Comprehensive Resource Usage Analysis of Homomorphic Encryption Libraries in Client-Server Architectures

Vincent Cohadon, Jessy Fallavier, Jiahui Xiang, Osman Salem, Ahmed Mehaoua
Centre Borelli UMR 9010
Université Paris Cité
Paris, France
{firstname.lastname}@u-paris.fr

***Abstract*—Homomorphic encryption enables computations to be performed on encrypted data, ensuring confidentiality throughout the data processing cycle. In this work, we evaluate several Python libraries implementing different protocols such as BFV, CKKS, TFHE, and Paillier. Our main focus is on measuring resource usage (CPU, memory, bandwidth, disk I/O) in a client-server scenario: the client encrypts the data, sends it to the server for processing, and subsequently receives the processed results in encrypted form. We do not benchmark key generation to concentrate on the homomorphic operations themselves. The preliminary results presented here aim to guide the choice of protocols or libraries in real-world contexts, highlighting practical trade-offs to consider.**

***Index Terms*—homomorphic encryption, privacy-preserving computation, performance benchmarking, Paillier, TFHE, BFV, CKKS**

I. Introduction

Homomorphic Encryption (HE) allows arithmetic operations to be carried out on encrypted data, preserving confidentiality from end to end. This cryptographic system proves especially compelling in applications such as healthcare analytics, cloud data protection, or any scenario demanding robust safeguards for sensitive information. However, it often incurs significant computational and memory overhead, deterring broader adoption. In this paper, we propose an evaluation of various Python libraries within a client-server model. The client encrypts data before transmitting it, the server performs remote computations on the ciphertext, and then returns the results, which the client decrypts for verification.

The concept of homomorphic encryption dates back to 1978 when Rivest, Adleman, and Dertouzos introduced the idea of privacy homomorphisms. However, it wasn't until 2009 that Craig Gentry proposed the first fully homomorphic encryption (FHE) scheme based on ideal lattices, capable of evaluating arbitrary circuits of bounded depth. This breakthrough catalyzed substantial research and development in the field, leading to various practical implementations with different capabilities and performance characteristics.

Modern homomorphic encryption schemes can be classified into several categories based on their capabilities. Partially homomorphic encryption (PHE) supports a limited set of operations, such as Paillier (addition) or ElGamal (multiplication). Somewhat homomorphic encryption (SHE) allows both addition and multiplication but only for a limited number of operations before noise growth renders decryption impossible. Fully homomorphic encryption (FHE) schemes like BFV, CKKS, and TFHE support unlimited operations through bootstrapping mechanisms that manage noise growth, though at significant computational cost.

The BFV (Brakerski-Fan-Vercauteren) scheme operates on integers and provides exact arithmetic. It uses polynomial rings modulo both a plain modulus (for the message) and a coefficient modulus (for noise management). The CKKS (Cheon-Kim-Kim-Song) scheme, by contrast, is designed for approximate arithmetic on real numbers, making it particularly suitable for machine learning applications where perfect precision isn't required. The TFHE (Fast Fully Homomorphic Encryption over the Torus) scheme excels at boolean operations and offers relatively fast bootstrapping, enabling efficient evaluation of logical circuits.

Despite theoretical advancements, practical implementations face several challenges. Performance overhead remains substantial—operations that take microseconds on plaintext may require seconds or even minutes on encrypted data. Memory consumption can be prohibitive, with ciphertexts often hundreds or thousands of times larger than the original data. Network bandwidth becomes a bottleneck in distributed scenarios, as encrypted data and evaluation keys must be transferred between parties. Parameter selection involves complex trade-offs between security, performance, and precision that require expert knowledge. Our research aims to provide empirical evidence to guide practical deployment decisions in client-server architectures. We focus specifically on:

- Comprehensive resource profiling across multiple dimensions (CPU, memory, network, disk)
- Comparative analysis of different HE libraries implementing the same protocols
- Evaluation of common arithmetic operations relevant to real-world applications
- Documentation of optimization opportunities specific to client-server scenarios

Unlike previous works that often focus on theoretical aspects or isolated benchmarks, our evaluation models the complete lifecycle of encrypted computation in a distributed setting, more closely approximating real-world usage patterns. By excluding key generation from our measurements, we concentrate on the operational costs that would dominate in production environments where keys are generated infrequently relative to data processing operations.

