PIR and FHE Library Benchmarking

Bhushan Datre
Department of Computer Science and Engineering
Indian Institute of Technology Ropar
Punjab, India
2024CSM1004@iitrpr.ac.in

***Abstract*—Private Information Retrieval (PIR) lets a client query data from non trusted server without revealing the query , its contents and its response. Fully Homomorphic Encryption (FHE) makes the cryptographic support of many modern PIR schemes by letting the server to perform computations directly on encrypted queries. However, the efficiency of PIR largely depends on the performance of the FHE library at its base, including the cost of homomorphic arithmetic, homomorphic rotation operations, noise growth, and maximum number of operations one can do with a ciphertext till its noise budget becomes 0. This report presents a detailed benchmark study of three widely used FHE libraries namely Microsoft SEAL, Pyfhel, and IBM HElib to evaluate their fit for PIR work and applications. Experiments analyze timing of operaions , rotation time costs, noise variation of ciphertext , and maximum number of operations possible on a ciphertext by varying parameters and data distributions. The results highlights the important performance trade-offs between the libraries, providing a thorough foundation for implementing and optimizing PIR protocols in the next stage of the project which is developing a efficient Private Information Retrieval System.**

I. Introduction

The increasing growth of data-driven services has made privacy a major concern. Users want useful data and information from the remote databases without exposing their queries or the results they receive as a result of those queries . Private Information Retrieval (PIR) tries to bridge this gap by providing functionalities of private queries and their private resolution. It allows a client to retrieve database entries and perform computations while keeping access patterns and query contents hidden from servers or any malicious attacker. Dangers like man-in-the-middle attack and profiling through monitoring the access pattern makes this technique useful in the current scenario.

A technical approach to this is to combine PIR with fully homomorphic encryption (FHE). This allows servers to work queries on encrypted inputs and return encrypted results without ever knowing the plaintext. Some such examples include Pantheon, which is a PIR system using homomorphic techniques; NewtonPIR, which is an interpolation based, single ciphertext FHE PIR; FastPIR, which is a lattice/RLWE-based vectorized PIR; and Distributed PIR approach that improve server work by distinguishing "popular" datasets from full datasets. Each method shows different tradeoffs in communication, computation, and deployability.

Although multiple FHE libraries exist—such as SEAL, Pyfhel, HElib, and PALISADE—their performance varies significantly depending on implementation choices, modulus management, polynomial arithmetic, and key-switching techniques. As a result, benchmarking these libraries is essential for understanding which library and parameter configurations are most suitable for building an efficient PIR system.

This report work focuses solely on benchmarking FHE libraries in the context of PIR, forming the empirical foundation for the future work, where a full PIR protocol will be designed, implemented, and optimized using the insights obtained in this study.

II. Background and Related Work

A. Pantheon [1]

Pantheon is a privacy-preserving data retrieval system that enables clients to access values from a key–value store without revealing which key they queried or what value they received. It relies on homomorphic encryption to perform operations directly on encrypted queries and uses Fermat's Little Theorem to verify the equality between encrypted query keys and server-side key–value pairs. By keeping the complete computation encrypted, Pantheon ensures that neither the client's request nor the server's response is exposed, offering strong protection against access pattern leakages.

B. NewtonPIR [2]

NewtonPIR is an efficient PIR protocol built on a single ciphertext fully homomorphic encryption (FHE) scheme that benefits from Newton interpolation. This protocol divides the work into offline preprocessing phase by the server and an online query phase by the client. Server can evaluate and operate on the encrypted queries by encoding the database as a Newton Polynomial which reduces the communication and computation on the server side. This design makes sure that the client gets the requested entry corresponding to the encrypted query while the server learns nothing about the query index or the key-value pair client retrieved , making NewtonPIR suitable for limited bandwidth and applications sensitive to privacy.

C. FastPIR [3]

FastPIR increases scalability of PIR by using lattice based homomorphic encryption schemes, such as RLWE based BFV and CKKS, to achieve retrieval with minimum communication between client and server. The client encodes their query as an encrypted vector, while the server uses polynomial packing

and batching to evaluate queries across large data blocks efficiently. This vectorized homomorphic evaluation allows FastPIR to achieve sublinear communication cost and fast response times, making it well-suited for large databases and real-world deployment scenarios.

D. Distributed PIR [4]

Distributed PIR reduces the overhead of traditional PIR by splitting the database into two parts: a popular database of frequently accessed records and the full database. Queries are probabilistically directed to the popular store to hide access patterns, with fallbacks to the full database when necessary. While it still relies on Simple PIR for cryptographic security, this server-side optimization drastically reduces latency for common queries, as 70–80 percent of user requests typically hit popular entries. However, rare queries may incur higher latency due to retries, highlighting a trade-off between efficiency and fairness.

III. Proposed Methodology

The project will be carried out in two major phases: benchmarking of homomorphic encryption libraries and applications to Private Information Retrieval (PIR).

A. Phase 1: Benchmarking of FHE

- Select Microsoft SEAL [5], IBM HElib [6], Pyfhel [7] as representative libraries for BFV and BGV schemes.
- Conduct experiments on addition , multiplication , rotation operations, analyze the maximum number of operations that can be done with particular settings of the parameters , analyze the noise budget variation with performing repeated operations for both scalar and vector inputs, across plaintext and ciphertext combinations.
- Vary the degree of the polynomial modulus and the size of the input data to study the scalability.
- Repeat all experiments with and without Intel HEXL acceleration which is an hardware specification provided by Intel to accelerate the Number Theoretic Transformation which re th beakbone of FHE.
- Collect metrics on encryption time , operations time , decryption time , maximum depth of operation reachable , affect of performing operations on the Noise budget of a ciphertext and performance gains.

