Alternatives for Homomorphic Encryption with Reduced Overheads

Homomorphic encryption (HE) is an advanced cryptographic method that allows computations to be performed on encrypted data without decrypting it first. This feature is extremely useful for maintaining privacy and security in sensitive environments like finance, healthcare, and cloud computing. However, the biggest drawback of fully homomorphic encryption (FHE) is its computational overhead, which makes it impractical for many real-time and large-scale applications. In this report, I have explored several alternative techniques and optimizations designed to reduce the overhead associated with traditional HE. These alternatives balance computational efficiency and security and include methods like batching, approximate HE, differential privacy, and more. I have compared these techniques, analysed their trade-offs, and recommended the most suitable methods based on specific use cases.

Overview of Traditional Homomorphic Encryption (HE)

Traditional HE, especially fully homomorphic encryption (FHE), is extremely secure but computationally expensive. FHE enables arbitrary computations on encrypted data, including both addition and multiplication, which are crucial for complex computations.

The key features of traditional HE:

- **Security**: Data remains encrypted throughout the process, ensuring that sensitive information is never exposed.
- **Versatility**: Both addition and multiplication operations can be performed on encrypted data, allowing for complex computations such as machine learning training or secure data analysis.

Challenges of Traditional HE:

- **High Computational Cost**: Encrypting, decrypting, and performing operations on ciphertexts require significant computational resources, leading to high latency.
- **Memory Overhead**: Ciphertexts in traditional HE schemes are much larger than their plaintext counterparts, leading to increased memory usage.
- **Performance Bottlenecks**: Especially with multiplication, operations on ciphertexts are orders of magnitude slower than similar operations on plaintexts.

Given these challenges, real-world implementations of traditional FHE often require optimizations to make the technology viable for large-scale or real-time applications.

Alternatives and Optimizations to Reduce Overheads

Below are several alternatives to traditional HE that provide varying levels of computational efficiency while maintaining security.

1. Batching with BFV (Microsoft SEAL)

Batching is a method that allows multiple plaintext values to be encrypted into a single ciphertext. By processing these batched ciphertexts, we can perform operations on multiple values simultaneously. This technique drastically reduces the number of encryption and decryption operations, leading to reduced computational and storage overhead. Instead of encrypting each value individually, the BFV scheme in Microsoft SEAL can batch several values into a single ciphertext. Homomorphic operations like addition and multiplication can then be performed on all the values in the batch simultaneously.

Advantages:

- **Reduced Encryption and Decryption Overhead**: By grouping multiple values into a single ciphertext, the number of encryption and decryption operations is drastically reduced.
- **Efficient for Large Datasets**: Batching is especially useful in scenarios where large datasets are processed, as it allows simultaneous operations on multiple data points.

Drawbacks:

- **Complexity in Managing Batch Sizes**: The size of the batch must be carefully managed, and operations need to be performed in a way that ensures each batched value is handled correctly.
- **Not Ideal for Small Datasets**: Batching is less beneficial for applications where individual values need to be processed separately.

Use Case Example: Financial transactions where large numbers of encrypted records must be processed in bulk (e.g., applying tax computations to multiple encrypted records).

2. Approximate Homomorphic Encryption (CKKS)

The CKKS scheme allows homomorphic operations on floating-point numbers, but it sacrifices a small amount of precision for increased computational efficiency. Approximate HE is ideal for applications like machine learning, where slight variations in

numerical precision are tolerable. CKKS allows computations on encrypted floating-point numbers, which is extremely useful in scenarios like financial modelling or machine learning, where exact precision is not critical. CKKS can perform operations on complex numbers, making it suitable for machine learning algorithms that use real numbers.

Advantages:

- **Improved Performance**: Approximate encryption is faster than exact encryption, especially in scenarios where floating-point operations are performed (e.g., matrix multiplication).
- **Suited for Machine Learning**: Approximate values are often good enough for ML algorithms, and CKKS supports matrix operations, which are common in ML.

Drawbacks:

- **Precision Loss**: The primary drawback is that CKKS only supports approximate results, which may not be suitable for applications requiring exact computations.

Use Case Example: Machine learning models where encrypted data is used to train or evaluate models, such as encrypted logistic regression or neural networks.

3. Optimizing Parameters in Homomorphic Encryption

Choosing the right parameters (such as polynomial modulus degree, plaintext modulus, and coefficient modulus) in homomorphic encryption can optimize performance without compromising security. By fine-tuning these parameters, we can reduce encryption time and improve the speed of operations on ciphertexts. Encryption parameters determine the security level and performance trade-offs of homomorphic encryption. A larger polynomial modulus degree increases security but also adds computational complexity. By selecting smaller values for the plaintext modulus and balancing the coefficient modulus, we can tailor the encryption scheme to our specific needs.

Advantages:

- **Reduced Computational Cost**: Optimizing parameters allows for faster encryption and decryption.
- **Maintain Security**: Well-chosen parameters provide a balance between performance and security, ensuring that encryption remains robust while minimizing overhead.

Drawbacks:

- **Requires Expertise**: Finding the right balance requires a deep understanding of cryptography, making it challenging for non-experts.

Use Case Example: Secure cloud computing where data needs to be encrypted for short periods and processed with minimal delays.

4. Partially Homomorphic Encryption (PHE)

Partially Homomorphic Encryption (PHE) supports only one type of operation—either addition or multiplication—but not both. This trade-off significantly reduces computational complexity, making it faster and more efficient than fully homomorphic encryption. PHE is particularly useful in specific domains, such as secure voting systems or digital cash, where only addition or multiplication is required. PHE schemes like the Paillier or RSA schemes allow either addition or multiplication to be performed on ciphertexts, but not both. For example, in a secure voting system, votes can be encrypted and summed up without being decrypted.