II. Related Work

The field of homomorphic encryption (HE) has witnessed substantial growth in both theoretical foundations and practical implementations. Acar *et al.* [1] provide a comprehensive survey of HE schemes, detailing their mathematical underpinnings and evolution from Gentry's initial lattice-based construction to more efficient schemes like BGV, BFV, CKKS, and TFHE. They highlight the trade-offs between expressiveness, performance, and ciphertext expansion, noting that practical implementations still face significant overhead compared to plaintext operations, a critical concern for communication-intensive applications.

In application domains, performance trade-offs are paramount, especially for networked systems. Liu *et al.* [7] survey approximate HE approaches, showing how accepting slight deviations from exact results can yield efficiency gains for machine learning and signal processing tasks often deployed over distributed networks, while Munjal and Bhatia [9] focus on healthcare, emphasizing that protecting sensitive medical data in cloud-based telemedicine systems often requires hybrid approaches or hardware acceleration due to computational demands. Similarly, Aziz *et al.* [2] demonstrate how parallelization mitigates performance overhead in genomic string search, underscoring the role of algorithm design for large encrypted datasets transmitted across high-latency networks.

Optimizations of HE computations are crucial for reducing network overhead. Bon *et al.* [3] investigate techniques for homomorphic evaluation of boolean functions, introducing packing methods and batching strategies that achieve substantial performance improvements, potentially minimizing bandwidth usage in client-server interactions. Additionally, Nakashima *et al.* [10] propose homomorphic linear transformations to accelerate bivariate function evaluation, illustrating the impact of domain-specific optimizations on computational efficiency over communication channels.

Deployment in cloud environments, a key area for modern communication networks, presents unique challenges, as examined by Mahato and Chakraborty [8], who analyze how HE schemes integrate with cloud infrastructure, balancing security benefits against resource demands that affect service-level agreements in multi-tenant systems. Complementing this, Onishi *et al.* [11] explore hybrid architectures combining HE with trusted execution environments (TEEs), leveraging TEEs for expensive operations to optimize the security-performance trade-off, though at the cost of increased system complexity in distributed communication setups.

Comparative analyses provide insights for implementation decisions in networked applications. Wiryen *et al.* [14] compare BFV and CKKS in IoT contexts using libraries like TenSeal and Pyfhel, quantifying trade-offs in accuracy, throughput, and resource consumption—key metrics for bandwidth-constrained environments. This aligns with our research, though we expand the scope to include additional schemes and broader resource metrics in client-server architectures, directly addressing communication overhead and latency challenges.

Collectively, these works illustrate the diverse challenges and ongoing developments in HE research relevant to next-generation networks. Our work builds upon these insights by providing a comprehensive resource consumption analysis tailored to client-server architectures, bridging the gap between theoretical capabilities and practical deployment decisions in communication systems.

III. Proposed Approach

A. Our Contribution

This paper makes the following contributions to the field of homomorphic encryption:

- Development of a standardized benchmarking framework for evaluating homomorphic encryption schemes implemented in Python libraries
- Comprehensive performance comparison of four prominent encryption schemes (Paillier, BFV, CKKS, TFHE) across a range of operations and data types
- Detailed analysis of resource consumption (CPU, memory, network, storage) for each scheme during encryption, computation, and decryption phases
- Identification of performance bottlenecks and optimization opportunities for homomorphic operations in client-server architectures
- Practical guidelines for selecting appropriate encryption schemes based on application requirements and computational constraints

B. Methodology

Our evaluation methodology employs a client-server architecture that mirrors real-world deployment scenarios for privacy-preserving computation:

- **Client Component**: Responsible for key generation, data encryption, and result decryption
- **Server Component**: Performs homomorphic operations on encrypted data without access to decryption keys
- **Benchmarking Framework**: Measures and records performance metrics including:
  - Execution time for key generation, encryption, computation, and decryption
  - CPU utilization during each operational phase
  - Memory consumption profiles
  - Network bandwidth requirements for transferring encrypted data
  - Storage requirements for encrypted data and intermediate results

For each encryption scheme, we implement a standardized set of computational tasks including basic arithmetic operations, statistical functions, machine learning primitives, and string processing operations. All experiments are conducted on identical hardware configurations to ensure fair comparison, with multiple runs to account for performance variability.