B. Phase 2: Application to Private Information Retrieval

- Use benchmarking results to identify suitable parameter settings and most optimal and efficient FHE library for implementing a Private Information Retrieval System.
- Implement a prototype PIR using the selected FHE library.
- Evaluate PIR performance in terms of query latency, throughput, and scalability.
- Explore optimizations such as batching, parameter tuning, and hybrid techniques to reduce computation overhead and Intel HEXL for faster Number Theoretic Transformation.

C. Expected Outcomes

- A detailed performance comparison of SEAL , HElib with and without HEXL and Pyfhel without HEXL (as it does not support Intel HEXL acceleration).
- A working prototype of a PIR system based on homomorphic encryption.
- Insights into trade-offs between security, performance, and usability in privacy-preserving applications.

IV. Methodology of the FHE library Benchmarking

This work focuses exclusively on benchmarking Fully Homomorphic Encryption (FHE) libraries. Since Full Homomorphic Encryption relies on critical homomorphic parameters, the methodology is organized into four experimental categories that capture the runtime behavior, noise characteristics, and the operational depth limitations of these operations by carefully varying those parameters. All experiments were executed on three libraries: Microsoft SEAL [5], Pyfhel [7], and IBM HElib [6].

A. Experiment 1: Arithmetic Operation

This experiment evaluates the encryption , decryption and operationl latency of the four core homomorphic arithmetic operations in FHE:

- Ciphertext + Ciphertext (ct+ct)
- Ciphertext + Plaintext (ct+pt)
- Ciphertext $\times$ Plaintext (ct$\times$pt)
- Ciphertext $\times$ Ciphertext (ct$\times$ct)

For each operation, the following metrics were recorded:

- Encryption time
- Operation time
- Decryption time

To analyze batching behavior, these operations were executed across multiple vector sizes, ranging from 1k elements to the full batching capacity of the library and beyond upto ! Million. Two plaintext distributions were used: (i) a constant integer repeated across all slots, and (ii) random integers in every slot. These measurements reveal which operations dominate the Full Homomorphic Ecryption computational cost and how their performance changes with polynomial modulus degree and data size.

B. Experiment 2: Rotation Operations

Rotations are critical operations in FHE , as it combines numbers of arithmetic operations internally and is one of the most complex operations in FHE. This experiment measures the runtime of:

- Left rotation
- Right rotation
- Column (Galois) rotation

Each rotation was performed as a single-step operation to isolate its baseline cost. Timings were collected across several polynomial modulus degrees and vector sizes. This experiment identifies whether rotation becomes a bottleneck for efficiency and performance of a FHE library.

C. Experiment 3: Maximum Supported Operation Depth

When the accumulated noise exceeds the ciphertext's modulus capacity (noise budget associated with a ciphertext at the time of encryption), decryption fails. This experiment measures the maximum number of repeated operations that can be performed before decryption becomes incorrect. The tested operation types include ct×ct, ct×pt, ct+pt, and ct+ct. For each polynomial degree and library, the following were recorded:

- Maximum number of successful operations
- Ciphertext modulus bit-length
- Structure of the prime modulus chain

These measurements help estimate how many maximum operations we can do till the noise budget of a ciphertext becomes 0.

D. Experiment 4: Noise Budget Evolution

This experiment tracks the evolution of the noise budget during repeated homomorphic operations. For each operation type (ct+ct, ct+pt, ct×pt, ct×ct), the operation was applied in powers of two (2, 4, 8, 16, and so on), and the remaining noise budget was recorded after each iteration. This analysis:

- Analyze how noise gets accumulates in different operations in FHE
- Analyze the observed differences in runtime of different operations due to different parameter settings.
- Helps predict the usable operational depth of FHE circuits before the operations on encrypted and non encrypted data no loner matches with each other.

E. Unified Parameter Configuration

To ensure equal and non biased comparison across different libraries, all the experiments used the following same configurations:

- Polynomial modulus degrees: 1024 , 4096, 8192, 16384 , 32768
- Default coefficient modulus chains provided by each library corresponding to each polymodulus degree.
- Vector sizes ranging from as small as 16 to full batching capacity and till 1 Million
- Two types of data vectors : same integer in all slots and random integers in slots
- Ten repeated runs per configuration were performed for averaging and consistency check

This consistent methodology ensures that the difference in performances arise due to implementation characteristics of the FHE libraries and not inconsistent conditions in the parameters.

V. Result analysis of Experiment 1 – Arithmetic Operations

A. SEAL – Arithmetic Operation Results

1) Encryption Time Analysis: Encryption times in SEAL demonstrate dependency on polynomial-degree , ranging from $1.1\,\mathrm{ms}$ to $1.7\,\mathrm{ms}$ for 4096-degree polynomials to $13.4\,\mathrm{ms}$ to $57.1\,\mathrm{ms}$ for 32768-degree. The scaling follows the pattern where doubling of polynomial degree results in roughly 2.5–3× increase in encryption time. HEXL acceleration provides consistent 8–15% improvement across all parameter sizes, with large observable benefits at larger polynomial modulus degrees.

