KU LEUVEN

FACULTY OF SCIENCE

Privacy Preserving Systems using FHE-friendly Ciphers

Yarne Thijs

Supervisor: Prof. Dr. Ir. Vincent Rijmen
Mentor: Betul Askin Ozdemir
Mentor: Ir. Dilara Toprakhisar

Thesis presented in
fulfillment of the requirements
for the degree of Master of Science
in Applied Mathematics

Academic year 2024-2025

Foreword

This thesis was written as part of the Master's program in Applied Mathematics at KU Leuven and reflects my work and research conducted during the academic year 2024-2025. It investigates privacy-preserving computation through the lens of homomorphic encryption and transciphering, combining theoretical analysis with practical implementation and benchmarking.

The purpose of this thesis is to investigate and evaluate privacy-preserving computation systems, with a specific focus on transciphering, an approach that combines symmetric encryption with fully homomorphic encryption to improve scalability and efficiency. By integrating theoretical cryptographic primitives with practical benchmarking, the work aims to contribute to the development of secure data processing architectures suitable for sensitive, distributed environments.

I would like to express my sincere gratitude to my supervisors, Betul Askin Ozdemir and Ir. Dilara Toprakhisar, for their insightful guidance and continuous support throughout this project. I also thank Prof. Dr. Ir. Vincent Rijmen for their advice and encouragement. Appreciation goes to my family, friends, and fellow students who provided helpful feedback and shared their perspectives together with their support. Lastly, I thank the developers and maintainers of the OpenFHE and Hybrid-HE libraries, whose work forms the foundation of the experimental components of this thesis.

Abstract

In privacy-preserving computation systems, Fully Homomorphic Encryption (FHE) offers strong confidentiality guarantees by enabling computation on encrypted data. However, FHE's transfer size expansion overhead presents a major bottleneck to real-world deployment. This thesis explores a hybrid cryptographic architecture that leverages FHE-friendly block ciphers (e.g., Pasta) within a transciphering framework. The goal is to reduce ciphertext size which needs to be transmitted while maintaining strong security guarantees.

The core contribution of this work is the design, implementation, and benchmarking of a privacy-preserving pipeline in which each Data Owner (DO) encrypts their private input using a lightweight FHE-friendly symmetric block cipher. This intermediate ciphertext is then transformed homomorphically by the Server (S), a semi-trusted entity responsible for managing encrypted computations. The Data Customer (DC), who provides the computation function and ultimately receives the result, holds the FHE secret key and can decrypt the final output. This model achieves end-to-end confidentiality without requiring DOs to perform computationally expensive FHE operations themselves, nor allowing DCs to read the secret data of the DOs.

We explore both standard FHE and transciphered pipelines to yield interesting results and comparisons. It is demonstrated that transciphering significantly inhibits ciphertext expansion at the cost of time complexity while preserving correctness and confidentiality. Additional protocols, such as thresholding of participation and algorithmic controls like choice vectors and dummy value insertion, could be used to prevent reconstruction attacks and enhance access control.

This work aspires to contribute to the development of scalable, efficient, and secure privacy-preserving systems, with applications in cloud computing and secure machine learning. By providing practical insights and implementation guidance, this paper aims to support the broader adoption of transciphering and FHE in real-world scenarios. This work highlights the practical trade-offs between efficiency, security, and complexity in multi-party privacy-preserving systems, and shows that transciphering with FHE-friendly ciphers presents a viable and scalable strategy for secure cloud-based analytics.

List of Abbreviations

HE	Homomorphic Encryption
FHE	Fully Homomorphic Encryption
HHE	Hybrid Homomorphic Encryption
DO	Data Owner
S	Server/Aggregator
DC	Data Customer
CV	Choice Vector
BFV	Brakerski/Fan-Vercauteren encryption scheme
BGV	Brakerski-Gentry-Vaikuntanathan encryption scheme
CKKS	Cheon-Kim-Kim-Song approximate FHE scheme
AES	Advanced Encryption Standard
PK	Public Key
SK	Secret Key
RAM	Random Access Memory
CPU	Central Processing Unit
I/O	Input/Output
SIMD	Single Instruction Multiple Data
MPC	Multi-Party Computation
SIMD	Single Instruction, Multiple Data
IoT	Internet of Things
RNG	Random Number Generator

List of Symbols

XOR	Exclusive OR
XOF	Extendable-output function

$\mathbb{Z}$	Ring of integers
$\mathbb{Z}_m$	Quotient ring $\mathbb{Z}/m\mathbb{Z}$, ring of integers modulo m, m integer
$\mathbb{A} = \mathbb{Z}[X]/\phi(X)$	Polynomial ring defined by $\phi(X)$
$\mathbb{F}_{p^n}$	The finite field of order p^n
$\lvert G \rvert$	The order of the group G

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The field of cryptography has evolved substantially, addressing the ever-growing challenges of securing communication and data in a digitally interconnected world. From classical encryption techniques to the advanced primitives of modern cryptography, from block ciphers to public-key encryption, the discipline has progressed to meet the demands of security, efficiency, and scalability. The emergence of cloud computing, distributed systems, and big data processing has heightened the importance of privacy-preserving mechanisms, which ensure that sensitive information is protected even in untrusted environments.

Privacy-preserving systems have gained prominence as a response to the dual challenges of data security and functionality. These systems enable secure computation on encrypted data without decryption, effectively minimizing risks associated with data breaches or unauthorized access. Cryptographic techniques such as Fully Homomorphic Encryption (FHE) and Secure Multi-Party Computation (MPC) form the foundation of privacy-preserving systems. In particular, FHE has garnered attention due to its ability to support arbitrary computations on encrypted data. However, its practical adoption is hindered by large ciphertext sizes and significant resource requirements, making it a subject of ongoing research and optimization [1, 2, 3, 4]. This paper will explore the properties and threat models of privacy-preserving systems, including input privacy, output privacy, anonymity, data integrity, and access control. The system's workflow is detailed through a step-by-step algorithm, illustrating how data is encrypted, processed, and decrypted in a secure and privacy-preserving manner.

Transciphering has emerged as a method to address the high transfer cost of FHE. It combines the efficiency in terms of the transfer-size-cost of symmetric encryption with the computational compatibility of homomorphic encryption. In a transciphering system, plaintext data is initially encrypted using a high-performance FHE-friendly symmetric (block) cipher, such as Pasta [5], Rasta [6], Chaghri [7], or Hera. The encrypted data is then transformed, or transciphered, into a homomorphic-compatible ciphertext format for processing. This hybrid approach leverages the speed and efficiency of symmetric encryption while enabling secure computations on encrypted data in homomorphic systems [8, 9].
This paper focuses on the implementation and benchmarking of transciphering systems in

privacy-preserving applications. As a supporting contribution to the theoretical aspects of privacy-preserving cryptography, this work centers on practical coding, scripting, and performance evaluation. By bridging theory and practice, this paper aims to provide experimental validation for transciphering techniques.

The primary goals of this paper include:

1. Evaluation of FHE-Friendly Block Ciphers: Implement and benchmark various FHE-friendly block ciphers such as Pasta [5] , using one or more homomorphic encryption libraries (e.g., SEAL [10], HElib [11] or OpenFHE [12]).

2. Analysis of Transferred Data Size: Compare the size of encrypted data transferred in systems with transciphering against systems without transciphering, analyzing the resulting impact on communication efficiency.

3. Performance Benchmarks: Measure and analyze computational overhead, latency, and resource usage in transciphering pipelines compared to direct FHE-based solutions.

4. Focus on the time required by the different parties, to check which one has what computational demand.

5. Implementation-Centric Approach: Focus on scripting and coding practical solutions, emphasizing reproducible experiments and implementation details as the central contribution.

By fulfilling these objectives, this paper aims to serve as a practical complement to theoretical advancements in the field. It highlights the real-world feasibility of transciphering systems while offering insights into their performance characteristics and limitations.

Chapter 2

Preliminaries

Before delving into system design and experimental evaluation, it is essential to establish the mathematical and cryptographic foundations on which this work builds. This chapter reviews group and ring theory (Sections 1.1 and 1.2 respectively), the basics of asymmetric and symmetric encryption (Sections 2.1 and 2.2 respectively) and block Ciphers (Section 2.2.1), and introduces what FHE schemes are (Section 3) and the three FHE schemes (BGV, BFV, and CKKS) used throughout this thesis. We then define the notion of FHE-friendly block cipher algorithms optimized for homomorphic evaluation (in Section 4) and explain the concept of transciphering in Section 5. By providing notation, formal definitions, and high-level overviews of all critical components, this chapter ensures that subsequent discussions remain clear and self-contained.

1 Mathematical Background

Homomorphic encryption schemes rely on algebraic structures called rings (and underlying groups). We briefly review group and ring theory, then describe how these concepts apply to the BFV scheme. For details on BGV and CKKS implementations, see Section 4 within this Chapter.

1.1 Group

A **group** is a set G equipped with a binary operation $\circ$ that satisfies the following properties:

1. Closure: $\forall a, b \in G : a \circ b \in G$.
2. Associativity: $\forall a, b, c \in G : (a \circ b) \circ c = a \circ b \circ c = a \circ (b \circ c)$.
3. Identity Element: $\exists! e \in G : \forall a \in G : e \circ a = a = a \circ e$.
4. Inverse Element: $\forall a \in G, \exists b \in G : a \circ b = e = b \circ a$.

When the group is also commutative ($\forall a, b \in G : a \circ b = b \circ a$), the group is Abelian (commutative). In that case, we typically denote the operation by '$+$' instead of '$\circ$'. Groups are fundamental in cryptography because they provide a structured way to perform operations like addition or multiplication while preserving certain properties.

1.2 Ring

A ring is an algebraic structure that generalizes the concept of groups by introducing two binary operations: addition ($+$) and multiplication ($\cdot$). Formally, a ring $\mathcal{R}$ is a set equipped with two operations satisfying the following properties:

1. Abel Group under Addition: All the properties of Abel group as in Section 1.1 with identity $e_+ = 0$ and the Inverse denoted as $-a \in \mathcal{R}$.

2. Monoid group under multiplication: With identity $e_\cdot = 1$ but not necessarily inverses.

3. Distributivity: Multiplication distributes over addition, $\forall a, b, c \in \mathcal{R} : a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

The addition and multiplication are to be interpreted as notation. They can also represent other operations, but at least one of these must be commutative. Rings are more general than fields because they do not require multiplicative inverses for all elements (except the identity). This makes rings suitable for cryptographic applications where certain operations (e.g., modular arithmetic) are required.
Example: The Brakerski-Fan-Vercauteren (BFV) scheme is a lattice-based homomorphic encryption scheme designed for exact arithmetic over integers. It belongs to the family of RLWE-based encryption schemes and supports additive and multiplicative homomorphic operations over ciphertexts. BFV is especially suited for applications requiring deterministic and precise computation on encrypted data and it operates over a polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$, where N is a power of 2, and $q \in \mathbb{Z}$ is the ciphertext modulus. Plaintexts are elements of the ring $R_t = \mathbb{Z}_t[X]/(X^N + 1)$, with $t < q$ being the plaintext modulus. The quotient ring structure supports modular arithmetic and polynomial encoding, which is crucial for batching and homomorphic operations.

2 Overview of Cryptography

Let us define different kinds of ciphers such that we can work with these definitions afterward. We can categorize these in three ways: 0 secret/known keys (Hash), only secret key(s) (Symmetric), and secret and public key(s) (Asymmetric). We can further subdivide the Symmetric ciphers into Block cipher and Stream Ciphers. We will only handle asymmetric, symmetric, and block ciphers in Sections 2.1, 2.2, 2.2.1 respectively.

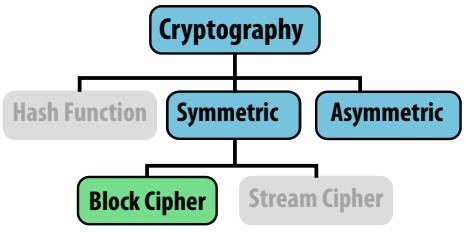
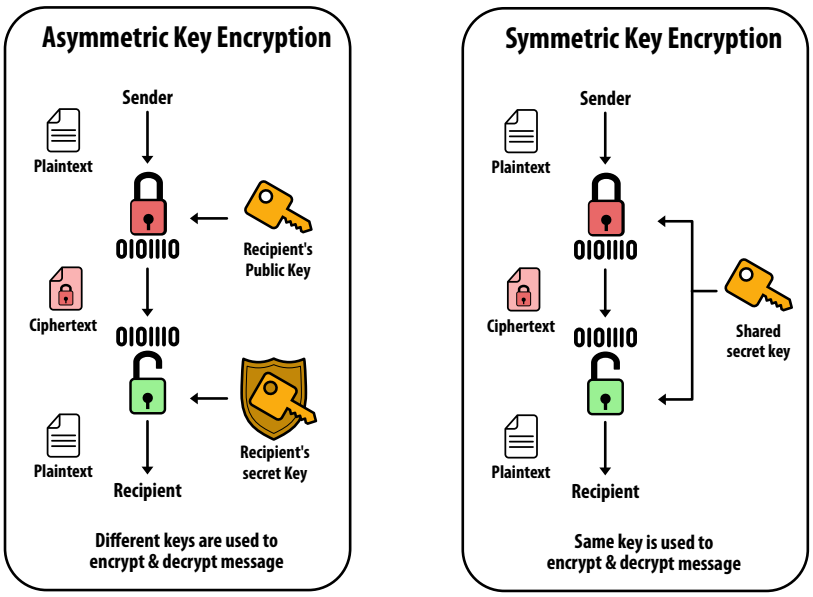


Figure 2.1: Cryptography diagram

2.1 Asymmetric

Asymmetric encryption, also known as Public-Key cryptography, is a foundational element of modern cryptographic systems. It operates using a pair of mathematically related keys: a Public Key (PK), which is openly distributed, and a Secret Key (SK), which is kept private. The encryption process uses the recipient's Public Key to secure a message, while the corresponding Secret Key decrypts the encrypted message. This mechanism ensures confidentiality and authentication in secure communications. See Figure 2.2a at page 5 for a comprehensive diagram of how this algorithm works.
The important factor is that these algorithms are carefully designed to ensure that deriving the Secret Key from the public key is computationally infeasible [13, 14]. The sender uses the public key to encrypt the plaintext message to a ciphertext and only the holder of the secret key can decrypt the ciphertext back to the plaintext [15].
Some examples of Asymmetric encryption include, but are not limited to: RSA (Rivest-Shamir-Adleman) [13, 16], Elliptic Curve Cryptography (ECC) [17], Diffie-Hellman Key Exchange [13], ElGamal Encryption [14]. Their applications go from securing internet traffic to digital signatures or authenticating software/documents/transactions and everything in between [13, 18].