C. Experiment Algorithm

Algorithm 1 Homomorphic Encryption Benchmarking (Public Context Version)

Require: Dataset D, Encryption Scheme S, Operation Set O
Ensure: Performance Metrics M

```
Generate contexts (public_context, private_context) for scheme S
Measure context generation time t_k
for each data element d_i in D do
    Encrypt d_i using public_context to obtain ciphertext c_i
    Measure encryption time t_{e,i}
end for
for each operation o_j in O do
    Initialize resource monitoring (CPU, memory, network, disk)
    Execute homomorphic operation o_j on the encrypted data
    Measure operation execution time t_{o,j}
    Record resource utilization metrics
end for
for each result ciphertext r_i do
    Decrypt r_i using private_context
    Measure decryption time t_{d,i}
    Verify correctness against the plaintext operation
end for
Aggregate and analyze performance metrics
return M = {t_k, t_e, t_o, t_d, resource metrics}
```

D. Protocols

- **Paillier** is a partially homomorphic encryption scheme supporting additive operations, based on the decisional composite residuosity assumption. It supports addition of encrypted values and multiplication by plain constants, with key sizes of 1024, 2048, and 4096 bits. Our implementation uses the `Python-Paillier` library[1] with optimizations for large integer arithmetic. A key limitation is the inability to perform multiplication between encrypted values.
  Paillier's encryption scheme is secured by the difficulty of factoring large composite numbers. Given two large prime numbers p and q, the modulus is computed as:
  $$n = p \cdot q.$$
  Encryption of a message $m \in \mathbb{Z}_n$ using a public key (n, g) and a random $r \in \mathbb{Z}_n^*$ is performed as follows:
  $$c = g^m \cdot r^n \pmod{n^2}.$$
  The security rests on the hardness of factoring n, computationally infeasible for sufficiently large primes.
- **BFV** (Brakerski/Fan-Vercauteren) is a ring-based somewhat homomorphic encryption scheme supporting addition and multiplication. It relies on the Ring-Learning With Errors (RLWE) problem and requires careful parameter selection (polynomial modulus degree, coefficient modulus, plaintext modulus). Our implementation utilizes the `TenSEAL` library[2] with batch encoding for parallel operations. BFV faces multiplicative depth limitations requiring parameter adjustments for complex computations.
- **CKKS** (Cheon-Kim-Kim-Song) also employs RLWE-based structures but enables approximate arithmetic on encrypted real numbers. It manages precision through rescaling and controls accumulated errors. Our implementation similarly uses the `TenSEAL` library with rescaling operations. A key limitation of CKKS is that approximation errors increase with operational depth.
  Both BFV and CKKS schemes operate on structured polynomial rings, typically defined as:
  $$R_q = \mathbb{Z}_q[x]/(f(x)),$$
  with polynomial modulus $f(x)$ usually selected as:
  $$f(x) = x^N + 1,$$
  where N is a power-of-two and q a large modulus.
- **TFHE** (Torus Fully Homomorphic Encryption) facilitates Boolean circuit evaluation based on the Learning With Errors (LWE) problem and its ring variant. It evaluates arbitrary Boolean circuits gate-by-gate, but incurs high computational costs during bootstrapping operations. Our implementation leverages the `Concrete` library[3] with optimized bootstrapping techniques.
  TFHE computations are defined on the torus, represented as the quotient:
  $$\mathbb{T} = \mathbb{R}/\mathbb{Z}.$$
  This mathematical structure allows efficient Boolean operations through bootstrapping at the expense of increased computational effort.

E. Computational Complexity

The computational complexity varies significantly across encryption schemes, as summarized in Table I.