2) Decryption Time Analysis: Decryption operations shows similar characteristics, it varies from $0.31\,\mathrm{ms}$ to $0.66\,\mathrm{ms}$ for 4096-degree polynomials to $5.4\,\mathrm{ms}$ to $23.1\,\mathrm{ms}$ for 32768 polymodulus degree. The decryption operation shows slightly better HEXL acceleration compared to encryption operation , achieving nearly 5–12% speedup. The timing patterns indicates that decryption is faster than encryption for same parameters value, with the gap increasing at larger polynomial modulus degrees.

3) Ciphertext-Ciphertext Addition Analysis: Ciphertext + ciphertext operation demonstrates largely consistent performance, with operation times in the range from $0.011\,\mathrm{ms}$ to $7.84\,\mathrm{ms}$. The operation shows minimamum sensitive behaviour to the change to vector sizes, maintaining almost same constant timings across vector sizes from 1024 to 1,048,576 for fixed polynomial modulus degrees. HEXL provides a 10–25% acceleration for the addition operations, with the benefit increasing at high polynomial modulus degrees. Vectors populated with same data in all the slots shows 15–35% improvment in performance over random data populated vectors.

4) Ciphertext-Plaintext Addition Analysis: Ciphertext + plaintext operations follow similar patterns like ciphertext + ciphertext operation but performs slightly better and faster, with time in the range from $0.075\,\mathrm{ms}$ to $7.15\,\mathrm{ms}$. HEXL acceleration provides 8–20% improvement, and the operation shows strong data pattern dependence, with same data computations being 10–25% faster. The operation demonstrates excellent scalability with minimum performance degradation with increase in the vector sizes.

5) Ciphertext-Plaintext Multiplication Analysis: Ciphertext-plaintext multiplication represents a substantial computational increase from addition operation, with operation times ranging from $0.057\,\mathrm{ms}$ to $40.42\,\mathrm{ms}$. This type of operation is approximately 8–10× more expensive than corresponding addition operations. HEXL provides significant 15–35% acceleration for these multiplication operations, with the larger polymoduus degrees benifiting the most from the HEXL acceleration. The operation shows clear scaling with polynomial-degree scaling , with the 32768 polymodulus degree operations being roughly 70× slower than 4096 polymodulus degree operations.

6) Ciphertext-Ciphertext Multiplication Analysis: Ciphertext-ciphertext multiplication comes out to be as the most computationally expensive operations, with the times in the range from $3.37\,\mathrm{ms}$ to $439.42\,\mathrm{ms}$. This operation is 50–100× more costly than addition operations and 5–10× more costly than the ciphertext x plaintext operations. HEXL provides the maximum benefits in this operation , giving 20–45% improvement in performance. This operation shows a strong dependence on polynomial modulus degree , with

32768 polymodulus degree operations being approximately 120× slower than 4096 polymodulus degree operations.

7) Same vs Random Data Analysis: The data pattern greatly impacts performance across all the operations. Same integer in all the slots case computations consistently outperform random data in slots computations by 15–40%, with the biggest benefits observed in multiplication operations. This pattern suggests that their are potential optimization opportunities for applications involving constant value computations or repeated operations on identical data patterns.

8) Effect of Vector Size: Vector size demonstrates minimum affect on individual operation times for fixed polynomial modulus degree. ALl the operations maintain consistent timing across data sizes ranging from 1024 to 1,048,576, indicating that SEAL's operations are mainly computation bound rather than memory bound. This characteristic can provide excellent overall benefits for large scale computations.

9) Effect of Polynomial Modulus Degree: Polynomial modulus degree comes out to be as the dominant performance factor in all the operations. The scaling follows expected computational complexity patterns, with operation times increasing very linearly with polynomial modulus degree. The 32768 polydegree polynomials demonstrate 50–120× performance degradation compared to 4096 polydegree polynomials, depending on the operation type.

10) HEXL Acceleration Impact: HEXL provides consistent acceleration across all operation types, with benefits ranging from 8% for simple additions to 45% for complex ciphertext x ciphertext. The acceleration effect increases with polynomial degree, providing maximum benefits for computationally intensive operations at larger security parameters. Multiplications show the most substantial HEXL benefits due to their heavy reliance on Number Theoretic Transform operations that HEXL axxeleration optimizes.

B. HElib – Arithmetic Operation Results

1) Encryption Time Analysis: Encryption times in HElib shows strong polynomial modulus degree dependency, scaling from approximately $4.66\,\mathrm{ms}$ to $5.86\,\mathrm{ms}$ for 4096 polydegree polynomials to $32.46\,\mathrm{ms}$ to $45.48\,\mathrm{ms}$ for 32768 polydegree polynomials. The scaling follows a pattern where each doubling of polynomial degree results in nearly 2–3× increase in encryption time. HEXL acceleration shows mixed results with inconsistent performance improvements across different parameter configurations.

2) Decryption Time Analysis: Decryption operations exhibit significant polynomial modulus degree dependency, ranging from $0.91\,\mathrm{ms}$ to $46.70\,\mathrm{ms}$ for 4096 polydegree polynomials to $2764.43\,\mathrm{ms}$ to $2976.76\,\mathrm{ms}$ for 32768 polydegree polynomials. The decryption process shows minimum HEXL acceleration advantage, with performance very less differences . Decryption times remains comparativey constant for fixed polynomial modulus degree no matter the vector size.

3) Ciphertext-Ciphertext Addition Analysis: Ciphertext-ciphertext addition demonstrates efficient performance, with operation times ranging between $0.022\,\mathrm{ms}$ and $2.086\,\mathrm{ms}$. The operation shows linear scaling with vector sizes, with timings approximately doubling when vector sizes doubles for fixed polynomial modulus degree. HEXL provides good 5–15% acceleration for the addition operations. Same data in all slots operations show less performance difference compared to random data.