Usage in paper: The Fully Homomorphic Encryption discussed in Section 3 can be seen as an asymmetric encryption because of the creation of a Secret Key-set, which decrypts the result, and a public key-set which encrypts the data. This will be further discussed in Section 3 of Chapter 3.



(a) Asymmetric (b) Symmetric

Figure 2.2: Diagram of Encryption

2.2 Symmetric

Symmetric encryption is a cryptographic system where the same key is used for both encryption and decryption. It is one of the most widely utilized methods of securing data due to its efficiency and relatively straightforward implementation. Symmetric encryption is particularly well-suited for scenarios where high-performance encryption is necessary, such as securing network traffic, protecting stored data, and enabling secure communication between trusted parties [13]. See Figure 2.2b at page 5 for a comprehensive diagram of how this algorithm works.
In symmetric encryption, the secret key must be shared (in a secure way) between the sender and recipient confidentially and securely. This key is used to transform plaintext into ciphertext during encryption and to reverse the transformation during decryption [16].
There are two big divisions within the Symmetric Encryption Systems, as explained at the beginning of Section 2: Stream Ciphers, which work by streaming data and generating the key on the fly, Block Ciphers work by dividing the message/file in blocks and handling them separately respectively. As we will not handle or use Stream Ciphers, we have ignored this and used Section 2.2.1 to explain Block ciphers.
Some examples of Symmetric encryption include but are not limited to: Advanced Encryption Standard (AES) [16, 15], Data Encryption Standard (DES) and Triple DES (3DES) [14], ChaCha20 [19] and ASCON [20]. Their applications go from securing internet traffic to digital signatures or authenticating software/documents/transactions and everything in between [13, 18].

2.2.1 Symmetric Block Ciphers

Block ciphers are symmetric encryption algorithms (as in Section 2.2) that process fixed size blocks of data. They operate by applying a series of transformations, controlled by a secret key, to achieve secure encryption. Block ciphers are fundamental to modern cryptography and are widely used in securing data transmission, file storage, and cryptographic protocols.
In this thesis, block ciphers play a critical role in the hybrid homomorphic encryption (HHE) pipeline, where they enable efficient initial encryption before transciphering is applied. Block ciphers are characterized by their internal structure and transformation layers. Most block ciphers are iterated constructions, applying a sequence of operations, known as rounds, to achieve diffusion and confusion, the two key principles defined by Shannon for secure cipher design [21].
Block ciphers apply a fixed number of rounds, where each round performs a transformation on the intermediate state. The round transform encapsulates:

1. Key mixing: Incorporating the round key
2. Non-linear substitution: S-box applications
3. Linear diffusion: Permutations or matrix multiplications

Each round increases resistance to cryptanalysis by increasing statistical distance between ciphertext and plaintext, ideally converging to pseudorandomness.

The key schedule is the algorithm that expands a master key $K \in \mathbb{Z}_p^n$ into a set of round keys $\{K^{(1)}, K^{(2)}, ..., K^{(r)}\}$ used in each encryption round. This expansion ensures that each round operates with a distinct key component, reducing structural weaknesses and improving resistance against related-key attacks. In lightweight or FHE-friendly ciphers, the key schedule may be simplified for performance, e.g., using a static key for all rounds or applying modular arithmetic-based derivation.
A Substitution-Permutation Network (SPN) is a widely adopted design structure in block cipher construction. It consists of alternating layers of substitution (non-linear) and permutation (linear) transformations. Each round in an SPN-based cipher typically includes:

1. Substitution layer: Applies a non-linear function (commonly known as an S-box) independently to each byte or word of the block. This layer provides non-linearity and is typically implemented using lookup tables or polynomial functions.

2. Permutation layer: Reorders or linearly mixes the outputs of the substitution layer. It can be implemented using matrix multiplication, slot rotation, or bitwise permutations.

An alternative cipher structure is the Feistel network, where the block is split into two halves and one half is updated based on a function of the other half and a round key. A single round of a Feistel cipher can be expressed as: $(L_{i+1}, R_{i+1}) = (R_i, L_i \oplus F(R_i, K_i))$ where (L_i, R_i) is the state at round i, F is the round function, and K_i is the round key. The Feistel structure allows for invertibility regardless of whether F itself is invertible.
There are many different kinds of block ciphers based on the SPN or Feistel networks. One, specifically Pasta, will be explored in-depth in Section 4.1.

Examples: Data Encryption Standard (DES) was an early block cipher standard using a 56-bit key and a Feistel structure. It is now considered insecure due to its small key size and vulnerability to brute-force attacks [14]. Triple DES (3DES) extends DES by applying the DES algorithm three times with different keys, effectively increasing its security. However, it has largely been replaced by AES due to performance limitations [13]. Advanced Encryption Standard (AES) is the most widely adopted block cipher, operating on 128-bit blocks with key sizes of 128, 192, or 256 bits. It employs an SPN structure with 10–14 rounds depending on the key size [16]. For this paper, we also consider PASTA an important example, which will be explained in Section 4.1. In short, this separates the data like a Feistel structure, has multiple rounds and each round consists of SPN operations while focusing on compatibility and efficiency regarding FHE.

Application in FHE-Compatible Systems: In the context of transciphering, the block cipher must meet specific criteria: Low multiplicative depth, to remain compatible with leveled FHE schemes. SIMD-amenable operations, to efficiently process vectors in packed ciphertexts. Polynomial substitutability, S-boxes must be representable by low-degree polynomials.
One example of this is PASTA [5] (explained in Section 4.1 of Chapter 2), which is a modern SPN-based cipher specifically designed with these constraints in mind. It demonstrates how traditional cryptographic structures can be adapted to operate securely

and efficiently within homomorphic environments. The integration of block ciphers into the transciphering pipeline offers significant benefits in terms of communication efficiency while preserving the capacity for encrypted computation.

3 Fully Homomorphic Encryption

Fully Homomorphic Encryption is a cryptographic technique that enables computations to be performed directly on encrypted data without requiring decryption. This property ensures the confidentiality of sensitive data during computation and is particularly suited for cloud computing, secure data outsourcing, and privacy-preserving applications.
Partially homomorphic encryption (supporting only a single operation (Group in Section 1.1)) has long been possible via group-based schemes. To support both addition and multiplication, we must move to a ring structure (Section 1.2). In other words, to evaluate arbitrary circuits, a homomorphic scheme must be a ring homomorphism rather than merely a group homomorphism.
As such, we want this new FHE to have the following properties to ensure secure computations where the owner of the secret key can decrypt the result, and any willing computer can encrypt and then compute the data according to some function.

Homomorphism
A homomorphic encryption scheme allows specific operations to be performed on ciphertexts such that the result, when decrypted, matches the outcome of the corresponding operation performed on the plaintexts. For FHE, this extends to both addition and multiplication, enabling the evaluation of arbitrary functions as circuits composed of these operations.

Noise Management
This noise is introduced in order to make it more difficult to map the ciphertext to the plaintext by diffusing the ciphertext while introducing a negligible difference. However, this noise grows with each operation. If the noise exceeds a certain threshold, decryption fails. Modern FHE schemes use techniques such as bootstrapping and leveled FHE to manage noise effectively [1].

Bootstrapping
Bootstrapping was first introduced in Gentry's construction of FHE [1]. Bootstrapping is a technique that refreshes a noisy ciphertext by homomorphically decrypting and re-encrypting it. This process reduces noise and allows more non-linear computations on encrypted data.

Leveled FHE
Leveled FHE schemes avoid the computational overhead of bootstrapping by restricting computations to circuits of a specific depth. These schemes are efficient for applications where the depth of computations can be predetermined [3].

3.1 Overview of FHE schemes: BGV, BFV and CKKS

This section provides a formal mathematical description of the three primary Fully Homomorphic Encryption (FHE) schemes used in this thesis: BGV, BFV, and CKKS. Each scheme is based on the hardness of the Ring Learning With Errors (RLWE) problem, and operates over a polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$, where q is the ciphertext modulus and N is the ring degree (usually a power of 2). Each scheme supports encrypted arithmetic operations under different constraints and capabilities: CKKS is the only of these three that can compute using floats instead of only integers.

3.1.1 BGV: Brakerski-Gentry-Vaikuntanathan Scheme

The Brakerski-Gentry-Vaikuntanathan (BGV) scheme [2] is a lattice-based leveled homomorphic encryption scheme that allows both addition and multiplication on ciphertexts. It supports batching through the Chinese Remainder Theorem (CRT), enabling Single Instruction, Multiple Data (SIMD) operations over encrypted vectors.

- Uses modulus switching to manage noise growth
- Supports key-switching and bootstrapping
- Efficient in homomorphic addition and multiplication for integers
- Addition: Component-wise addition modulo q
- Multiplication: Increases the ciphertext size, requiring relinearization
- Modulus switching: Reduces noise and ciphertext modulus after multiplication

BGV is particularly suitable for applications that require fine control over modulus and noise, and it is implemented in libraries such as HElib and OpenFHE.

Key Generation: Secret key: $s \in R_q$. Public key: $(a, b = -a \cdot s + e)$, where $a \in R_q$ is random and $e \in R_q$ is a small error sampled from a noise distribution.

Encryption: For plaintext $m \in R_t$:

$$\text{Enc}(m) = (c_0, c_1) = (a \cdot r + m + e_1, b \cdot r + e_2)$$

where $r, e_1, e_2 \in R_q$ are sampled from a noise distribution.

Decryption:

$$m = c_0 + c_1 \cdot s \mod q \mod t$$

3.1.2 BFV: Brakerski-Fan-Vercauteren Scheme

The Brakerski-Fan-Vercauteren (BFV) scheme [22] is conceptually similar to BGV but designed to align more closely with integer arithmetic. It allows exact computation on encrypted integers and is well-suited to machine learning and data analytics pipelines that work with discrete input.

- Operates over the same polynomial ring R_q
- Encodes plaintexts as integers modulo t (plaintext modulus)
- Uses relinearization and rescaling to manage noise
- Efficient SIMD support for encrypted vectors
- Homomorphic addition and multiplication are defined similarly to BGV
- After each multiplication, relinearization is required to reduce ciphertext size
- Rescaling helps manage noise growth, with modulus switching as optional

BFV offers a balance between usability and performance and is commonly used for applications involving encrypted aggregation or statistics over integer-valued data.

Key Generation: Secret key: $s \in R_q$. Public key: $(a, b = -a \cdot s + e)$, where $a \in R_q$ is random and $e \in R_q$ is a small error sampled from a noise distribution.

Encryption: For plaintext $m \in R_t$, encode it into R_q:

$$\text{Enc}(m) = (c_0, c_1) = (\Delta m + a \cdot s + e_0, a)$$

where $\Delta = \lfloor \frac{q}{t} \rfloor$ is the scaling factor and e_0 is noise.

Decryption: This converts the decrypted polynomial from R_q back into R_t.

$$m = \left\lfloor \frac{t}{q}(c_0 + c_1 \cdot s) \right\rceil \mod t \qquad \mu = c_0 + c_1 \cdot s$$

3.1.3 CKKS: Cheon-Kim-Kim-Song Scheme

The Cheon-Kim-Kim-Song (CKKS) scheme [23] extends FHE to support approximate arithmetic on real or complex numbers. Unlike BGV and BFV, which are exact, CKKS allows small errors in computation in order to enable the processing of non-integer operations.

- Supports approximate homomorphic encryption for floating-point arithmetic
- Allows encoding of real-valued vectors with rescaling
- Suitable for encrypted signal processing, neural networks, and analytics

- Trades exactness for performance and scalability
- CKKS supports approximate additions and multiplications: $\text{EvalAdd}(\text{Enc}(x), \text{Enc}(y)) \approx x + y, \text{EvalMult}(\text{Enc}(x), \text{Enc}(y)) \approx x \cdot y$

CKKS is the preferred scheme when the application domain can tolerate minor precision loss, such as in machine learning inference or differentially private statistics.

Key Generation: Secret key: $s \in R_q$. Public key: $(a, b = -a \cdot s + e)$, where $a \in R_q$ is random and $e \in R_q$ is a small error sampled from a noise distribution.

Encoding: Real (or complex) plaintext vectors $\mathbf{m} \in \mathbb{C}^n$ are encoded into R_q using a canonical embedding.

Encryption: For plaintext $m \in R_t$, encode it into R_q:

$$\text{Enc}(m) = (c_0, c_1) = (\Delta m + a \cdot s + e_0, a)$$

where $\Delta \in \mathbb{R}$ is a scaling factor and $e_0 \in R_q$ is the error.

Decryption:

$$m' = \frac{1}{\Delta}(c_0 + c_1 \cdot s) \approx m$$

Due to rounding and noise, the result m' is only approximately equal to the original plaintext m

Rescaling: After each multiplication, ciphertexts are rescaled to reduce their modulus and maintain numerical stability. Rescaling also adjusts the effective scale, limiting the number of levels (multiplications) before bootstrapping is required.