Notes:

- λ denotes the modulus size in bits (Paillier scheme).
- N denotes the polynomial dimension for (RLWE) schemes (BFV, CKKS).
- n denotes the dimension for (LWE)-based schemes (TFHE).
- TFHE complexity is dominated by bootstrapping.

[1] https://github.com/data61/python-paillier

[2] https://github.com/OpenMined/TenSEAL

[3] https://github.com/zama-ai/concrete

- Paillier complexities are due to modular exponentiation [12]; BFV and CKKS leverage polynomial arithmetic using FFT/NTT methods [6], [4]; TFHE relies on optimized bootstrapping and LWE operations [5].

TABLE I: Complexity Overview for Homomorphic Encryption Schemes

Operation	Paillier	BFV, CKKS	TFHE
Encryption	$O(\lambda^3)$	$O(N \log N)$	$O(n)$
Decryption	$O(\lambda^3)$	$O(N \log N)$	$O(n)$
Addition	$O(\lambda^2)$	$O(N)$	$O(n)$
Multiplication	Not supported	$O(N \log N)$	$O(N \log N)$
Bootstrapping	Not supported	$O(N \log N)$	$O(N \log N)$

IV. Experiment Results

Our experimental evaluation reveals significant performance differences between the four homomorphic encryption schemes.

A. System Specifications

We run our experiments on two virtual machines managed by a Kernel-based Virtual Machine. One machine acts as the client, responsible for generating or receiving the public key, encrypting or decrypting data, and sending requests. The other machine performs the actual arithmetic operations on the encrypted data. Although both virtual machines reside on the same local physical host, separating them still helps capture and assess real network overhead, CPU load, and memory usage under a simple client–server architecture.

The system configuration used in our experiments is summarized in Table II.

TABLE II: System specifications for the virtual machines and physical host

Specification	Details
Operating System (VM)	Debian 12
RAM (VM)	2 GB
CPU (VM)	2 CPUs
Storage (VM)	20 GB SSD
VM Role	Client or Server
Memory Type	LPDDR5X
Performance Core (per core)	Base: 1.4 GHz, Turbo: 4.8 GHz
Efficient Core (per core)	Base: 0.9 GHz, Turbo: 3.8 GHz

B. Benchmark Scenarios

We carried out two benchmark campaigns using a 4096-bit key length and a data range from 0 to 2^7 for TFHE compilation. For the BFV and CKKS schemes, Galois keys were generated. Each benchmark scenario was repeated 10 times.

- **Scenario 1: Partial and Full HE Benchmark**
  Using nb_data 16 ciphertexts, we performed nb_operations 16 additions on encrypted data with the add_encrypted operation. The schemes under test were paillier and bfv. The resulting phase durations for encryption, operation, and decryption are presented in Table III.
- **Scenario 2: FHE Benchmark**
  In this scenario, we performed nb_operations 32 encrypted additions on nb_data 512 ciphertexts using the add_encrypted operation. The schemes compared were bfv, ckks, and tfhe. Key performance metrics, including phase durations, client-side metrics, and server-side metrics, are shown in Tables IV, V, and VI. In addition, detailed Additional RAM and CPU usage for bfv on both client and server are visualized in Figure 1.

C. Performance Measurements

TABLE III: Phase Durations for Scenario 1 (seconds averages)

Phase	Paillier	BFV
Encryption	7.851059	**0.016119**
Operation	0.027003	**0.007654**
Decryption	2.126538	**0.004756**

TABLE IV: Phase Durations for Scenario 2 (seconds averages)