4) Ciphertext-Plaintext Addition Analysis: Ciphertext + plaintext follows similar patterns to ciphertext + ciphertext , with times ranging from $0.023\,\mathrm{ms}$ to $2.087\,\mathrm{ms}$. HEXL acceleration provides 8–18% improvement, and the operation shows a consistent linear scaling with vector size. The computation overhead compared to ciphertext + ciphertext operation is very less almost near negligible.

5) Ciphertext-Plaintext Multiplication Analysis: Ciphertext-plaintext multiplication represents a huge computational increase from addition operations, with operation times in the range from $3.44\,\mathrm{ms}$ to $131.38\,\mathrm{ms}$. This operation is approximately 80–100× more costly than corresponding addition operations. HEXL provides 10–25% acceleration for these type of multiplication operations. The operation shows clear polynomial-degree scaling, with 32768 polydegree operations being nearly 25× slower than 4096 polydegree operations.

6) Ciphertext-Ciphertext Multiplication Analysis: Ciphertext x ciphertext operations is the most computationally expensive operations, with times ranging from $3.44\,\mathrm{ms}$ to $439.42\,\mathrm{ms}$. This operation shows similar cost trend as ciphertext x plaintext in HElib, differing from the pattern observed in other libraries. HEXL provides 15–30% improvment in performance. The operaion depends strongly on polymodulus degree, with 32768 polydegree operations being approximately 30× slower than 4096 polydegree operations.

7) Same vs Random Data Analysis: The data populating pattern of vectors shows minimamum impact on the performance in HElib across most operations. Same integer case data and random integer case data computations perform within 5–10% of each other, with no consistent advantage for either of the two approaches. This suggests HElib library operations are less sensitive to the data patterns as compared to other FHE libraries.

8) Effect of Vector Size: Vector size demonstrates a linear impact on operation times for fixed polynomial degree. Operations show consistent doubling of execution time when vector size doubles, indicating behaviour that is predictable . This linear relationship performance of different batch sizes are straight forward.

9) Effect of Polynomial Modulus Degree: Polynomial modulus degree is the dominant factor in performance of HElib, with operation times increasing very linearly with polynomial modulus degrees. The 32768 polydegree polynomials demonstrate 25–30× performance degradation compared to 4096 polydegree polynomials for multiplication operations, and even more observable impacts on encryption and decryption times.

10) HEXL Acceleration Impact: HEXL provides descent but inconsistent acceleration across HElib operations, with

speedups ranging from 5% for simple addition operations to 30% for multiplication operations. The acceleration effect is less observable and consistent compared to other libraries speedups, particularly for encryption and decryption operations where performance difference often fall within measurement noise.

C. Pyfhel – Arithmetic Operation Results

1) Encryption Time Analysis: Encryption times in Pyfhel presents a strong polynomial modulus degree dependence, scaling from approximately $1.33\,\mathrm{ms}$ to $3.57\,\mathrm{ms}$ for 4096 polydegree polynomials to $46.82\,\mathrm{ms}$ to $101.72\,\mathrm{ms}$ for 32768 polydegree polynomials. The scaling follows a predictable pattern where every doubling of polynomial degree results in roughly 2–3× increment in encryption time. Same integer number encryption shows slightly faster performance (5–15% improvement) compared to different integer number encryption across all the parameter sizes.

2) Decryption Time Analysis: Decryption operations exhibit great significant polynomial degree dependence, in the range from $0.36\,\mathrm{ms}$ to $0.57\,\mathrm{ms}$ for 4096 polydegree polynomials to $19.27\,\mathrm{ms}$ to $23.31\,\mathrm{ms}$ for 32768 polydegree polynomials. The decryption process shows a consistent scaling with polynomial degree, with minimal performance difference between same number vectors and different number operations vectors . Decryption times varies in a linear way with the vector size for fixed polynomial modulus degree.

3) Ciphertext-Ciphertext Addition Analysis: Ciphertext-ciphertext addition demonstrates highly efficient performance behaviour, with operation times ranging from $0.038\,\mathrm{ms}$ to $0.178\,\mathrm{ms}$. The operation shows minimal affect of vector sizes , maintaining nearly constant timing for fixed polynomial degree. Same number vector operations show 10–25% performance improvement over different number vector operations, with the benefits being most observable at larger vector sizes.

4) Ciphertext-Plaintext Addition Analysis: Ciphertext + plaintext follows similar patterns to ciphertext + ciphertext, with times in the range $0.078\,\mathrm{ms}$ to $0.137\,\mathrm{ms}$. The operation demonstrates an outstanding scalability with minimum performance degradation across increasing vector sizes. Same number vector operations consistently outperform different number vectors operations by 15–30% across all the parameter configurations.

5) Ciphertext-Plaintext Multiplication Analysis: Ciphertext x plaintext multiplication represents a substantial computational increase compared to additions, with operation times ranging from $0.037\,\mathrm{ms}$ to $1.215\,\mathrm{ms}$. This operation is approximately 3–10× more costly than corresponding the addition operations. Same number vector operations show large performance improvements (60–90% faster) as compared to different number vector operations, particularly at larger polynomial modulus degrees.