Advantages:

- **Faster Than FHE**: Since only one operation is supported, the encryption and decryption processes are much faster than those in FHE.
- **Simpler to Implement**: Because only one type of operation is supported, PHE is simpler and faster to implement than FHE.

Drawbacks:

- **Limited Functionality**: PHE is only useful when a single operation (either addition or multiplication) is needed.

Use Case Example: Secure electronic voting, where only the sum of votes is required, and multiplication operations are unnecessary.

5. Secure Multi-Party Computation (SMPC)

Secure Multi-Party Computation (SMPC) is a cryptographic method that enables multiple parties to jointly compute a function over their inputs while keeping those inputs private. Instead of relying solely on encryption, SMPC splits the computation across multiple parties, who work together to produce a result without revealing their individual data. In SMPC, data is divided into "shares," and each party operates on their own share of the data. After completing their individual computations, the parties combine their results to get the final output.

Advantages:

- **Distributed Computation**: SMPC distributes the computational burden across multiple parties, reducing the load on any one party.
- **High Security**: Even though data is shared among multiple parties, no individual party can learn the data of another.

Drawbacks:

- **Requires Coordination**: SMPC involves multiple parties and requires communication between them, which can introduce delays.
- **Higher Latency**: The need for multiple rounds of communication between parties can lead to higher latency in real-time applications.

Use Case Example: Secure financial auditing, where different parties contribute encrypted data for analysis without revealing sensitive financial information.

6. Differential Privacy

Differential Privacy (DP) ensures that the inclusion or exclusion of a single data point in a dataset does not significantly affect the outcome of a computation. Instead of encrypting data, DP adds noise to computations, making it difficult to trace individual data points back to their original values. In DP, a small amount of random noise is added to the result of computations on the dataset. This ensures that the output does not reveal sensitive information about any individual in the dataset, while still allowing useful aggregate statistics to be extracted.

Advantages:

- **Minimal Computational Overhead**: DP does not involve complex encryption algorithms, making it computationally lightweight.
- **Scalable**: DP scales well to large datasets, as the noise added is proportional to the size of the dataset, not the complexity of the data.

Drawbacks:

- **Inexact Results**: The results of computations using DP are not exact due to the noise that is added, which may be unacceptable in certain applications.

Use Case Example: Large-scale data analysis for healthcare data, where individual privacy is crucial but exact precision is not always required.

7. Functional Encryption

Functional Encryption allows computations to be performed on encrypted data, but only authorized users can learn specific parts of the output. Unlike homomorphic encryption, where any computation can be performed on the ciphertext, functional encryption restricts access to specific functions of the data. Functional encryption allows specific computations to be performed on ciphertexts, and only the result of that computation can be decrypted. For example, a user may be allowed to compute the average of a dataset, but not the individual values.

Advantages:

- **More Control Over Data**: Functional encryption provides fine-grained control over what information can be accessed from the encrypted data.
- **More Efficient Than FHE**: By restricting access to only certain functions, functional encryption is more efficient than fully homomorphic encryption.

Drawbacks:

- **Limited Use Cases**: Functional encryption is only useful when predefined functions can be applied to the data, making it less flexible than other encryption schemes.

Use Case Example: Secure financial reporting where only summary statistics (e.g., averages, totals) can be accessed, without revealing individual transactions.

Comparison of Alternatives

The following table summarizes the key trade-offs between the alternatives:

Technique	Computational Overhead	Accuracy	Security Level	Best Use Case
Batching (BFV)	Low	Exact	High	Large-scale encrypted data processing
Approximate HE (CKKS)	Medium	Approximate	High	Machine learning, financial analytics
Optimized Parameters in HE	Medium	Exact	High	Secure cloud computing, secure messaging
Partially Homomorphic (PHE)	Low	Exact	Medium	Secure voting, electronic cash
Secure Multi-Party Computation	Low	Exact	High	Distributed financial auditing
Differential Privacy (DP)	Very Low	Probabilistic	High	Healthcare analytics, large-scale data analysis
Functional Encryption	Medium	Exact	High	Financial reporting, restricted data access

Selecting the Best Method

- **For Large-Scale Data Operations**: **Batching (BFV)** is the best option because it reduces computational overhead while ensuring that multiple encrypted values can be processed at once.
- **For Machine Learning and Data Analytics**: **Approximate HE (CKKS)** is the best choice, as it allows efficient computation on encrypted floating-point numbers, which is common in machine learning tasks.
- **For Secure Simple Operations**: **Partially Homomorphic Encryption (PHE)** is ideal for applications like voting systems where only one type of operation (e.g., addition) is needed.
- **For Distributed Privacy-Preserving Applications**: **Secure Multi-Party Computation (SMPC)** is the best choice for distributed environments where multiple parties need to compute a function without revealing their data.

Conclusion

While traditional fully homomorphic encryption provides the highest level of security, its computational overhead makes it impractical for many real-world applications. Alternatives like **batching** and **approximate HE** offer more efficient solutions for large-scale encrypted data processing, particularly in applications like machine learning and financial analysis.

For the PTFI project, where scalability and real-time data processing are key, **Batching with BFV** and **Approximate HE (CKKS)** are the recommended choices. These methods provide significant reductions in computational overhead while maintaining strong privacy guarantees. By implementing these alternatives, we can balance computational efficiency and security, making homomorphic encryption more viable.