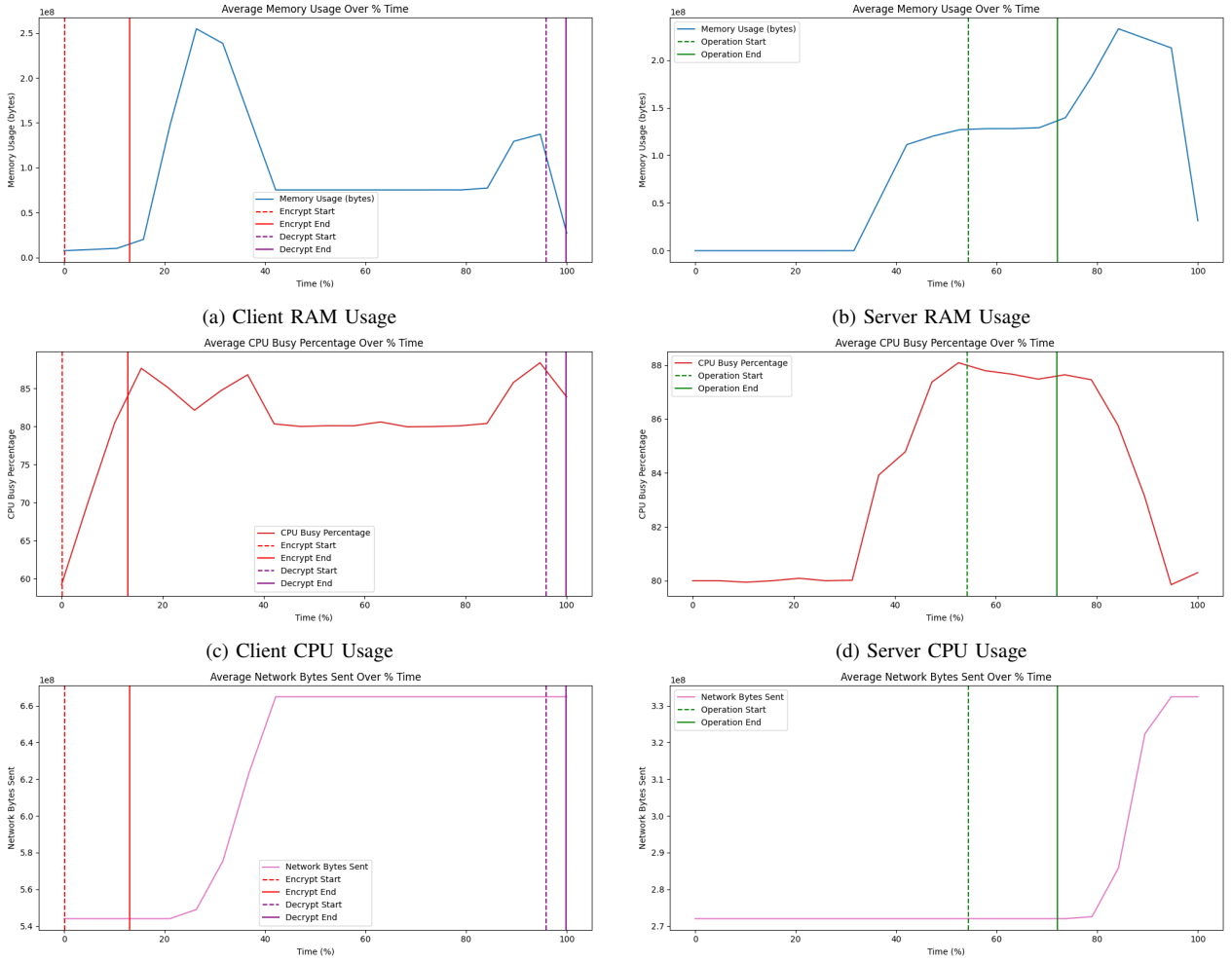
Phase	BFV	CKKS	TFHE
Encryption	**0.437**	0.727	0.646
Operation	**0.609**	0.877	13.958
Decryption	**0.134**	0.200	0.333

TABLE V: Client-Side Metrics for Scenario 2 (averages)

Metric	BFV	CKKS	TFHE
Execution Time (s)	**3.367**	4.526	15.073
Additional Memory Usage (MB)	81.93	68.12	**0.256**
CPU Busy (%)	81.21	80.87	**80.43**
Bandwidth (MB)	879.54	959.33	**50.65**
Bytes Sent (MB)	613.00	669.95	**35.85**
Bytes Received (MB)	266.54	289.38	**14.80**
Latency (ms)	1.137	1.132	0.967
Disk Read (MB)	**0.417**	0.804	1.600
Disk Write (MB)	1.400	7.450	**0.170**