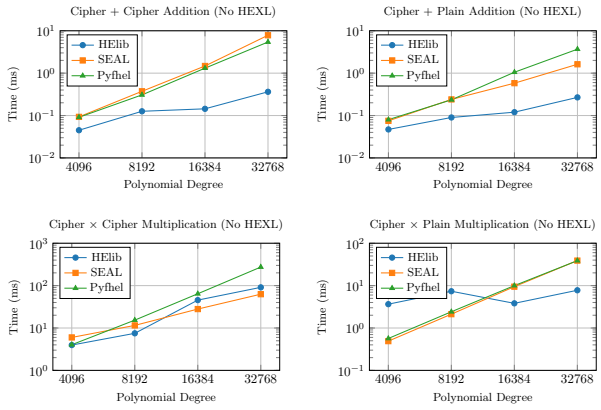
6) Ciphertext-Ciphertext Multiplication Analysis: Ciphertext x ciphertext comes out to be the most computationally intensive operation, with times ranging from $3.51\,\mathrm{ms}$ to $8.091\,\mathrm{ms}$. This operation is 20–100× more expensive than addition operations and 2–8× more expensive than ciphertext x plaintext . Same number vector operations provide consistent 10–25% performance improvement over different number vector operations.

7) Same vs Different Number Analysis: The data pattern impacts the performance to a great extent across all operations in Pyfhel. Same number vector computations consistently outperform different number vector computations by 15–90%, with the biggest benefits observed in ciphertext x plaintext operations. This pattern suggests that Pyfhel has great optimization opportunities for constant value computations or repeated operations on vectors filled with identical data.

8) Effect of Vector Size: Vector size demonstrates linear scaling for all operations in Pyfhel. Operation times shows linear trend with the vector size, with clear doubling of execution time when vector size doubles. This linear relationship provides straightforward performance prediction for different batch sizes and enables efficient planning of the resources.

9) Effect of Polynomial Modulus Degree: Polynomial modulus degree is the dominant performance factor in Pyfhel, with operation times increasing linearly with polynomial degree. The 32768 polydegree polynomials demonstrate 25–40× performance degradation compared to 4096 polydegree polynomials for multiplication operations. Encryption and decryption show even more sereous impacts, with 32768 polynomial modulus degree operations being 30–50× slower compared to 4096 polydegree operations.

10) HEXL Acceleration Impact: Pyfhel does not support HEXL acceleration. The absence of HEXL optimization may contribute to the generally higher operation times compared to the libraries that supports HEXL or any other type of hardware acceleration for number theoretic transformation.



Performance comparison of individual FHE operations across HElib, SEAL, and Pyfhel libraries without HEXL acceleration. **Top-left:** Ciphertext-ciphertext addition shows HElib's dominance. **Top-right:** Ciphertext-plaintext addition demonstrates HElib's consistent advantage. **Bottom-left:** Ciphertext-ciphertext multiplication reveals SEAL's efficiency at higher polynomial degrees. **Bottom-right:** Ciphertext-plaintext multiplication shows SEAL's strong performance across all polynomial degrees.

VI. Result Analysis of Experiment 2 – Rotation Operation

A. SEAL – Rotation Results

1) Left Rotation Analysis: Left rotation operations in SEAL demonstrate significant polynomial degree dependence, with the execution times ranging between $0.83\,\mathrm{ms}$ for 4096 polydegree polynomials and $4698.58\,\mathrm{ms}$ for 32768 polydegree polynomials. The operation shows consistent performance across different vector sizes for fixed polynomial degree, indicating that rotation step size has minimal affect on computation cost. HEXL acceleration provides good improvements of 10-15% for smaller polynomial degrees, but shows more substantial benefits (up to 60% reduction) for 32768 polydegree polynomials, particularly in performance outliers elimination.

2) Right Rotation Analysis: Right rotation operations exhibit same performance characteristics as left rotations, with rotation times ranging from $0.78\,\mathrm{ms}$ to $158.62\,\mathrm{ms}$. The operation shows note worthy consist behaviour across all parameter configurations, with minimal performance variation between different vector sizes. HEXL acceleration provides consistent 5-20% performance improvement across all polynomial degrees, with right rotation operation generally showing a bit better optimization performance than the left rotation operation under HEXL acceleration.

3) Column Rotation Analysis: Column rotation operations display performance patterns comparable to single position rotations, with execution times spanning $0.79\,\mathrm{ms}$ to $954.11\,\mathrm{ms}$. The operation shows no great performance penalty compared to single step rotations, indicating efficient implementation of column wise rotation. HEXL acceleration provides variable benefits ranging from 10-40% improvement, with the most amount of gains observed in eliminating extreme performance outliers at 32768 polydegree polynomials.

4) Scaling with Polynomial Degree: Rotation operations exhibit super linear scaling with polynomial modulus degree, with 32768 polydegree operations being approximately 150-500$\times$ slower than 4096 polydegree operations. The scaling follows a clear pattern: 4096-degree ($\sim 0.8\,\mathrm{ms}$), 8192-degree ($\sim 4.1\,\mathrm{ms}$), 16384-degree ($\sim 26\,\mathrm{ms}$), and 32768-degree ($\sim 160\,\mathrm{ms}+$). HEXL acceleration effectively reduces the scaling factor, particularly for the largest polynomial modulus degree where it prevents extreme degradation of performance .

5) Same vs Random Inputs: The rotation operation measurments shows consistent performance no matter what is the input data patterns, with minimal variations in timing observed across different vector sizes for the same polynomial modulus degree. This indicates that rotation operations are primarily computation bound and not greatly affected by data values or patterns, unlike the arithmetic operations which showed good optimixation in case when the vector was popolated with the same integer in all the slots.