3.1.4 Conclusion and Summary

All three schemes are supported by OpenFHE and can be configured for different performance, security, and precision trade-offs. The choice of scheme depends on the nature of the input data, the required arithmetic (exact vs. approximate), and the overall system constraints.

Table 2.1: Comparing FHE schemes

Scheme	Domain	Precision	Note
BGV	$\mathbb{Z}_q$	No	
BFV	$\mathbb{Z}_t$	No	Ideal for discrete values
CKKS	$\mathbb{R}, \mathbb{C}$	Yes	Efficient floating-point

These schemes differ in their handling of noise, modulus switching, and supported arithmetic. Their mathematical underpinnings define the range of secure operations and directly impact the performance and scalability of encrypted computation pipelines.
In order to visualise the branches of these differing HE schemes, Figure 2.3 is most appropriate.

4 FHE-Friendly Block Ciphers

Block ciphers designed for FHE applications are known as FHE-friendly block ciphers. These ciphers are optimized for arithmetic complexity, making them efficient when implemented in homomorphic encryption schemes. Unlike traditional block ciphers, which prioritize low latency and hardware efficiency, FHE-friendly block ciphers minimize the number of non-linear operations, such as multiplications, which are computationally expensive in FHE [24].
Key Characteristics of FHE-Friendly Block Ciphers can be summarized as follows:

- Arithmetic Complexity Optimization: FHE-friendly block ciphers reduce the number of non-linear operations (e.g., modular exponentiation or multiplication) to improve performance in homomorphic environments.
- Field-Based Operations: These ciphers often operate over finite fields (For example F_{2^n} or F_p) to align with the mathematical foundations of FHE schemes.
- Efficient Key Schedules: The key schedule in FHE-friendly block ciphers is designed to ensure minimal computational overhead during key expansion.

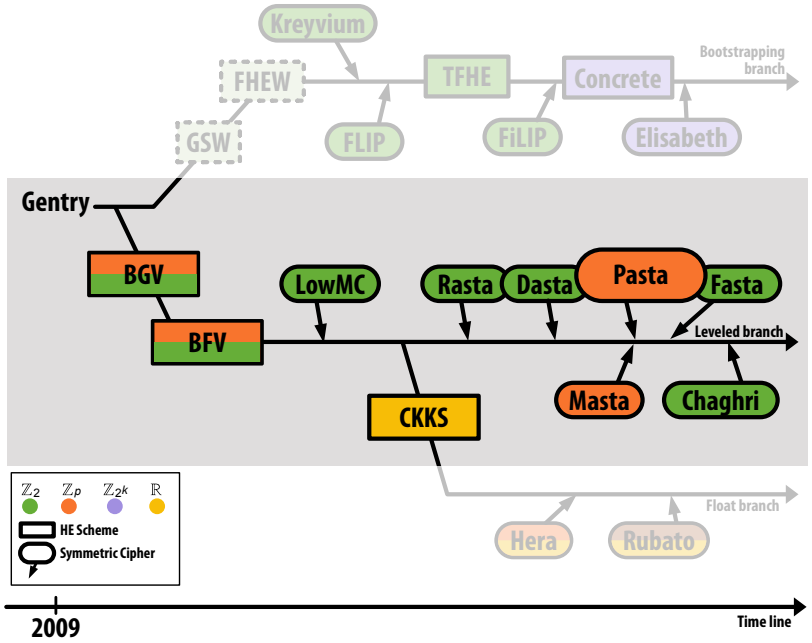


Figure 2.3: FHE-friendly Block ciphers

Examples:
We have a whole evolution of FHE-friendly block ciphers, as shown in Figure 2.3. Out of these, we will pick a few as examples. However, as seen in Chapter 4, we will be focusing on the Pasta cipher, so this will be explained fully in Section 4.1 of this Chapter.
Some additional examples include **LowMC**, which is a block cipher designed specifically for cryptographic protocols, including FHE and Multi-Party Computation (MPC). It

minimizes the number of non-linear S-boxes, reducing computational costs while maintaining security [8]. **HE-Friendly AES** which is an adapted variant of AES for homomorphic encryption by reducing the number of rounds or modifying the S-box layer to decrease arithmetic complexity [9]. **Chaghri and SELJUK** were proposed in more recent research, these ciphers are optimized for homomorphic encryption and achieve better efficiency than traditional counterparts like AES when implemented in FHE systems [26].

4.1 Pasta [5]

We will now present in detail one FHE-friendly block cipher, Pasta, and explain its design and implementation.

The PASTA cipher is a family of FHE-friendly symmetric block ciphers designed to operate efficiently in homomorphic environments. Its core innovation is to ensure that all operations, including non-linear substitutions, are composed of low-depth arithmetic circuits compatible with FHE evaluation, particularly under schemes like BFV, BGV, and CKKS.

The cipher is specifically designed to achieve low multiplicative depth, minimize the use of non-linear gates, and maximize SIMD efficiency under FHE, while maintaining cryptographic strength. It achieves this by adopting a simple SPN (Substitution-Permutation Network) structure where all components are chosen to be arithmetic operations over finite fields or rings.

PASTA operates on plaintext blocks $m \in \mathbb{F}_p^n$, where $\mathbb{F}_p$ is a finite field (often $\mathbb{F}_p$), n is the block size (commonly 128 or 256 bits, split across multiple slots) and p satisfies $\gcd(p-1, 3) = 1$ such that x^3 is invertible over this field. The cipher takes a secret key $k \in \mathbb{F}_p^n$ and produces ciphertext $c \in \mathbb{F}_p^n$. Additionally, we need a nonce N, meaning this number can only be used once (often a timestamp), some number of rounds $r \geqslant 1$, and a block counter i to the secret key.

With vector $\vec{x} \in \mathbb{F}_p^{2t}$, Pasta $-\pi$ is defined as follows, and then represented in diagram form in Figure 2.4.

$$\text{Pasta}-\pi(\vec{x}, N, i) = A_{r,N,i} \circ S_{\text{cube}} \circ A_{r-1,N,i} \circ S_{\text{feistel}} \circ A_{r-2,N,i} \circ S_{\text{feistel}} \circ \cdots \circ A_{1,N,i} \circ S_{\text{feistel}} \circ A_{0,N,i}(\vec{x})$$

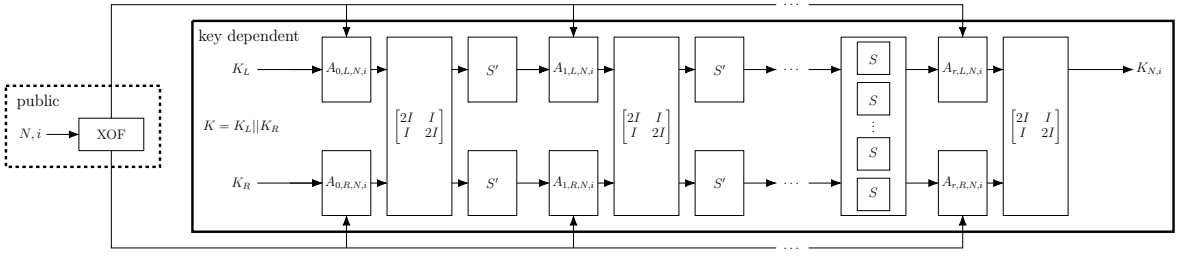


Figure 2.4: Pasta Diagram [5]

Each round consists of the Linear/affine layer and non-linear layer, in this case, a substitution box (S-box):

Linear layer: This will include a calculation as $\begin{pmatrix} \vec{y}_L \\ \vec{y}_R \end{pmatrix} = \begin{pmatrix} 2I_t & I_t \\ I_t & 2I_t \end{pmatrix} \cdot \begin{pmatrix} A_{j,L,N,L}(\vec{x}_L) \\ A_{j,R,N,L}(\vec{x}_R) \end{pmatrix}$ for $j \in \{0, 1, \ldots, r\}$. Where I_t the identity matrix of $\mathbb{F}_p^{t \times t}$ and in the following equation, $M_{j,L,N,i}, M_{j,R,N,i} \in \mathbb{F}_p^{t \times t}$ (invertible matrix) and $\vec{c}_{j,L,N,i}, \vec{c}_{j,R,N,i} \in \mathbb{F}_p^t$ get generated for each round according to an XOF with seed N and counter i.

$$A_{j,N,i}(\vec{x}) = \begin{pmatrix} M_{j,L,N,i}(\vec{x}_L) + \vec{c}_{j,L,N,i} \\ M_{j,R,N,i}(\vec{x}_R) + \vec{c}_{j,R,N,i} \end{pmatrix}$$

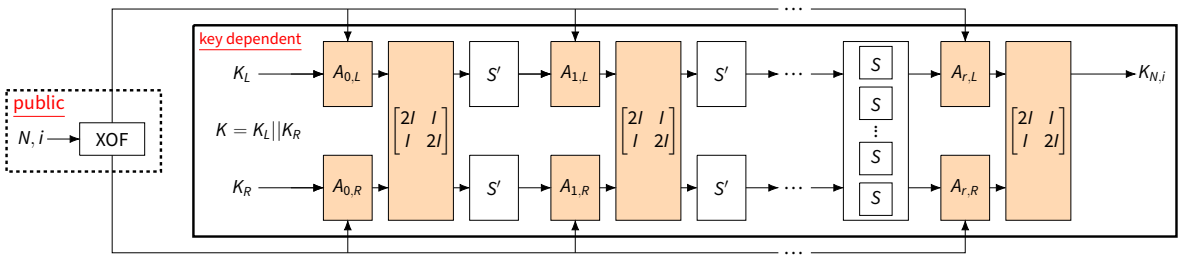


Figure 2.5: Pasta - Linear Layer [27]

The colored regions in Figure Figure 2.5 represent all the locations where the linear layer takes place.

S-Box layer: This is the S_{feistel} layer for every round except the last one, which is the S_{cube} layer. Define $\vec{y} = \begin{bmatrix} y_0 & y_1 & \ldots & y_{t-1} \end{bmatrix} \in \mathbb{F}_p^t$ and then:

$$S_{\text{feistel}}(\vec{x}) = \begin{bmatrix} S'(\vec{x}_L) & S'(\vec{x}_R) \end{bmatrix} \qquad S'(\vec{y}) = \begin{cases} y_0 & l = 0 \\ y_l + (y_{l-1})^2 & l \neq 0 \end{cases} \in \mathbb{F}_p^t$$

$$S_{\text{cube}}(\vec{x}) = \begin{bmatrix} x_0^3 & x_1^3 & \ldots & x_{s-1}^3 \end{bmatrix}$$

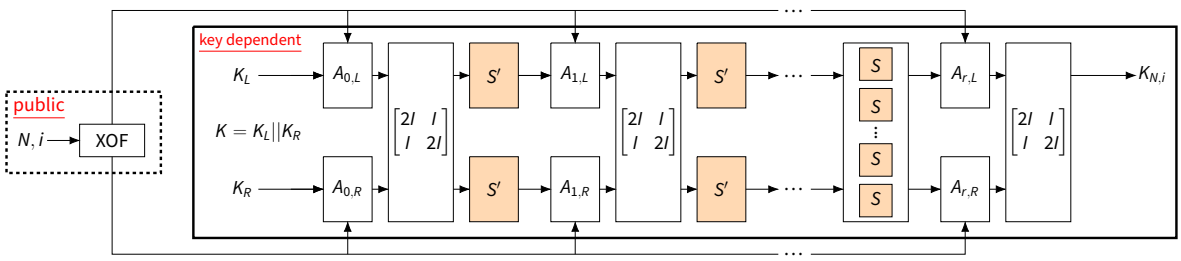


Figure 2.6: Pasta - S Box [27]

The colored regions in Figure Figure 2.6 represent all the locations where the S-Box layer takes place. Note that only within the last round, the substitution is $S = S_{\text{cube}}$, while in all other rounds, the substitution is $S' = S_{\text{feistel}}$.