D. Key Findings

1) **Paillier vs. BFV in simple addition workloads** As shown in Table III, BFV drastically outperforms Paillier across all phases for the addition of 16 ciphertexts: encryption (7.851,s vs. 0.016,s), operation (0.027,s vs. 0.008,s), and decryption (2.127,s vs. 0.005,s). These results indicate that BFV's homomorphic additions can be conducted far more efficiently, making it a



(a) Client RAM Usage

(b) Server RAM Usage

(c) Client CPU Usage

(d) Server CPU Usage

(e) Client Sent

(f) Server Sent

Fig. 1: Performance metrics (RAM, CPU, Network Sent) for BFV on Scenario 2 on both client and server. Note that immediately after (resp. before) encryption (resp. decryption), we perform a serialization (resp. deserialization) process to transmit data between client and server.

TABLE VI: Server-Side Metrics for Scenario 2 (averages only)

Metric	BFV	CKKS	TFHE
Execution Time (s)	**3.439**	4.524	15.078
Additional Memory Usage (MB)	69.68	98.44	**0.610**
CPU Busy (%)	**82.48**	83.22	89.28
Bandwidth (MB)	842.68	922.09	**50.56**
Bytes Sent (MB)	269.28	292.00	**14.82**
Bytes Received (MB)	573.40	630.09	**35.74**
Latency (ms)	1.139	1.120	0.967
Disk Read (MB)	**0.000**	0.005	0.540
Disk Write (MB)	**0.018**	1.540	0.150

strong choice for workloads that rely heavily on repeated additions—especially where low-latency encryption/decryption is crucial.

2) **BFV (fastest overall in our measurements).** Across our experiments, BFV delivered the shortest encryption (0.437 s), operation (0.609 s), and decryption (0.134 s) times. Consequently, BFV also exhibited the lowest overall client/server execution times ($\sim$ 3.4 s each). Although BFV's additional memory usage on the client side (81.93 MB) was higher than CKKS (68.12 MB), it was lower on the server side (69.68 MB vs. 98.44 MB for CKKS). BFV's balanced performance profile makes it appealing for many machine-learning inference workloads, moderate-depth encrypted computation, and basic statistical analyses that require both addition and multiplication.
3) **CKKS (slightly slower but supports floating-point).** CKKS took longer in all phases than BFV (e.g., 0.727 s encryption, 0.877 s operation, 0.200 s decryption) and required more server-side memory (98.44 MB). However, it used moderately less client-side memory (68.12 MB)

than BFV and is uniquely suited for approximate floating-point operations. This makes CKKS particularly valuable in applications such as neural network inference, complex statistical analyses, and scenarios where real-number arithmetic at scale is desired—even with some precision loss at larger multiplicative depths.

4) **TFHE (unlimited depth, smallest memory/bandwidth, but slowest).** TFHE's design supports arbitrary Boolean circuits of essentially unlimited depth via frequent bootstrapping. Yet, the measured operation time was substantially higher (13.958 s) than BFV and CKKS, and encryption/decryption were also roughly 1.5–2.5 times slower. Interestingly, TFHE required far less additional memory on both client and server (hundreds of kilobytes vs. tens of megabytes) and consumed only $\sim$ 50 MB of bandwidth (compared to $\sim$ 900 MB for BFV/CKKS). Consequently, TFHE is best suited for workloads requiring complex bitwise or logical operations with unlimited depth, where computational overhead is acceptable but memory/bandwidth usage must remain minimal.

V. Conclusion and Future Work

This paper presented a comprehensive benchmarking framework for evaluating homomorphic encryption schemes, providing practitioners with concrete insights for selecting appropriate techniques based on application-specific requirements. Our comparison of Paillier, BFV, CKKS, and TFHE across various operations and data types revealed significant performance differences, highlighting the trade-offs between computational capabilities and resource requirements.