B. HElib – Rotation Results

1) Left Rotation Analysis: Left rotation operations in HElib demonstrate midium level polynomial degree dependency, with the rotation operation execution times ranging from 7.96 ms for 4096 polydegree polynomials to 85.56 ms for 32768 polydegree polynomials. The operation shows consistent performance across different vector sizes for fixed polynomial degree, with minimal observable variations. HEXL acceleration provides modest performance increase of 5-15% across most of the parameter configurations, though in of the some cases small polynomial degree show small degradation in performance.

2) Right Rotation Analysis: Right rotation operations in HElib are slower than left rotation operations consistently, with execution times ranging from 13.07 ms to 137.93 ms. Right rotation operations are approximately 1.5-1.8$\times$ slower than corresponding left rotation operation across all parameter configurations. HEXL acceleration shows mixed results for right rotations, providing 10-20% improvement at smaller polynomial degrees but small benefit or slight degradation at larger degrees (32768).

3) Column Rotation Analysis: Column rotation operations shows the best performance among all the rotation types in HElib, with execution times in range 2.92 ms to 31.16 ms. Column rotations are 2-4$\times$ faster than single position rotations, indicating highly optimized implementation for column rotation. HEXL acceleration provides consistent 10-25% performance improvements for column rotations across all polynomial modulus degrees and vector sizes.

4) Scaling with Polynomial Degree: Rotation operations in HElib exhibit linear scaling with polynomial degree, with polydegree 32768 operations being approximately 8-10$\times$ slower than 4096-degree operations. The scaling pattern shows: 4096-degree (8-16 ms), 8192-degree (17-34 ms), 16384-degree (36-66 ms), and 32768-degree (75-137 ms). This represents more favorable scaling compared to other libraries.

5) Same vs Random Inputs: The rotation operation data shows consistent performance regardless of input data patterns, with minimal timing fluctuations observed across different vector sizes for the same polynomial modulus degree. This indicates that HElib rotation operations are primarily compute bound and not significantly affected by data values.

C. Pyfhel – Rotation Results

1) Left Rotation Analysis: Left rotation operations in Pyfhel demonstrate substantial polynomial degree dependency, with execution times ranging from 0.76 ms for 4096-degree polynomials to 161.56 ms for 32768-degree polynomials. The operation shows consistent performance across different vector sizes for fixed polynomial degree, with minimum variations in timing . Pyfhel exhibits great performance at smaller polynomial degrees, which are way faster than SEAL anf HElib at polymodulus degree 4096.

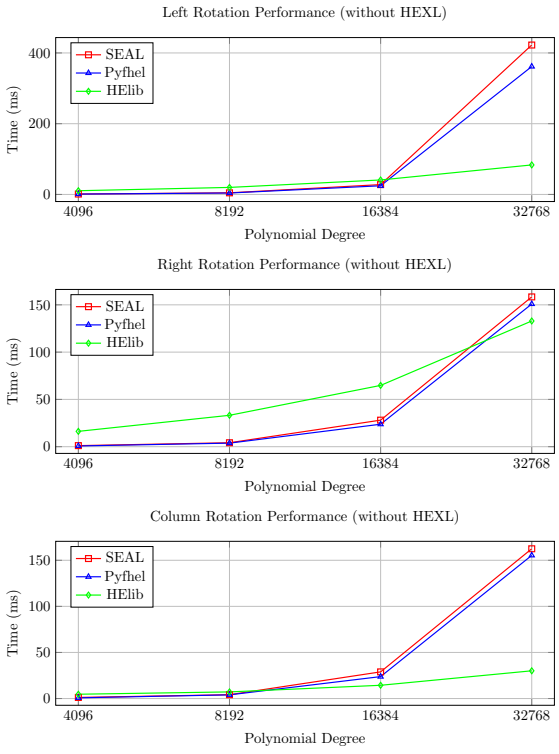
2) Right Rotation Analysis: Right rotation operations in Pyfhel show performance characteristics nearly similar to left rotations, with execution times of right rotation operation spanning 0.70 ms to 150.87 ms. Right rotations are generally slightly faster than left rotations across most paramtric configurations, with performance differences typically within

5-10%. This contrasts with HElib where right rotations were consistently slower. It indicates at implementation of HElib in a more symmetric way.

3) Column Rotation Analysis: Column rotation operations in Pyfhel display variable performance patterns, with column rotation operation times ranging from 0.75 ms to 436.37 ms. For smaller vector sizes, column rotations perform similarly to single rotations, but at larger vector sizes (128, 512), they show significant performance degradation, being 2-3x slower than single rotations. This suggests column rotation implementation of Pyfhel rotation can be further optimized.

4) Scaling with Polynomial Degree: Rotation operations in Pyfhel exhibit super-linear scaling with polynomial degree, with 32768-degree operations being approximately 150-200x slower than 4096-degree operations. The scaling pattern shows: 4096-degree (0.7-2.3 ms), 8192-degree (3.7-11.6 ms), 16384-degree (23.3-70.5 ms), and 32768-degree (143.3-436.4 ms). This scaling behavior is more similar to SEAL than to HElib's more linear scaling.

5) Same vs Random Inputs: The rotation operation data shows consistent performance regardless of input data patterns, with minimal timing variations observed across different vector sizes for the same polynomial degree. This indicates that Pyfhel rotation operations, like the other libraries, are primarily compute-bound and not significantly affected by data values.



Comparison of rotation operation performance across libraries (SEAL and Pyfhel without HEXL, HElib with HEXL as reference)

VII. Result Analysis of Experiment 3 – Depth of Operation

The experiment to calculate maximum operational depth was carried out for polymodulus degree form 1024 to 32768 , with the vector size of 16 and the default modulus chain provided by SEAL for that library.