Encryption/Decryption: Under secret key, the Pasta block cipher defines a symmetric encryption scheme designed to operate efficiently over message space $\vec{m} \in \mathbb{F}_p^l$, particularly within FHE and transciphering settings. The key generation algorithm samples a secret key $\text{sk} \in \mathbb{F}_p^{2t}$. Encryption and decryption are performed in a nonce-based mode to ensure semantic security across repeated encryptions of the same message.
To encrypt a plaintext, the message is divided into sub-blocks $\vec{m} = \begin{bmatrix} m_0 & m_1 & \dots & m_j \end{bmatrix}$, where each $\vec{m}_i \in \mathbb{F}_p^t$. The encryption algorithm $\text{Enc}_{\text{sk}}(\vec{m}, N)$ maps this message to a corresponding ciphertext $\vec{c} = \begin{bmatrix} c_0 & c_1 & \dots & c_j \end{bmatrix}$ with $\vec{c_i} = \vec{m}_i + \text{left}_t\left(\text{Pasta}-\pi(\text{sk}, N, i)\right)$, where left_t extracts the first t words of the pseudorandom output generated by a Pasta instance initialized with secret key sk, nonce N, and block index i.
Decryption proceeds analogously. The ciphertext $\vec{c} \in \mathbb{F}_p^l$ is parsed into the same sub-block structure $\vec{c} \in \mathbb{F}_p^t$, and the decryption algorithm $\text{Dec}_{\text{sk}}(\vec{c}, N)$ reconstructs the original message via $\vec{m}_i = \vec{c}_i - \text{left}_t\left(\text{Pasta}-\pi(\text{sk}, N, i)\right)$. The subtraction in $\mathbb{F}_p^t$ ensures that the message is exactly recovered, assuming the correctness of the internal round function $\text{Pasta}-\pi$. The structure of the cipher supports vectorized computation and non-deterministic encryption while preserving compatibility with homomorphic evaluation in transciphering frameworks.
According to Page 19 of [5]: "We propose a 3-round instance Pasta-3 as well as a 4-round instance Pasta-4 using Shake128 [NIS15] as XOF. These instances provide at least 128 bits of security for the prime fields $\mathbb{F}_p$ with $\log_2(p) > 16$ and $\gcd(p-1, 3) = 1$. Table 5 shows the block and key sizes and compares them to Masta and Hera." Where 'NIS15' refers to [28]

5 Transciphering

Transciphering combines an FHE-friendly block cipher with Fully Homomorphic Encryption to enable more efficient computation on encrypted data. While FHE supports arbitrary computations on ciphertexts, its high computational overhead makes it impractical to transfer large datasets. Transciphering addresses this by first encrypting data with a symmetric-key cipher and then converting the ciphertext into an FHE-compatible format.
This process is also known as Hybrid Homomorphic Encryption (HHE), originally introduced by Naehrig, Lauter, and Vaikuntanathan in [29], presents a strategy to reduce communication overhead in fully homomorphic computation pipelines. The core idea is to offload encryption complexity from the data owner by utilizing a symmetric encryption scheme locally while retaining the ability to perform computations homomorphically on encrypted data at the server.
Formally, let $m \in \mathbb{Z}_p^n$ denote a plaintext message held by the data owner, and let SK_{BC} be a symmetric key belonging to an FHE-friendly block cipher. Instead of directly applying a homomorphic encryption function FHE. Enc, the sender first computes: $C = \text{Symm.Enc}(m, \text{SK}_{\text{BC}})$, meaning a symmetric encryption algorithm (e.g., a block cipher such as Pasta).
The symmetric ciphertext C is then transmitted to the server/aggregator or computing entity, along with an FHE-encrypted version of the symmetric key: $\text{C}_{\text{key}} = \text{FHE.Enc}(\text{SK}_{\text{BC}}, \text{PK}_{\text{FHE}})$
On the server side, the system homomorphically evaluates the decryption function of the

symmetric scheme: $V = \text{FHE.Eval}\left(\text{Symm.Dec}(\cdot, \text{C}_{\text{key}}), V', \text{PK}_{\text{FHE}}\right)$ where $V' = \text{FHE.Enc}(C, \text{PK}_{\text{FHE}})$ which just gets the values of the Block-cipher encrypted data into FHE compatible format.
This process, often referred to as Transciphering or Hybrid HE, converts the efficiently encrypted symmetric ciphertext into a homomorphically encrypted form, upon which arbitrary computations can be securely performed using FHE evaluation functions:

$$F = \text{FHE.Eval}(f, V, \text{PK}_{\text{FHE}})$$

If one would decrypt this using the FHE Secret Key, one would receive the $f(m)$ result. This hybrid methodology trades communication efficiency since symmetric ciphertexts have no expansion, for increased computational load on the server, which must evaluate the symmetric decryption logic under FHE. It also introduces an additional cryptographic requirement: the data owner must transmit the symmetric key in a secure, homomorphically encrypted form to enable the transciphering process.

Applications: While Transciphering doesn't expand the capabilities of cryptographic computations using FHE means, it is something that allows more complex and larger processes to become feasible. As the FHE encryption expands much more, and the symmetric encryption has a constant expansion factor, it would become more reasonable to send larger data files through the wire, at the cost of compute time on the Server side, which will be thoroughly explored in Chapter 4 Section 3. The purpose of this paper is to evaluate and put in numeric values the efficiency gain of the transferred data.

Chapter 3

Privacy Preserving Systems

With the necessary background in place, we now present the concept of a Privacy Preserving System (PPS) along with our architecture of a privacy-preserving computation framework that employs transciphering. This chapter begins by listing the key security properties and defining the threat model(s) considered in our analysis. We then describe in detail the roles of Data Owners, the Server, and the Data Customer, and provide a step-by-step workflow for homomorphic evaluation with and without transciphering. Variants such as dummy-value padding, thresholding, and no choice-vector are introduced to illustrate configurable privacy controls. Finally, we discuss concrete use cases that demonstrate how this architecture can be applied in real-world domains.

1 Introduction

Privacy-preserving systems are designed to enable secure data processing while ensuring that sensitive information remains protected. These systems are increasingly relevant in applications involving data sharing, outsourced computations, and collaborative environments where privacy is a critical concern. The fundamental principles of privacy-preserving systems include data confidentiality, data aintegrity, and access control, often achieved through cryptographic techniques. This solution would cover various use cases, some of which will be briefly mentioned here, while a more detailed explanation will be given in Section 6 of Chapter 3.

In the era of data-driven decision-making, the protection of sensitive personal, medical, financial, and behavioral data has become a critical concern. Modern digital services increasingly rely on large-scale data analytics and cloud-based computation, often involving third-party platforms that are not fully trusted by data contributors. While centralized computation offers substantial performance and scalability benefits, it also introduces severe privacy risks. Notably, data breaches, unauthorized profiling, and inference attacks are among the most pressing threats to user confidentiality and trust [30, 31].

To address these challenges, the design of Privacy-Preserving Systems has emerged as a central objective in applied cryptography and secure computing. A privacy-preserving system allows parties to collaboratively perform computations on private inputs without revealing those inputs to one another or the computing infrastructure. These systems are particularly relevant in contexts where legal regulations such as the General Data Protection Regulation (GDPR) or the Health Insurance Portability and Accountability

Act (HIPAA) mandate strict confidentiality guarantees for personal data [32].
In a privacy-preserving architecture, three roles can be defined as:

- Data Owner (DO): The individual or organization holding sensitive raw data.
- Server (S): A computationally capable but semi-trusted platform that executes encrypted computations.
- Data Customer (DC): An authorized party who defines the computation and receives the final result.

A key goal of the system is to enable the evaluation of some function $f(v_1, ..., v_n)$ over encrypted inputs v_i provided by the DOs, such that the DC learns only the result $f(v)$, and the Server learns nothing about the inputs nor outputs. Afterward, this paper will also introduce a manner in which participation is obligatory instead of an option as well as a way to appear to participate by the DOs.
This thesis investigates a specific instantiation of such systems using Transciphered FHE pipelines, where data is first encrypted with a lightweight symmetric cipher, as explained in Section 2.2.1, and then transformed into an FHE-encrypted format via homomorphic evaluation of the decryption logic, as seen in Section 5 of Chapter 2. This hybrid approach significantly reduces communication overhead and supports scalable secure computation, while still ensuring robust privacy guarantees. The remainder of this chapter introduces the system architecture, threat models, and technical building blocks underlying the implementation and analysis of such a privacy-preserving framework.

2 Privacy Properties and Threat Models

Privacy-preserving systems protect sensitive data during storage, transmission, and computation. They uphold key properties, such as confidentiality, data integrity, and anonymity, to ensure data security end-to-end. We also model participant behavior via two common adversarial traits: semi-honest and malicious. What follows is a list of important Privacy Properties, which will be used to compare different variations on the Privacy Preserving System and protocol.

1. Input Privacy ensures that the raw data provided by the Data Owners is not revealed to unauthorized parties, including the Server and other DOs. This is achieved through encryption techniques, such as Fully Homomorphic Encryption, which allows computations to be performed on encrypted data without decryption [1].
2. Output Privacy ensures that the results of the computation are only accessible to authorized parties, typically the Data Customer. The results are decrypted using the DCs' secret key, ensuring that no intermediate or final results are exposed to the Server or other entities [33].
3. Anonymity ensures that the identities of the Data Owners are not revealed during the computation process. This is particularly important in scenarios where the data is sensitive, such as medical or financial data.

4. Data integrity ensures that the data is not tampered with during transmission or computation. Cryptographic techniques, such as digital signatures or hash functions, can be used to verify the authenticity and integrity of the data [33].

5. Confidentiality ensures that the data is only accessible to authorized parties. This is achieved through encryption techniques, such as FHE or symmetric-key encryption, which prevents unauthorized access to the data [1].

6. Access Control ensures that only authorized parties can perform specific operations on the data. This is typically achieved through role-based access control (RBAC) or attribute-based access control (ABAC), which define the permissions of each entity in the system [34].

Threat Models of Adversaries/Participants
The behavior of participants in a privacy-preserving system can be modeled using different adversarial traits. These models define the level of trust placed in the participants and the potential threats they may pose. The two most common Threat models are the **semi-honest** (honest-but-curious) model and the **malicious** model.

1. Semi-Honest (Honest-but-Curious) Model:
   In the semi-honest model, participants are assumed to follow the protocol correctly but may attempt to learn additional information from the data they observe. For example, the Server may try to infer sensitive information about the Data Owners by analyzing the encrypted data or computation results. While this model is less restrictive, it still provides strong privacy guarantees through cryptographic techniques like FHE or MPC [1].

2. Malicious Model:
   In the malicious model, participants may deviate from the protocol in arbitrary ways, including tampering with data, providing false inputs, or attempting to disrupt the computation. This model is more realistic in scenarios where participants cannot be fully trusted. To defend against malicious adversaries, additional cryptographic techniques, such as zero-knowledge proofs or verifiable computation, are required to ensure the correctness and integrity of the computation [35].

3 System Architecture Designed in This Work

In short and without delving into the details, the interaction and workflow of the different parties will be very close to the following enumeration. Please refer to Figure 3.1 for a visual aid. These are all the steps that should be taken per each DC, in case there are multiple, the whole process has to be redone as there is no secure way to transfer secret keys between DCs.

1. Key Management: The DC generates and distributes public keys to the DOs and server, while keeping the private key secure. This is a standard practice in FHE-based systems [1].

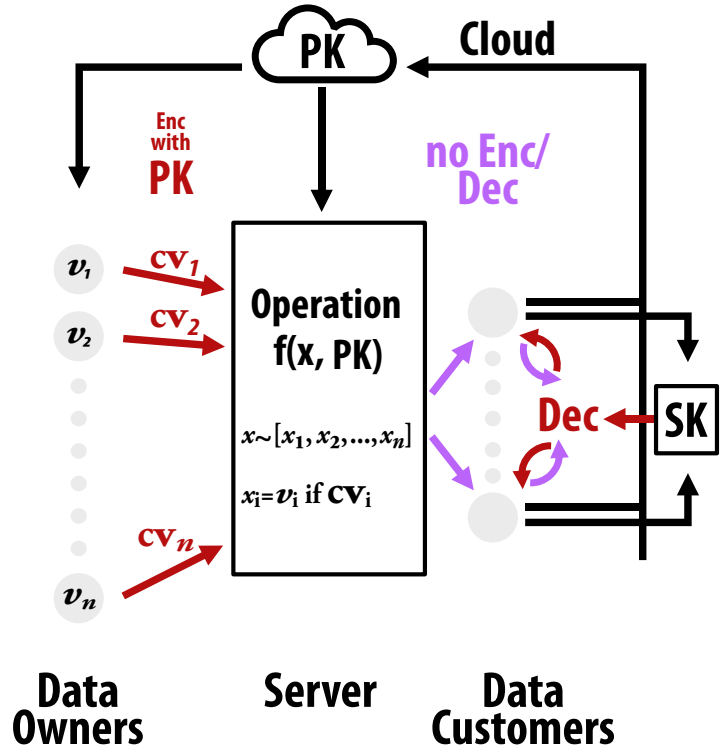


Figure 3.1: Diagram of the Privacy Preserving System

2. Data Encryption: The DOs encrypt their data using the provided public key and optionally apply a choice vector cv_i to control data sharing. This ensures that only authorized data is shared for computation [33].

3. Computation: The server performs homomorphic computations on the encrypted data, ensuring that the results remain encrypted. This is a core feature of FHE, enabling secure computation on encrypted data [36].

4. Result Decryption: The DC decrypts the final computation results using the private key. This step ensures that only the DC can access the results of the computation [1].

This threat model ensures that this protocol operates securely and efficiently, with a clear separation of responsibilities between the components.
To better understand the workflow, a diagram can be created to illustrate the interactions between the Data Owners, Server, and Data Customer. The following section is a description of the diagram and its various elements, but a full explanation of their workings will be provided in Section 4 of this chapter.
Each step is labeled with a number, 1 through 6, and a brief description.

Entities are the parties that are required to follow the protocol.

- Data Owners (DOs): Represented as multiple nodes on the left side of the diagram. These are the various connected devices that each hold their private data, which

they do not want to reveal, but are required to calculate the requested output of the system.

- Server (S): Represented as a central node in the middle. Also known as aggregator.
- Data Customers (DCs): Represented as multiple nodes on the right side of the diagram. These are the various requesters of some desired function which is based on the secret data held by the DOs.

Steps of protocol as represented in Figure 3.1.

1. Step 1 (Key Generation): SK and PK are created, and PK flows from the DC to the DOs and the Server.
2. Step 2 (Data Encryption): Using the PK, DOs encrypt their data.
3. Step 3 (Data Transfer): The encrypted data is transferred to the server.
4. Step 4 (Computation): The Server performs the required computations once all the encrypted data is received.
5. Step 5 (Result Transfer): The encrypted results of the calculation are transferred to the DC.
6. Step 6 (Decryption): The DC decrypts the result using the SK, and the final output is displayed.

4 Data Flow and Processing Pipeline of the Architecture Used in This Work

In this section, there will be a step-by-step guideline in how the protocol functions, from keygen to result decryption. First a general workflow, then the different implementations as the differences between pure FHE and transciphering/Hybrid HE are important and need to be highlighted. An algorithmic workflow will also be provided.

4.1 General Data Flow

The system workflow is designed to facilitate secure and privacy-preserving computations on sensitive data. It involves multiple parties, including Data Owners, Data Customers, and a Server, each with distinct roles and responsibilities. Below, we provide a formal explanation of the workflow, which can be visualized in Figure 3.1. A reminder that this full protocol must be followed for each DC and each choicevector is from a specific DO to a specific DC.

