Cryptology ePrint Archive*, pp. 144–157, 2012.

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology — EUROCRYPT 2017*, pp. 409–437, Springer, 2017.