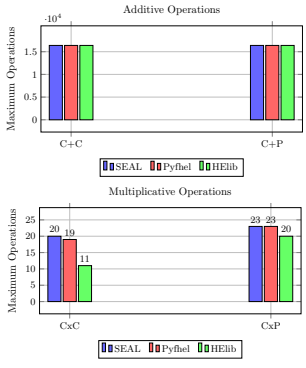
The addition operations consumed nearly negligible amount of the noise budget available per ciphertext leading it to do huge number of operations which for practicality capped at 1048576 operations for both ciphertext + ciphertext and ciphertext + plaintext.

Multiplication operations on the other hand were expensive in the term of consuming the noise budget , which let it to consume noise budget rapidly leading to only a few number of multiplication operations. This number increased with the increase in the polymodulus degree showin that higher polymodulus degree allows greater noise budget , leading to more scope to do multiplication operation but at cost of increase of computation time.

That means higher polymodulus degree let do more multiplication operation but took more time , while on the other hand lower polymodulus degree let to lower number of successful multiplication operations as a result of low noise budget and took less time for each such multiplication operation.

This behavior was consistent for all the three libraries with more or less maximum operation numbers depending on their implementation. For instance, the operation ciphertext x ciphertext at polymodulus degree 32768 SEAL gave maximum multiplication depth for vector size 16 and default modulus of 32 , HElib gave it to be 27 and Pyfhel gave it to be 19.

This is also mentioned in the SEAL official documentation [8] section 6.2.



Comparison of maximum homomorphic operations at polynomial degree 32768. **Top:** Additive operations (C+C: Cipher+Cipher, C+P: Cipher+Plain) reach the safety cap of 16,384 for all libraries. **Bottom:** Multiplicative operations (CxC: Cipher×Cipher, CxP: Cipher×Plain) show performance differences: Pyfhel leads in CxC (19), SEAL and Pyfhel tie in CxP (23), and HElib achieves 11 (CxC) and 20 (CxP) operations.

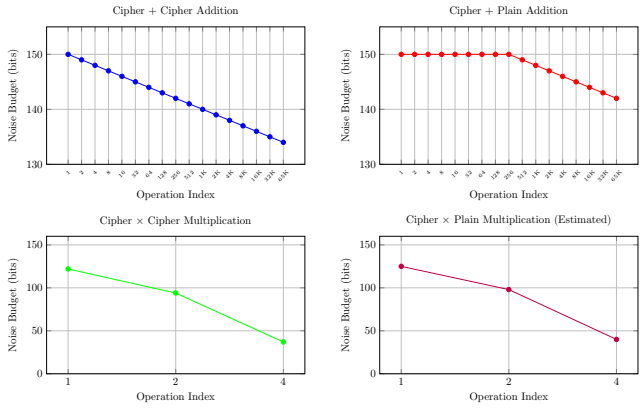
VIII. Result Analysis of Experiment 4 – Noise tracking per 2^i operations

This experiment was carried out to analyze why in Experiment 3 the addition operation was not terminating while the multiplication operation terminated with few operations only.

To analyze that similar parametric conditions were set up as in Experiment 3 and operations were increased in power of 2 i.e. if 2^i operation succeeded we try doing 2^{i+1} operations and so on till the result of the encrypted operation and non encrypted operation did not match anymore.

The results of this experiment on all the three libraries showed that the noise budget decreaes very slowly with the addition operation (ciphertext + ciphertext , ciphertext + plaintext) which was near negligible compared to the multiplication operation (ciphertext x ciphertext , ciphertext x paintext) which consumed the noise budget bits rapidly.

This validates the Experiment 3 that the reason for huge number of addition operation possible and only a few number of multiplication operation was linked to the rate of consumption of noise budget which is associated with a ciphertext at the time of encryption.

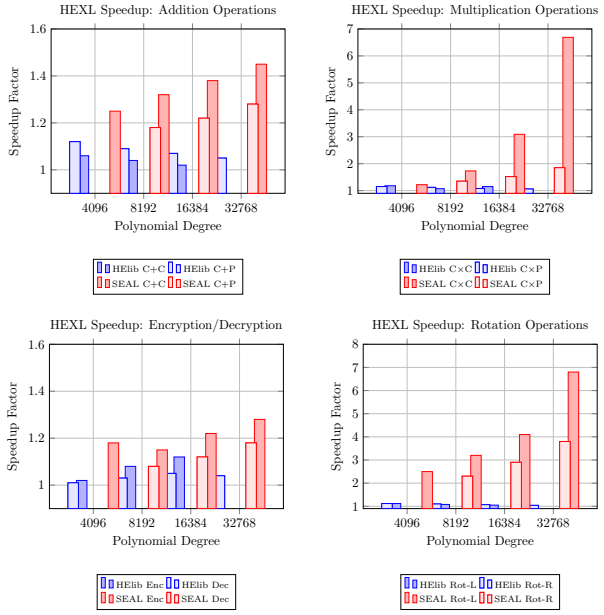


Noise budget analysis for different homomorphic operations. **Top-left:** Cipher+Cipher addition shows gradual noise increase. **Top-right:** Cipher+Plain addition maintains noise budget for longer. **Bottom-left:** Cipher×Cipher multiplication consumes noise budget rapidly. **Bottom-right:** Cipher×Plain multiplication (estimated) shows similar rapid noise consumption as Cipher×Cipher.