Step 1 - Key Generation: Begins with each DC, who initiates a request for a specific dataset and defines the computation to be performed. The DC generates a pair of cryptographic keys: a **public key (PK)** and a **secret key (SK)**. The public key is shared with the DOs and the S, while the secret key is kept private by the DC. This step is critical for ensuring the confidentiality of the data throughout the computation process [1].

Step 2 - Data Encryption by Data Owners: The DOs encrypt their data using the public key provided by the DC. The encryption process ensures that the data remains confidential and cannot be accessed by unauthorized parties, including the Server. Additionally, each DO may apply a **choice vector** $\mathrm{cv_i}$, which determines whether their data will be included in the computation. If a DO chooses not to participate, they may transmit a handshake, acknowledgment, or metadata indicating their decision. This mechanism ensures that individual DOs retain control over their data and who they will send their encrypted data toward [33].
At this point, how the DOs encrypt their data is kept vague, as this will be explained in Section 4.1.1 and 4.1.2, as each version has its way of implementing this.

Step 3 - Data Transfer to the Server: Once the data is encrypted, it is transferred to the S. The transfer process is straightforward, as the data is already encrypted and does not require additional protection during transmission. The size of the transferred data is directly proportional to the size of the encrypted values, ensuring efficient communication. However, for the two systems: Hybrid HE and pure HE, this proportionality is not similar, as will be explored in later chapters, and the Hybrid HE has an expansion factor of 1.

Step 4 - Computation on the Server: Performs the requested computation on the encrypted data using the public key. The computation is performed homomorphically, meaning that the Server does not need to decrypt the data to process it. This step is constrained by the capabilities of the Fully Homomorphic Encryption (FHE) scheme, which may limit the types of operations that can be performed (e.g., no exponentiation or division). For the toy example, the specific calculation is compliant with available operations and will be further explained in Section 6.2 of this Chapter.
Again, the specifics of how this gets processed are left vague in order to be compatible with both versions in the following sections.

Step 5 - Result Transfer to the Data Customer: After the computation is complete, the Server transfers the encrypted result back to the Data Customer. Since the result remains encrypted, no additional encryption is required before or during transmission. It should be noted that this step always transfers the same amount of data, regardless of the number of DOs, and is solely dependent on the requested function/algorithm.

Step 6 - Decryption by the Data Customer: Finally, the Data Customer decrypts the result using their secret key ($\mathrm{SK_{FHE}}$). This step reveals the output of the computation, which is the desired result of the algorithm or function requested by the DC. The notation of this operation is $\mathrm{FHE.Dec}\left(F, \mathrm{SK_{FHE}}\right)$.

Algorithm 3.1 Privacy-Preserving Computation With Choice Vector

Input (DO): Plaintext data v_i, Choice Vector $\mathrm{cv_i}$
Input (Cloud): Algorithm f
Output (DC): Decrypted result $f(v)$

▷ Step 1: Key Generation (At DC)

$(\mathrm{PK_{FHE}}, \mathrm{SK_{FHE}}) \leftarrow \mathrm{FHE.KeyGen}()$
$\mathrm{PK_{FHE}} \rightarrow$ Cloud

▷ Step 2: Data Encryption

for $i \in 1 : \mathrm{length(DO)}$ **do**
    $V_i \leftarrow \mathrm{EncodeData}(v_i, \mathrm{PK_{FHE}})$
end for

▷ Step 3: Data Transfer

for $i \in 1 : \mathrm{length(DO)}$ **do**
    if $\mathrm{cv_i} = 1$ **then**
        $V_i \rightarrow$ Server
    else
        $\mathrm{cv_i} \rightarrow$ Server
    end if
end for

▷ Step 4: Computation

$F \leftarrow \mathrm{ProcessData}(f, \{V_i : \forall i \in 1 : \mathrm{length(DO)} \mid \mathrm{cv_i} = 1\}, \mathrm{PK_{FHE}})$

▷ Step 5: Result Transfer

$\mathrm{PK_{FHE}} \rightarrow$ DC

▷ Step 6: Decryption

$f(v) \leftarrow \mathrm{FHE.Dec}\,(F, \mathrm{SK_{FHE}})$
Return: $f(v)$

4.1.1 Privacy-Preserving System with FHE

In this approach, the Data Owners encrypt their data directly using the Fully Homomorphic Encryption (FHE) scheme. The implementation changes in the following steps.

Step 2 - Data Encryption by Data Owners: Use the public key ($\mathrm{PK}_{\mathrm{FHE}}$) generated by the Data Customer to encrypt their data. The encryption function is defined as: $V_i := \mathrm{FHE.Enc}\,(v_i, \mathrm{PK}_{\mathrm{FHE}})$, where v_i is the plaintext message, and V_i is the ciphertext. It is important that this works like an asymmetric encryption: the Public key can encrypt it, but not decrypt this, only the Secret Key can do this.

Step 4 - Computation on the Server: The Server performs homomorphic operations on the encrypted data with the Public Key without needing the Secret Key. This is a fundamental property of FHE schemes [1]. The main operation that it needs to do can be defined as: $F = \mathrm{FHE.Eval}(f, V, \mathrm{PK}_{\mathrm{FHE}})$ where V is the collection of all participating V_i.

Algorithm 3.2 Encode & Process Without Transciphering

function EncodeData($v_i, \mathrm{PK}_{\mathrm{FHE}}$)
    $V_i \leftarrow \mathrm{FHE.Enc}\,(v_i, \mathrm{PK}_{\mathrm{FHE}})$
    return V_i
end function

function ProcessData($f, V, \mathrm{PK}_{\mathrm{FHE}}$)
    $F \leftarrow \mathrm{FHE.Eval}(f, V, \mathrm{PK}_{\mathrm{FHE}})$
    return F
end function

4.1.2 Privacy-Preserving System with FHE-friendly Block Ciphers

Transciphering combines the transfer size efficiency of symmetric cryptography with the privacy-preserving capabilities of FHE. The implementation changes in the following steps.

Step 2 - Data Encryption by Data Owners:
The DO generates the symmetric key ($\mathrm{SK}_{\mathrm{BC}}$) for the block cipher, then encrypts their data using the block cipher and the symmetric key is encrypted using the FHE scheme.

$$C_i = \mathrm{Symm.Enc}(v_i, \mathrm{SK}_{\mathrm{BC}}) \qquad \mathrm{C}_{\mathrm{key}} = \mathrm{FHE.Enc}(\mathrm{SK}_{\mathrm{BC}}, \mathrm{PK}_{\mathrm{FHE}})$$

Even though this expands the key's size, which is what we avoided in the data, this counts as the meta-data and is assumed to be much smaller than the actual data. As the Keys of a block cipher do not depend on how much the block cipher is used, this will remain the same (expanded) size for many different orders of magnitude of data. While FHE encrypting the original data will expand much more.

Step 4 - Computation on the Server: The Server performs the symmetric decryption circuit in a homomorphic manner to transform the symmetric ciphertext toward a homomorphic ciphertext. From this point on, the steps remain the same and identical as the pure HE counterpart.

$$V_i' = \text{FHE.Enc}(C_i, \text{PK}_{\text{FHE}}) \qquad V_i = \text{FHE.Eval}\left(\text{Symm.Dec}(\cdot, \text{C}_{\text{key}}), V_i', \text{PK}_{\text{FHE}}\right)$$

The Server performs homomorphic computations on V_i as described in Section 4.1: $F = \text{FHE.Eval}(f, V, PK_{FHE})$ where V is the collection of all participating V_i and analogously, C is the collection of all participating C_i. It should be noted that this V_i should be the same between Hybrid HE and Pure FHE, and apart from the noise this is the case.
This approach reduces the computational overhead of FHE by leveraging the efficiency of symmetric-key cryptography [37].

Algorithm 3.3 Encode & Process With Transciphering

```
function EncodeData(v_i, PK_FHE)
    SK_BC ← GenSymmKeys()
    C_i ← Symm.Enc(v_i, SK_BC)
    C_key ← FHE.Enc(SK_BC, PK_FHE)
    return (C_i, C_key)
end function

function ProcessData(f, (C, C_key), PK_FHE)
    V' ← FHE.Enc(C, PK_FHE)
    V ← FHE.Eval(Symm.Dec(·, C_key), V', PK_FHE)
    F ← FHE.Eval(f, V, PK_FHE)
    return F
end function
```

4.2 Privacy Properties and Threat model

First, it is essential to verify that the protocol satisfies the stated privacy requirements. Specifically, one must assess whether any entity other than the Data Customer is capable of reconstructing the output and whether the private input of any Data Owner can be inferred at any stage of the pipeline. Table 3.1 summarizes the access rights and data exposure levels for each participant. As the table demonstrates, neither the Server nor other DOs are granted access to sensitive input or output data. Consequently, the protocol upholds the required confidentiality guarantees for both the input and the output. Our privacy-preserving system incorporates the properties and threat models to ensure robust security and privacy guarantees:

- Input Privacy: The Data Owners can choose to participate through the choice vector cv_i.
- Confidentiality: Achieved through FHE and transciphering, which ensure that data is encrypted at all stages of computation.

Table 3.1: Data allocation scheme

(a) Without Transciphering

Parameter	DO_i	S	DC
PK_{FHE}	✓	✓	✓
SK_{FHE}	✗	✗	✓
v_i	✓	✗	✗
cv_i	✓	✓	✗
$FHE.Enc(v_i \cdot cv_i)$	✓	✓	✗
F	✗	✓	✓
$FHE.Dec(F)$	✗	✗	✓

(b) With Transciphering

Parameter	DO_i	S	DC
PK_{FHE}	✓	✓	✓
SK_{FHE}	✗	✗	✓
v_i	✓	✗	✗
cv_i	✓	✓	✗
$Symm.Enc(v_i \cdot cv_i)$	✓	✓	✗
SK_{BC}	✓	✗	✗
$FHE.Enc(SK_{BC})$	✓	✓	✗
F	✗	✓	✓
$FHE.Dec(F)$	✗	✗	✓

- Output Privacy: Ensured by decrypting the results only at the Data Customer, using their secret key.
- Anonymity: Achieved through not connecting the DCs directly to the DOs, which protects the identities of the Data Owners and the Data Customers.
- Access Control: Implemented through the choice vector, where every DO can decide which DCs will have access to their (protected) data.

The system can operate under both the semi-honest and malicious models, depending on the level of trust placed in the participants. For example:

- In the semi-honest model, the Server is assumed to follow the protocol but may attempt to infer sensitive information from the encrypted data.
- In the malicious model, additional safeguards, such as zero-knowledge proofs, could be implemented to ensure that the Server cannot tamper with the data or computation.

We will therefore assume the semi-honest model for the remainder of this paper.

5 Variants of the Protocol

It is interesting to ponder slightly differing versions even though they will not progress throughout this paper. It might inspire future research with an interesting problem. This paper will name the default (as seen in previous, this, and further Sections and explained in Section 3 of this chapter (3)), as the "Choice Vector" or V1 method. The other variants will focus on the way and when the DOs share their encrypted data.
In this section, we analyze the privacy properties of the three computation methods presented in the previous algorithms. Specifically, we evaluate each method based on:

1. Input Privacy: The confidentiality of the data provided by the Data Owners.

2. Output Privacy: The confidentiality of the final computation result at the Data Customer.
3. Anonymity: Whether the system hides which users are involved in the computation.
4. Access Control: Whether the system enforces restrictions on which data the Data Customer can access, and from which DOs.

5.1 V2: Privacy-Preserving Protocol Without Choice Vector

This method encrypts all input data using Fully Homomorphic Encryption, without regard to whether the DO wants to participate in this function, for the corresponding DC.

- Input Privacy - Strong: Data is encrypted with FHE before being sent to the Server. The Server never sees plaintext data during computation.

- Output Privacy - Strong: The computation result is encrypted under FHE and can only be decrypted by the DC with the FHE secret key.
  No other entity, including the Server, can infer information from intermediate results.

- Anonymity - Weak: The Server knows which Data Owners contribute data.
  Even though the server cannot see the plaintext values, the presence of encrypted data reveals participation.

  Additionally, because there does not exist a Choice Vector implementation, every DO is required to send data.

- Access Control - Absent/Weak: All encrypted data is processed by the server, meaning the DC can retrieve results for all Data Owners.
  There is no explicit mechanism to restrict access to specific data owners' inputs.

This approach is best suited for high-security computations where all data must be processed. However, this method lacks access control mechanisms: The Data Customer receives the result of computations over all Data Owners' inputs, regardless of whether individual consent was given or certain inputs should have been excluded.

5.2 V3: Privacy-Preserving Protocol With Trivial Value Insertion

This method replaces excluded data values with a trivial placeholder (e.g., zero or noise) instead of omitting them. This value can be defined by the function that will be run on the Server side to ensure a reasonable computation will happen. As with the Variant defined in Section 3, there is the presence of a choice vector cv_i to determine which DC each DO wants to send data to.

Algorithm 3.4 Privacy-Preserving Protocol V2

Input (DO): Plaintext data v_i
Input (Cloud): Algorithm f
Output (DC): Decrypted result $f(v)$

▷ Step 1: Key Generation (At DC)

$(\text{PK}_{\text{FHE}}, \text{SK}_{\text{FHE}}) \leftarrow \text{FHE.KeyGen}()$
$\text{PK}_{\text{FHE}} \rightarrow \text{Cloud}$

▷ Step 2+3: Data Encryption + Transfer

for $i \in 1 : \text{length(DO)}$ **do**
    $V_i \leftarrow \text{EncodeData}(v_i, \text{PK}_{\text{FHE}})$
    $V_i \rightarrow \text{Server}$
end for

▷ Step 3: Computation

$F \leftarrow \text{ProcessData}(f, \{V_i : \forall i \in 1 : \text{length(DO)}\}, \text{PK}_{\text{FHE}})$

▷ Step 4: Result Transfer

Transfer F to Data Customer (DC)

▷ Step 5: Decryption

$f(v) \leftarrow \text{FHE.Dec}(F, \text{SK}_{\text{FHE}})$
Return: $f(v)$

- Input Privacy - Strong: Data is still encrypted before being sent to the Server . However, trivial values replace real data where access is restricted, reducing potential information leaks.