Building upon this comparative benchmarking study, several avenues for future research emerge:

- **Limitation**: The current implementation supports only integer operations. For example, `Concrete 2.10` uses a subset of Python numerical operators (e.g., __add__), select NumPy functions (e.g., `np.log`), ndarray methods (e.g., `np.ndarray.flatten`), and ndarray properties (e.g., `np.ndarray.shape`). String operations are unsupported. To simulate string concatenation, one can use numeric encoding with the formula:

$$c = a \times 10^{\lfloor \log_{10}(b) \rfloor + 1} + b,$$

  which shifts the digits of a and appends b (after converting characters to ASCII values). This approach is limited to integers, fixed input lengths, and ASCII encoding, making it unsuitable for variable-length strings or non-ASCII characters without further encoding.
- **Approximate vs. Exact Arithmetic**: CKKS employs approximate arithmetic, which may accumulate rounding errors, while BFV offers exact arithmetic for integers. New methods and libraries are emerging to combine the precision of exact arithmetic with the efficiency of approximate methods, potentially enhancing polynomial computations.
- **Communication Bottleneck**: Client-server architectures require encrypted data to undergo serialization prior to transmission and deserialization upon receipt. This introduces additional computational overhead alongside inherent network latency. The issue is further exacerbated with homomorphic encryption schemes such as BFV and CKKS, which involve ciphertexts ranging up to several hundred MB. Consequently, the size of these ciphertexts significantly prolongs transfer times, creating a notable performance bottleneck in practical deployments.

References

[1] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–35, 2018.

[2] M. M. A. Aziz, M. T. M. Tamal, and N. Mohammed, "Secure genomic string search with parallel homomorphic encryption," *Information*, vol. 15, no. 1, p. 40, 2024.

[3] N. Bon, D. Pointcheval, and M. Rivain, "Optimized homomorphic evaluation of Boolean functions," *IACR TCHES*, vol. 2024, no. 3, pp. 302–341, 2024.

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Advances in Cryptology – ASIACRYPT 2017: 23rd Int. Conf. Part I*, 2017, pp. 409–437.

[5] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast Fully Homomorphic Encryption over the Torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[6] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," *Cryptology ePrint Archive*, Report 2012/144, 2012. [Online].

[7] W. Liu, L. You, Y. Shao, X. Shen, G. Hu, J. Shi, and S. Gao, "From accuracy to approximation: A survey on approximate homomorphic encryption and its applications," *Computer Science Review*, vol. 55, 2025, Art. no. 100689.

[8] G. K. Mahato and S. K. Chakraborty, "A comparative review on homomorphic encryption for cloud security," *IETE J. Research*, vol. 69, no. 8, pp. 5124–5133, 2021.

[9] K. Munjal and R. Bhatia, "A systematic review of homomorphic encryption and its contributions in healthcare industry," *Complex Intell. Syst.*, vol. 9, no. 3, pp. 3759–3786, 2023.

[10] A. Nakashima, T. Hayashi, H. Tsuchida, Y. Sugizaki, K. Mori, and T. Nishide, "Faster homomorphic evaluation of arbitrary bivariate integer functions via homomorphic linear transformation," in *Proc. 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2024, pp. 76–86.

[11] R. Onishi, T. Suzuki, S. Sakai, and H. Yamana, "Security and performance-aware cloud computing with homomorphic encryption and trusted execution environment," in *Proc. 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2024, pp. 36–42.

[12] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology—EUROCRYPT '99*, J. Stern, Ed., Lecture Notes in Computer Science, vol. 1592, Springer, Berlin, Heidelberg, 1999, pp. 223–238.

[13] S. Rajashree, B. Vineetha, A. B. Mehta, and P. B. Honnavalli, "Homomorphic encryption approach for string concatenation," in *Proc. 4th Int. Conf. on Cybernetics, Cognition and Machine Learning Applications*, 2022, pp. 267–272.

[14] Y. B. Wiryen, N. W. A. Vigny, M. J. Ngono, and F. L. Aimé, "A comparative study of BFV and CKKS schemes to secure IoT data using TenSeal and Pyfhel homomorphic encryption libraries," *Int. J. of Smart Security Technologies*, vol. 10, no. 1, pp. 1–17, 2024.