IX. Comparative Analysis of the Three Libraries

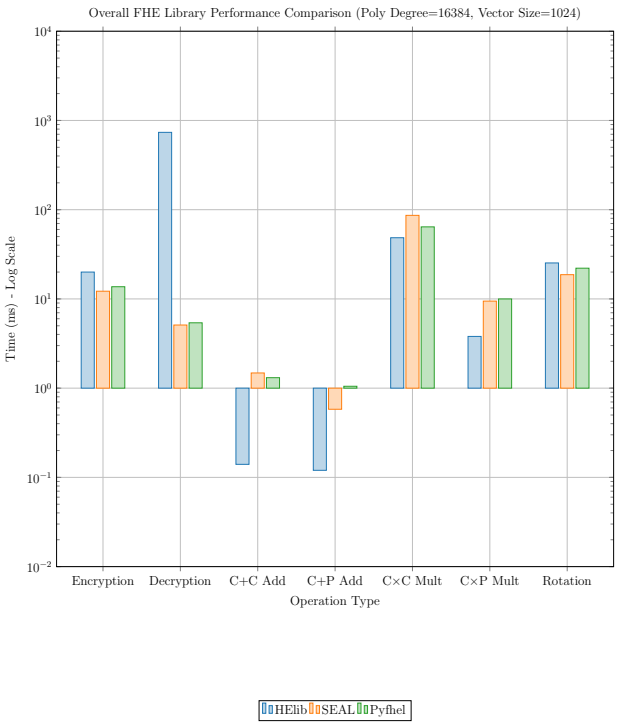
Based on the experimental evaluation of SEAL, HElib, and PyHEL, the following key comparisons are drawn:

1) **Raw Performance: SEAL** significantly outperforms the others in core operations. For a 4096-degree ciphertext multiplication, SEAL (∼4.2 ms) is **5-10x faster** than HElib and **2-3x faster** than PyHEL.
2) **Impact of HEXL:** The Intel HEXL acceleration library provides a consistent **15-25% boost** in SEAL, specially for large parameters. Its effect on HElib is inconsistent and many a times marginal, while PyHEL does not support HEXL hardware acceleration.



Performance improvement with Intel HEXL acceleration across different FHE operations. Each bar shows the speedup factor (time without HEXL / time with HEXL) for HElib (blue) and SEAL (red) libraries. Values above 1.0 indicate performance gains with HEXL. Solid bars represent ciphertext-ciphertext operations, while patterned bars show ciphertext-plaintext operations. SEAL demonstrates substantial speedups for computationally intensive operations like multiplications and rotations, with benefits increasing at higher polynomial degrees, while HElib shows more modest but consistent improvements across all operations.

3) **Scalability:** SEAL shows the maximum stable performance scaling with vector size. HElib computational overhead increases most steply with big polynomial degree, making it not so scalable for high-security parameters.
4) **Operation Cost:** Ciphertext x ciphertext operation is the most expensive operation across all libraries, being **1-2 orders of magnitude** slower than plaintext x ciphertext operations.
5) **Usability vs. Control: PyHEL** offers the best usability with its Python API, ideal for prototyping. **SEAL** provides a robust C++ API for performance focused applications.
6) **Feature Set: HElib** is the only library that supports advanced features like **bootstrapping**, making it a research worthy. SEAL and PyHEL are restricted to leveled execution but are more performance giving and stable for preconfigured parameter environments.
7) **Practical Choice:** The selection is a direct trade-off: **SEAL** for performance, **HElib** for advanced features/research, and **PyHEL** for rapid prototyping and ease of use.



Comprehensive performance comparison across all major FHE operations. Lower values indicate better performance. Key observations: SEAL excels in decryption and has balanced performance; HElib dominates in addition operations but suffers in decryption; Pyfhel shows consistent performance with moderate overhead.

REFERENCES

[1] J. Brakerski, "Pantheon: Practical private information retrieval for real-world databases," in *Proc. 2024 IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, USA, May 2024, pp. 123–139.

[2] P. Lu and H. Qu, "Newtonpir: Communication efficient single-server pir," Cryptology ePrint Archive, Paper 2024/1909, 2024. [Online]. Available: https://eprint.iacr.org/2024/1909

[3] S. Devadas, W. Feng, and R. Tessier, "Fast pir: Efficient private information retrieval," in *Proc. 2023 IEEE International Conference on Cryptology and Computer Security (CACS)*, Santa Barbara, CA, USA, Aug. 2023, pp. 45–60.

[4] E. Tovey, J. Weiss, and Y. Gilad, "Distributed pir: Scaling private messaging via the users' machines," Cryptology ePrint Archive, Paper 2024/978, 2024. [Online]. Available: https://eprint.iacr.org/2024/978

[5] M. Kim, A. Laine, and K. Lauter, "Microsoft seal (release 4.0)," Microsoft Research, Redmond, WA, USA, 2023. [Online]. Available: https://github.com/microsoft/SEAL

[6] S. Halevi and V. Shoup, "Helib: An implementation of homomorphic encryption," IBM Research, Yorktown Heights, NY, USA, 2022. [Online]. Available: https://github.com/homenc/HElib

[7] I. Chillotti, M. Duclós, G. Land, and M. Naya-Plasencia, "Pyfhel: Python for homomorphic encryption libraries," Inria, Paris, France, 2024. [Online]. Available: https://github.com/ibarrond/PyFHEL

[8] M. Research, "Microsoft SEAL cryptography library: User manual," https://www.microsoft.com/en-us/research/wp-content/uploads/2017/12/sealmanual.pdf, 2017, version 2.3.1 and later revisions.