- Output Privacy - Moderate to Strong: The output remains encrypted under FHE, ensuring confidentiality.
  However, trivial values affect the final result, meaning that some statistical leakage could occur if an adversary notices that computations are being performed on zero/noise values. Some functions are not possible, such as taking the average and assuming the Trivial value is 0, the denominator still grows.

- Anonymity - Stronger: Unlike the Choice Vector method (Section 3), the Server does not know which users contributed real data because all data positions are filled (even if some are trivial).
  This hides user participation, making the scheme more anonymous.

- Access Control - Strongest: Since all data slots contain encrypted values (real or trivial), the DC cannot infer which values were valid.
  This prevents access patterns from leaking which Data Owners contribute meaningful data.

Privacy Trade-offs

This variant offers strong anonymity and access control, as the server cannot distinguish between genuine and dummy inputs. All encrypted values appear structurally identical, which enhances privacy by obfuscating participation. Nonetheless, the computational overhead is increased, since the system must encrypt and process dummy values alongside real data. Additionally, if not handled carefully, trivial values may influence the correctness of the computed result, especially in aggregate statistics, if they are not adequately filtered or normalized.

Algorithm 3.5 Privacy-Preserving Protocol V3

Input (DO): Plaintext data v_i, Choice Vector $\mathrm{cv_i}$
Input (Cloud): Algorithm f, Trivial Value T
Output (DC): Decrypted result $f(v)$
▷ Step 1: Key Generation (At DC)
$(\mathrm{PK_{FHE}}, \mathrm{SK_{FHE}}) \leftarrow \mathrm{FHE.KeyGen}()$
$\mathrm{PK_{FHE}} \rightarrow$ Cloud
▷ Step 2+3: Data Encryption + Transfer
for $i \in 1 : \mathrm{length(DO)}$ **do**
    if $CV_i = 1$ **then**
        $V_i \leftarrow \mathrm{EncodeData}(v_i, \mathrm{PK_{FHE}})$
    else
        $V_i \leftarrow \mathrm{EncodeData}(T, \mathrm{PK_{FHE}})$
    end if
    $V_i \rightarrow$ Server
end for
▷ Step 4: Computation
$F \leftarrow \mathrm{ProcessData}(f, \{V_i : \forall i \in 1 : \mathrm{length(DO)}\}, f, \mathrm{PK_{FHE}})$
▷ Step 5: Result Transfer
Transfer F to Data Customer
▷ Step 6: Decryption
$f(v) \leftarrow \mathrm{FHE.Dec}(F, \mathrm{SK_{FHE}})$
Return: $f(v)$

Conclusion

Table 3.2: Comparison Table of Privacy Properties

Protocol	Input Privacy	Output Privacy	Anonymity	Access Control
V1 (Section 3)	Strong	Strong	Moderate	Moderate
V2 (Section 5.1)	Strong	Strong	Weak	Absent
V3 (Section 5.2)	Strong	Moderate	Strong	Strong

Each method provides a different trade-off between input privacy, output privacy, anonymity, and access control:

Therefore, each method has its optimal use-case: Use our primary protocol when certain data should be hidden from computations, but efficiency is still important and it doesn't matter that the Server knows the choice of each DO. The non-choice vector can be used when **all** data needs to be computed or the system can assume that each DO will agree and finally, the Trivial Value Insertion method can be used when maximum anonymity and access control are needed, at the cost of extra computation and transfer cost.

Thresholding Based on Participant Count
A widely acknowledged method for preventing reconstruction attacks and protecting individual privacy is the use of *thresholding based on participant count*. In this approach, encrypted computations are only performed or released if the number of contributing data owners exceeds a predefined threshold T.
Formally, let n denote the number of active or valid data contributions in a given aggregation task, This is equal to the amount of DOs with a Choice Vector of 1. Then, the computation is conditionally executed only if $n \geq T$, where T is a security or privacy threshold. If this condition is not met, the computation is aborted, or a null result is returned.
This thresholding model is crucial in settings such as:

- Federated learning, where individual user updates should not dominate the global model.
- Secure multi-party analytics, where contributions from fewer than T sources could lead to deanonymization.
- Privacy-preserving surveys or statistics, where a minimum sample size is required to uphold privacy guarantees [38, 39].

Systems implementing thresholding typically use secure counters or multi-party protocols to verify participation without compromising contributor identities. This mechanism adds robustness to FHE pipelines by ensuring that computation only proceeds under adequate anonymity and data density.

6 Use Cases of Privacy Preserving System

While the preceding, current, and future chapters detail the architectural, cryptographic, and performance aspects of the proposed system, it is equally important to contextualize its applicability in real-world domains. This section highlights concrete scenarios in which privacy-preserving computation can be effectively deployed. The discussion is divided into two parts: a high-level overview of common domains that benefit from secure multiparty computation, and a more focused analysis of a specific mathematical function used within this work. These examples serve to illustrate both the flexibility and relevance of the proposed system.

6.1 General Use cases

The privacy-preserving system described in this work has a wide range of applications, particularly in domains where data sensitivity and confidentiality are paramount. Below, we discuss key use cases and after this, the paper will evaluate Use cases when implementing one specific evaluation function, which will be benchmarked in Chapter 4.

- Medical Data Analysis: In the healthcare sector, patient data is highly sensitive and subject to strict privacy regulations (e.g., GDPR, HIPAA). The system enables secure analysis of medical data without exposing individual patient records [40]. For example:
  - Disease Prediction: Hospitals can share encrypted patient data with researchers to develop predictive models for diseases without revealing identifiable information. This approach aligns with privacy-preserving machine learning techniques [33].
  - Clinical Trials: Pharmaceutical companies can analyze encrypted data from multiple healthcare providers to evaluate the efficacy of new treatments while maintaining patient privacy. This is particularly important for multi-party collaborations in clinical research [37].
- Big Data Processing: In the era of big data, organizations often need to process large datasets from multiple sources. The system allows for secure aggregation and analysis of such data:
  - Financial Data: Banks and financial institutions can securely analyze transaction data to detect fraud or assess market trends without exposing sensitive customer information. This is a key application of FHE in the financial sector [36].
  - IoT Data: Companies can process encrypted data from IoT devices to monitor and optimize operations while ensuring that the data remains confidential. This is particularly relevant for smart cities and industrial IoT applications [41].
- Machine Learning Model Building: Privacy-preserving machine learning techniques, such as federated learning, allow multiple parties to collaboratively train models on distributed datasets without sharing raw data [42]. Or for Data-Owners to 'rent' data instead of selling it completely: interested parties can then train their models on this rented data for a cheaper price, without retaining the data.
  - Federated Learning: Multiple organizations can collaboratively train machine learning models on their encrypted data without sharing the raw data. This approach is a key enabler of privacy-preserving AI [33].
  - Privacy-Preserving AI: Companies can build AI models on encrypted customer data, ensuring that sensitive information is not exposed during the training process. This is particularly important for applications in personalized healthcare and finance [37].

- Secure Multi-Party Computation: The system can also be used for secure multi-party computation (MPC), where multiple parties wish to compute a function over their combined data without revealing their individual inputs:
  - Supply Chain Optimization: Companies in a supply chain can collaboratively optimize logistics and inventory management without sharing proprietary data. This is a key application of MPC in business operations [41].
  - Voting Systems: The system can be used to implement secure electronic voting systems, where votes are encrypted and tallied without revealing individual choices. This ensures the integrity and confidentiality of the voting process [36].

6.2 Use Cases of Specific Function in a Privacy Preserving Setting

In this section, we consider the practical relevance and real-world applications of the following computation performed in a privacy-preserving manner. We will then evaluate different FHE-friendly block ciphers on their performance within the Privacy-Preserving System:

$$\sum_i (M \cdot v_i + b)$$

Here, each $v_i \in \mathbb{Z}^d$ is a private data vector held by Data Owner i, $M \in \mathbb{Z}^{k \times d}$ is a transformation matrix, and $b \in \mathbb{Z}^k$ is a fixed bias vector. The operation consists of applying a linear transformation and offset to each private input and then aggregating the results across multiple users. When performed under Fully Homomorphic Encryption (FHE), the entire process can be carried out without revealing any intermediate or individual data points, thus preserving data confidentiality throughout [1, 43].
Despite its power and versatility, the function $\sum_{i=0}^{N}(M \cdot v_i + b)$ also introduces a subtle risk: namely, the possibility of misusing the transformation matrix M and bias vector b to infer individual inputs.
If a malicious or overly curious evaluator is allowed to choose the transformation matrix $M = I_d$ (the identity matrix) and the bias vector $b = 0$, the computation reduces to:

$$\sum_{i=0}^{N} v_i$$

In such a case, if only a small number of data owners are included, or if the evaluator computes this aggregate multiple times while selectively including or excluding specific contributors, it becomes possible to isolate and reconstruct individual v_i values. This is a well-known risk in privacy-preserving data analysis and underscores the importance of access control or cryptographic accountability mechanisms [39, 44].
Therefore, while the transformation matrix provides useful flexibility, its configuration should be carefully managed and restricted in practical deployments to ensure that aggregation cannot be weaponized for reconstruction attacks.

This function has broad applicability across domains that require secure, decentralized data analytics. Below, we elaborate on several motivating scenarios where such a computation would be valuable.

Medical Data Aggregation
In healthcare, hospitals, clinics, or laboratories may each hold sensitive patient data. Often, there is a need to compute population-level metrics such as disease risk scores, aggregated biomarker levels, or early-warning indicators based on collective data [44].
Let, for example, v_i denote a patient's medical feature vector, such as cholesterol levels, blood pressure, and age. A linear transformation matrix M could correspond to weights in a risk prediction model (e.g., the Framingham risk score [45]), and b could encode base risk factors or thresholds.
The function $M \cdot v_i + b$ computes an individual risk estimate, and summing this across all patients allows the medical system to derive an encrypted aggregate measure of risk across the entire cohort. This can be used to guide public health decisions or allocate resources, without revealing any individual patient's data.
This would be an invaluable tool in order to get accurate and useful results without violating Data-privacy rules with regards to patients.

Privacy-Preserving Financial Scoring
Another important application domain is the financial sector. Multiple banks or credit institutions may wish to compute the average creditworthiness of their clients or assess cumulative risk without exposing individual financial histories [46].
Here, v_i could encode features such as credit history, debt-to-income ratio, or spending behavior. The matrix M represents a set of model coefficients for a credit scoring algorithm, while b may represent regulatory thresholds or institutional bias corrections. The computation yields encrypted scores for each customer, and their sum can be used to evaluate the overall exposure to financial risk across the entire system.
By using FHE or transciphering approaches, the final result can be decrypted and interpreted by the regulator or authorized auditor, while the sensitive raw features remain fully encrypted [47].
Or, when applied to specific regions by either tax bracket or geographical locations, this would be a useful scoring algorithm to determine where specific risks reside.

Smart Infrastructure and Urban Analytics
In the context of smart cities, households or sensors may continuously produce privacy sensitive data such as electricity consumption, water usage, or mobility traces [5].
Local devices can encode this data into a feature vector v_i, which is then processed homomorphically to compute transformed sustainability metrics or demand indicators.
The matrix M in this case could represent a transformation into a target metric space (e.g., per capita energy impact, carbon emissions), and b could reflect baseline allowances or incentive thresholds. The sum of transformed vectors across households allows a municipal authority to assess the overall system load, efficiency, or carbon footprint, all while respecting the privacy of individual households.
Another useful use-case is to gather information within a specific space, i.e. a library or study-room. And gather data on IOT devices and personal devices, and then process

this data off-site to get an indication of occupancy or factors that indicate revisiting the location.

Federated Data Analysis and Secure Statistics
In federated learning or secure data analysis frameworks, the aggregation of transformed local statistics is a fundamental building block [48]. Each client or data owner contributes local statistics or feature representations v_i, and the system computes a global summary in the form of a sum of linearly transformed inputs.
This is particularly relevant when performing secure initialization (e.g., mean-centering features), and federated PCA. By keeping v_i encrypted, the system ensures that only the aggregated result is visible, mitigating the risk of reverse-engineering individual data contributions.

Interpreting the Numerical Representation
The data vectors v_i in this formulation may represent any structured numerical data, demographics, temporal activity logs, feature encodings, or device statistics. The transformation matrix M allows encoding a wide range of operations, including feature scaling, projection into a lower-dimensional space, or application of linear models. The bias term b introduces flexibility to shift or align outputs to desired thresholds or categories.
The outer summation serves to accumulate these transformed values over all contributors, thus enabling the system to compute meaningful aggregated outputs, such as totals, averages (after normalization), or policy compliance scores. Crucially, when this computation is implemented using FHE or transciphered methods, it supports rigorous confidentiality guarantees without compromising utility.

Risk Factor Modeling for Crime Based on Vehicle Characteristics
Another application of secure data aggregation could involve modeling the likelihood of criminal behavior based on attributes of owned vehicles. In such a system, each Data Owner (e.g., vehicle registration authority or insurance provider) holds private vectors v_i describing characteristics of a vehicle, such as make, model, horsepower, engine size, age, and modifications. These can be combined with characteristics of the driver/owner of the vehicle such as age, duration of drivers license, et cetera.
The matrix M encodes the weightings of these attributes in relation to different classes of criminal risk, which could be derived from statistical correlations or prior law enforcement data. For example, certain vehicle types may statistically correlate with higher incidence of traffic violations, street racing, or use in illicit transport activities. The vector b introduces baseline adjustment factors or legal thresholds.
By performing the transformation $M \cdot v_i + b$ and aggregating across many owners, the system computes an encrypted, population-level risk profile without revealing any individual vehicle information. This can be useful for:

- Enabling secure urban planning or traffic policy adjustments.
- Informing risk-based pricing models for insurance companies.
- Generating heatmaps of elevated vehicular risk without surveillance overreach.

Importantly, since the data remains encrypted during processing, such evaluations can be made without violating individual privacy, thus maintaining public trust while leveraging predictive modeling for public safety.

Conclusion

The computation $\sum_{i=0}^{N}(M \cdot v_i + b)$ is a core building block in many real-world secure analytics applications. It combines linear algebra with privacy-preserving cryptography to enable secure, aggregate computation across untrusted parties. The modularity of the transformation matrix M and bias vector b allows it to be adapted for various use cases, including healthcare, finance, infrastructure, and federated systems. However, care must be taken to prevent adversarial configurations of M and b, and thresholding mechanisms can be deployed to further safeguard data privacy [1, 44, 23].

7 Conclusion and Expectations of Implementation

In this chapter (3), we introduced the conceptual and technical foundations of Privacy-Preserving Systems, detailing the architecture, participating entities, and the cryptographic primitives that enable secure collaborative computation. Through the use of Fully Homomorphic Encryption (Chapter 2, Section 3, [1]), transciphering (Chapter 2, Section 5, [5]), and FHE-friendly block ciphers (Chapter 2, Section 4), we have constructed a system that supports efficient and secure evaluation of functions over sensitive, distributed data sources.

We explored the motivations for such systems in the context of rising data privacy concerns and regulatory requirements and demonstrated how our design enables computations to be executed without exposing raw data to untrusted intermediaries. By combining symmetric encryption with homomorphic evaluation, the transciphering pipeline reduces bandwidth and data owner computation costs while preserving strong privacy guarantees [49].

While the theoretical design of this system demonstrates privacy robustness and scalability potential, practical adoption depends on concrete performance characteristics. Therefore, in the following chapter (4), we shift our focus to the empirical evaluation of the system. This includes:

- Measuring the time and memory footprint of each system component, as Section 2 and 3
- Comparing different transciphering configurations (e.g., Pasta variants), as Section 3
- Analyzing scalability with respect to the number of participating data owners, as Section 2 and 3.
- Benchmarking transciphered pipelines against pure FHE implementations, as per Section 3

These numerical results serve to validate the feasibility of the proposed system and highlight the trade-offs between computational overhead, resource usage, and data protection. They also inform our recommendations for real-world deployment scenarios and future optimization efforts.
With the system architecture now fully defined, we proceed to quantify its behavior through comprehensive benchmarking and analysis.

Chapter 4

Experimental Results and Evaluation

This chapter turns theory into practice by presenting empirical benchmarks that measure runtime, memory consumption, and communication overhead across various system configurations. We begin by explaining our choice of cryptographic library and implementation details. The core of this chapter systematically compares pure FHE pipelines against transciphering pipelines using different FHE-friendly block ciphers (e.g., Pasta2-3, Pasta-3, Pasta2-4, and even Hera-5). Through bar plots and normalized ratios, we illustrate how performance scales with the number of Data Owners, analyze per-DO transfer times, and highlight server-side bottlenecks. These results validate the trade-offs between reduced ciphertext expansion and increased computational cost, ultimately guiding future optimization efforts.
To evaluate the practical viability of the proposed privacy-preserving system, this chapter presents an in-depth performance analysis based on a series of targeted benchmarks. The aim is to validate the theoretical benefits of transciphering, such as reduced data size and communication overhead, while assessing the computational trade-offs introduced by homomorphic evaluation. We consider various configurations of the system, including different block cipher implementations and degrees of user participation, to measure performance metrics such as encryption time, server-side computation cost, memory usage, and scalability with respect to the number of Data Owners. The results provide insight into the system's real-world applicability and help identify performance bottlenecks that future work may address.

1 Frameworks and Libraries

Before presenting the numerical results, it is essential to establish the technological foundations on which the experiments were conducted. The choice of cryptographic library plays a critical role in the accuracy, efficiency, and reproducibility of the benchmark results. The following section outlines the rationale behind selecting the Hybrid-HE framework [5] as the core library for this implementation and evaluation effort with some assistance from the OpenFHe library [12].
The development of homomorphic encryption can be viewed as a series of four distinct "generations," each marked by a key conceptual advance (Table 4.1). The first generation began with Gentry's seminal 2009 construction [1], which demonstrated that arbitrary computation on encrypted data was possible via a procedure known as bootstrapping.

Table 4.1: Generations of Fully Homomorphic Encryption (FHE)

Generation	**Key Contribution**	**Reference**
1st Generation	Gentry's breakthrough: First FHE scheme.	[1]
2nd Generation	Leveled FHE schemes (BGV, BFV).	[2, 22]
3rd Generation	Efficient schemes for real numbers (CKKS).	[23, 50]
4th Generation	Practical implementations (HElib, SEAL, TFHE).	[11, 10, 36]

While this breakthrough established the feasibility of Fully Homomorphic Encryption, its reliance on squashing and recursive bootstrapping imposed prohibitive computational costs.
The second generation (circa 2012) dispensed with Gentry's onerous "squashing" step by introducing leveled-FHE schemes, most notably BGV [2] and BFV [22]. These schemes leveraged modulus-switching and key-switching techniques to control noise growth without repeated bootstrapping, significantly reducing the overhead for circuits of known depth. Around 2016–2017, the third generation brought approximate arithmetic into the picture with the CKKS scheme [23, 50], enabling efficient evaluation of real-valued functions at the cost of small, controllable approximation errors. Finally, we have entered a fourth generation characterized by highly optimized, practical implementations (e.g., HElib [11], OpenFHE [12], SEAL [10], TFHE [51]) and the integration of FHE with FHE-friendly block ciphers. These developments have collectively transformed FHE from a theoretical curiosity into a toolkit that can, albeit with careful parameter selection, be deployed for real-world privacy-preserving applications. Another visualization of this timeline with additional FHE-friendly block ciphers is given in Figure 2.3 within Chapter 2, Section 4.

Table 4.2: Libraries and Frameworks for Homomorphic Encryption [52]

Name	**BGV** [2]	**BFV** [22]	**CKKS** [23]	**TFHE** [51]	**FHEW** [53]	**Description**
HElib	✓	✗	✗	✗	✗	Library for BGV scheme.
SEAL	✗	✓	✓	✗	✗	Implements BFV and CKKS schemes.
PALISADE	✓	✓	✓	✗	✓	Supports BGV, BFV, and CKKS schemes.
TFHE	✗	✗	✗	✓	✗	Fast FHE over the torus with bootstrapping.
FHEW	✗	✗	✗	✗	✓	Focuses on efficient bootstrapping.
HEAAN	✗	✗	✓	✗	✗	Implements CKKS for approximate arithmetic.
cuHE	✓	✗	✗	✗	✗	GPU-accelerated BGV implementation.
Lattigo	✓	✓	✓	✗	✗	Go library for BGV, BFV, and CKKS.
OpenFHE	✓	✓	✓	✗	✓	Unified library for BGV, BFV, CKKS, and more.
Liberate	✓	✓	✓	✗	✗	Focuses on performance and usability.

OpenFHE [12] was chosen as a complementary library for this project due to its comprehensive support for multiple Fully Homomorphic Encryption (FHE) schemes, active development, and strong community backing. Below are the key reasons for selecting

OpenFHE:

- Support for Multiple Schemes:
  OpenFHE supports a wide range of FHE schemes, including BGV, BFV, and CKKS, making it highly versatile for different types of computations (exact and approximate arithmetic).
- Active Development:
  OpenFHE is actively maintained and updated, ensuring compatibility with the latest advancements in FHE research and practice.
- Performance and Usability:
  The library is optimized for performance and provides a user-friendly API, making it accessible for both researchers and developers. Besides, this library also exposes it's calls to a python implementation which should be easier to work with.
- Community and Documentation:
  OpenFHE has a strong community and extensive documentation, which simplifies the learning curve and provides robust support for troubleshooting.
- Interoperability:
  OpenFHE is designed to be interoperable with other cryptographic libraries, enabling seamless integration into existing systems.

For these reasons, OpenFHE is well-suited for implementing privacy-preserving systems that require flexibility, performance, and ease of use. However, it might not be most suitable to benchmark down to the lower code parts and the Hybrid method is not yet implemented.
In addition to utilizing OpenFHE as a foundation for implementing and analyzing Fully Homomorphic Encryption schemes, this thesis also integrates the Hybrid-HE benchmarking framework of the Pasta-paper [5] for comparison and performance evaluation. The inclusion of Hybrid-HE serves several strategic and technical purposes that complement the objectives of this research.
The Hybrid-HE framework [5] provides a dedicated infrastructure for evaluating homomorphic and transciphered pipelines in a modular, extensible, and platform-agnostic manner. Designed with benchmarking in mind, it enables a granular breakdown of computational steps and resource consumption across key phases of a privacy-preserving pipeline, including symmetric encryption, homomorphic evaluation of decryption, and encrypted computation.
There are several motivations for adopting this framework in parallel with OpenFHE:

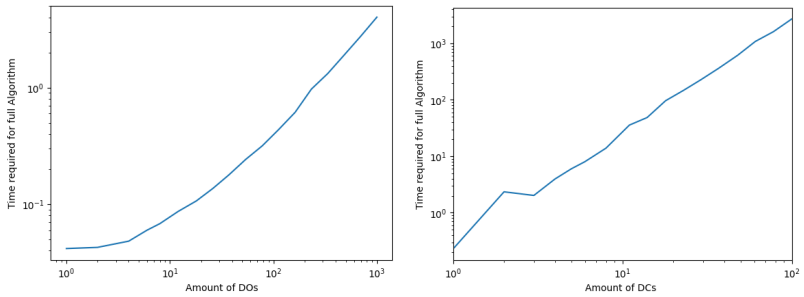
- Specialization to Transciphering: Hybrid-HE was specifically engineered to evaluate hybrid encryption scenarios involving symmetric ciphers like Pasta in conjunction with FHE backends. As this thesis focuses heavily on transciphering performance, the toolchain aligns directly with the required experimental goals.
- Comparative Performance Metrics: Hybrid-HE supports detailed timing analysis, enabling side-by-side benchmarking of hybrid, and pure-FHE implementations. This is critical for drawing quantitative conclusions about trade-offs in runtime and memory usage under different encryption paradigms.

- Cipher Modularity: The library provides plug-and-play support for different FHE-friendly block ciphers (e.g., Pasta, Hera, agrasta, rasta, etc.), facilitating experimentation with alternative designs and parameter sets. This flexibility accelerates the process of testing and validating new cipher configurations.

Overall, this framework serves as a vital complement to OpenFHE in this thesis, enabling rigorous benchmarking of privacy-preserving systems and helping to characterize their behavior under various cryptographic configurations and workloads. The combination of these tools enhances both the correctness and the empirical reliability of the results presented in the following sections.
However, for valuable benchmarks within this paper, this framework has undergone heavy modifications to make sure it generates correct results such as an abstraction for DO, S, and DC and a change to the core evaluation model to fit our example: $\sum_{i=1}^{\#DO} Mv_i + b$ where v_i is encrypted.

2 Toy Example Using OpenFHE

To establish a performance baseline for our hybrid transciphering pipeline, we first implement and benchmark a minimal "pure-FHE" toy example in OpenFHE. In this simplified scenario, each Data Owner holds 8 confidential values v_i, and the Data Customer k wishes to compute the numerator of the weighted sum: $\sum_{i=1}^{\#DO} \sum_{j=1}^{8} w_j \left(v_{i,j} \cdot \mathrm{cv_{i}}_j\right)$ where $\mathrm{cv_{i}}_j$ denotes the choice vector of DO i toward DC j. No block-cipher transciphering is used, the DOs simply encrypt their inputs with the FHE public key, the Server homomorphically evaluates the sum, and the DC decrypts the result.
This toy example exercises the full Privacy-Preserving System workflow: key generation, encryption, homomorphic evaluation, and decryption. Because the FHE-friendly Block Cipher will be studied in Section 3 of this Chapter. As such, the bfv scheme will also be used, with as close to similar parameters. However, there is no point in exactly comparing results as the underlying libraries are very different and therefore not comparable. Additionally, timing results should be taken with a pinch of salt as OpenFHE is under a Python layer.
Varying Number of DOs: We keep the number of DCs fixed at one and vary the number of participating DOs between 1 and 1000. Figure 4.1a shows that runtime grows linearly with the number of DOs. We expect the same trend to appear within the results of the Hybrid-HE library [5] in Section 3. The interesting part will be found in that section as we will explore the time requirement of each party, and this will answer an important question: whether assuming that each DO is computing consecutively or concurrently will matter at all.
Varying Number of DCs: We fix 50 DOs and measure the end-to-end time as the number of distinct DCs increases from 1 to 100. As illustrated in Figure 4.1b, the growth in time per DC remains approximately the same, as the whole protocol needs to be redone. This can be attributed to the fact that each DC needs to create its keypair and therefore needs to redo the whole operation.



(a) Timing with respect to DOs (b) Timing with respect to DCs

Figure 4.1: Timing results of OpenFHE

3 Performance and Scalability Analysis

After establishing the experimental setup and cryptographic framework, this section presents a detailed analysis of the system's performance across multiple dimensions and parties. The evaluation focuses on runtime efficiency, memory usage, and the scalability of the system with respect to the number of Data Owners. These aspects are essential in determining the system's feasibility for large-scale deployment, especially in real-world environments where communication, computation, and memory are critical bottlenecks. The metrics collected reflect both the overhead imposed by homomorphic operations and the trade-offs introduced by transciphering techniques. Results are drawn from repeated empirical measurements and presented using comparative bar plots and normalized ratios to highlight key patterns.

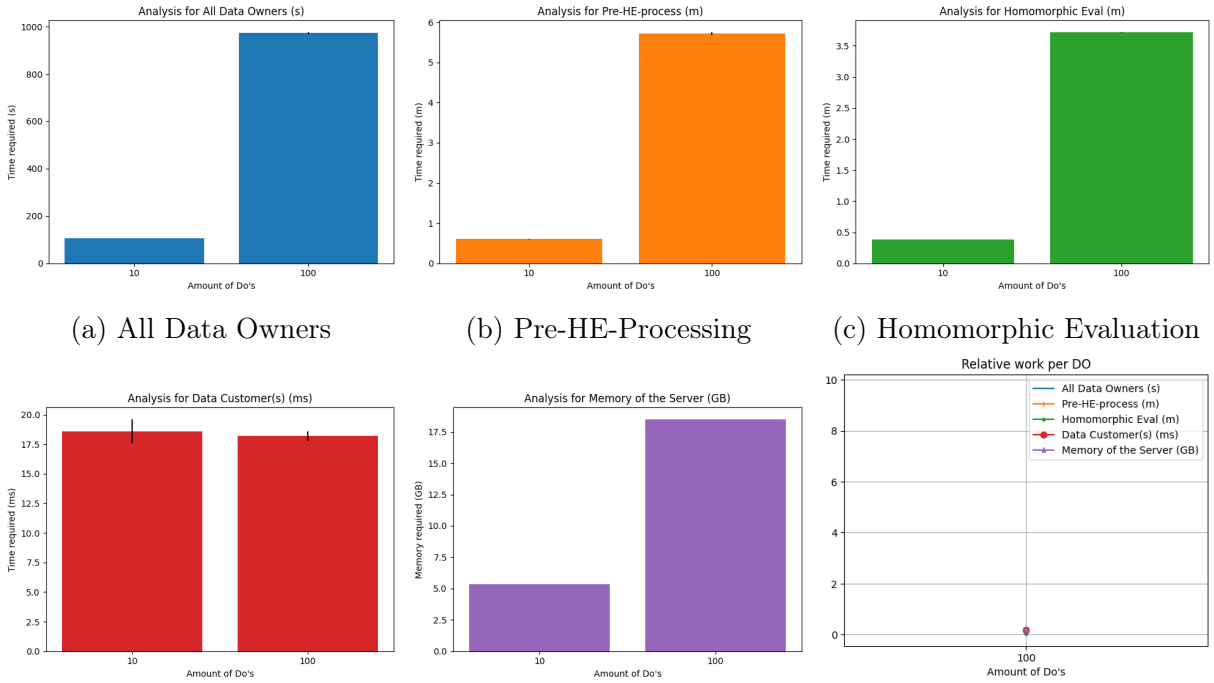
3.1 Scalability of Each Party

To rigorously evaluate the computational behavior of our transciphered privacy-preserving system, we measured runtime and memory usage across increasing numbers of Data Owners. These results are summarized in six barplots, each capturing a different performance metric across the system's participants and components. All measurements were conducted with multiple repetitions to ensure reliability, and standard error values (represented as small black bars) are included in each plot to depict the timing variance due to external system factors or process scheduling.

Each of the first five barplot's uses the number of participating DOs as the x-axis, while the y-axis quantifies a specific resource: time (in milliseconds, seconds, or minutes) or memory (in gigabytes) as denoted within the graph. The sixth plot aggregates and normalizes the data by dividing each measurement by the number of DOs, yielding a per-DO cost profile that reveals whether each subsystem scales linearly or remains constant in practice. The y-axis units differ across plots, and careful attention must be paid to the varying scales.

In order to further understand the trade-offs in transciphering performance, we conducted comparative experiments using two different configurations of the Pasta cipher, `pasta2_3` and `pasta_3`, alongside a baseline implementation that used no FHE-friendly block cipher (i.e., direct symmetric encryption followed by homomorphic encryption). See Figure 4.3, 4.4 and 4.2 for respectively the `pasta_3`, `pasta2_3` and Only Homomorphic Encryption,

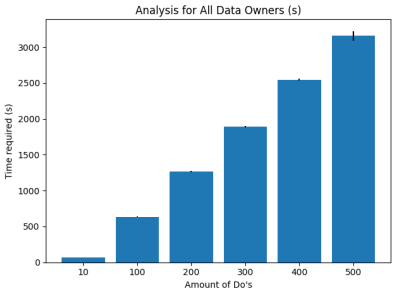
the focus and explanation of the following section can best be guided by Figure 4.4, but the trend should be a bit similar to the others.
Please note that there are additional tests available, such as Pasta2-4 which has an additional round, and Hera-5 which is a completely different FHE-friendly block cipher and have been appended to the Appendix as this already took a lot of space.



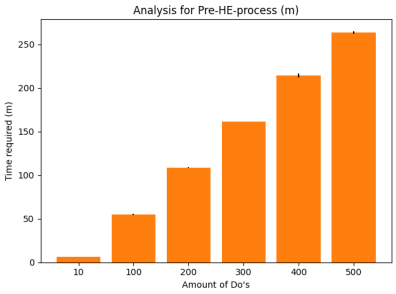
(a) All Data Owners (b) Pre-HE-Processing (c) Homomorphic Evaluation

(d) Data Customer(s) (e) Memory of the Server (f) Scale with DO

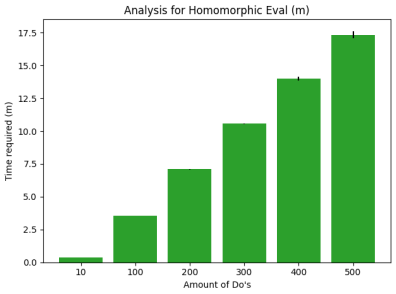
Figure 4.2: Results for 1 DC, Cipher: `None`, mod degree: 32768



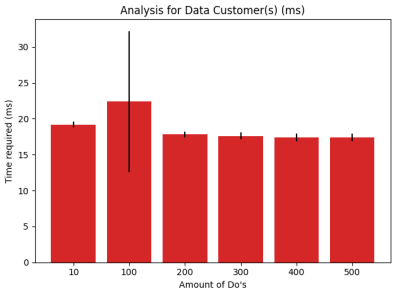
(a) All Data Owners



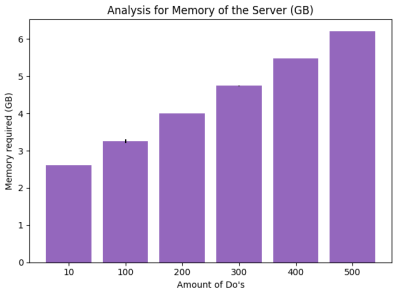
(b) Pre-HE-Processing



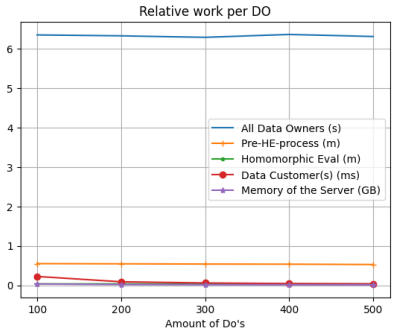
(c) Homomorphic Evaluation



(d) Data Customer(s)

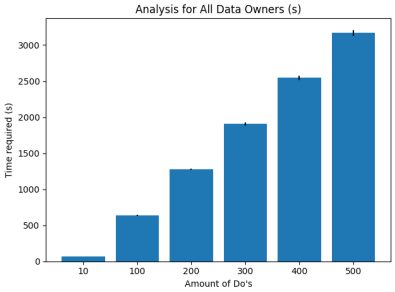


(e) Memory of the Server

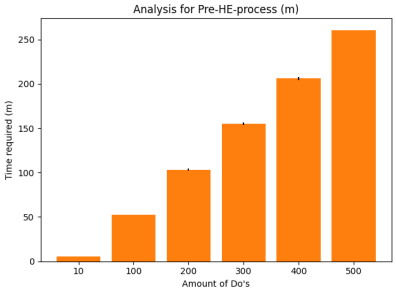


(f) Scale with DO

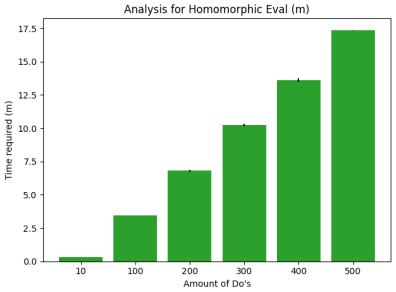
Figure 4.3: Results for 1 DC, Cipher: `Pasta-3`, mod degree: 32768



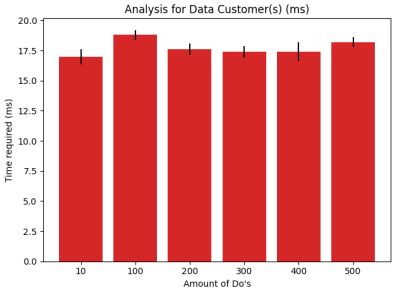
(a) All Data Owners



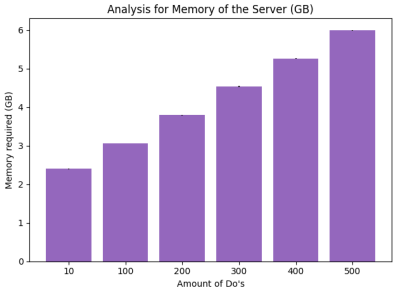
(b) Pre-HE-Processing



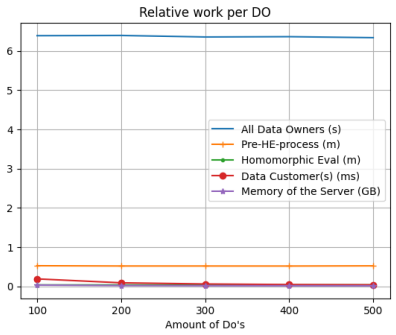
(c) Homomorphic Evaluation



(d) Data Customer(s)



(e) Memory of the Server



(f) Scale with DO

Figure 4.4: Results for 1 DC, Cipher: `Pasta2-3`, mod degree: 32768

Graph (a): Total Time Cost Across All Data Owners

The first graph illustrates the cumulative time each DO requires to complete their portion of the protocol. This includes encoding, symmetric encryption (as part of the transciphering pipeline), and encrypted data transfer. As expected, the time increases linearly with the number of DOs, indicating that each participant independently and sequentially processes their input. The magnitude of the slope confirms that the symmetric operations are computationally inexpensive, but the aggregation of these small contributions results in a growing system-wide cost. The error bars in this graph are minimal, suggesting that variance across experiments is negligible and the process is stable across executions.

Graph (b): Pre-HE-Processing Time (Server)
The second barplot focuses on the time required by the Server to homomorphically decrypt the transciphered data and pack it such that it is ready to be homomorphically evaluated. This stage involves wrapping the symmetrically encrypted data in an FHE layer. Interestingly, the graph shows a highly consistent growth pattern, both in absolute values and in slope, that matches the homomorphic computation stage almost identically. This similarity is critical, as it suggests that the complexity of FHE decryption scales in a linear fashion with the number of DOs. The near-equivalence with the computation step (see Graph 4.4c) supports the claim that the Homomorphic decryption and evaluation paths in our system have symmetrical performance characteristics. However, at the Server/Aggregator side, these numbers have a huge impact: It takes a little less than 1 hour per 100 participating DOs (according to Graph 4.4f this is approximatelly 0.52 minutes per individual DO). This is by far the most time consuming part of this privacy preserving system and reinforces the idea that transciphering is very good to compress the transfer size, but is not recommended when computation time is a concern.

Graph (c): Homomorphic Evaluation Time (Server)
This graph represents the time it takes to perform the homomorphic evaluation on the encrypted inputs. In our implementation, this represents something that is clearly explained in Section 6.2 of Chapter 3, but is, in short, the sum over an affine transformation of data-vector of each DO. As anticipated, this value increases with the number of DOs, but what is most striking is the near-parallel trajectory with Graph 4.4b. Both operations, Pre-HE-processing, and evaluation, involve applying arithmetic to ciphertexts of similar size and structure, and this similarity is directly reflected in their time cost. This equivalence implies that server-side cost in a transciphered system is composed of two scaling layers, effectively increasing the per-DO workload with a set amount. The small variance (again indicated via the error bars) confirms this finding across multiple experiments.

Graph (d): Time Consumption at the Data Customer (DC)
The fourth graph presents the time required by the Data Customer to decrypt the final computation result. Importantly, the values here are almost flat, showing no significant increase in the number of Data Owners. This confirms that the final ciphertext transmitted to the DC is always of fixed size, regardless of how many DOs participated in the computation. The decryption workload remains constant because it acts on a single aggregate result. This graph's descending trend in the sixth normalized plot further supports the claim that the DCs cost per DO drops as the number of participants grows, an

efficiency benefit of centralized decryption in homomorphic protocols. The bigger error bars are a result of the scale of this process: where others have been calculated in seconds, minutes or even hours, this one is in terms of milliseconds and therefore extremely fast but volatile to fluctuations. This is as expected as the DC only needs to decrypt 1 aggregated result.

Graph (e): Memory Usage at the Server
The fifth barplot examines the server's peak memory footprint during execution. Unlike the constant behavior observed at the DC, server memory usage increases steadily with the number of DOs. This is expected, as each homomorphic ciphertext requires memory space, and more participants result in more ciphertexts being simultaneously loaded, processed, and temporarily stored. The memory slope suggests that system scalability is bounded not just by time, but also by available server memory, an important factor when moving from prototype to production. The error bars are almost nonexistent, as there is no garbage collection and reinforce the idea that the allocation is always the same and therefore a stable method to check the requirements at the server side.

Graph (f): Normalized Cost (Relative Performance)
The sixth and final graph provides the most critical insight: it displays the per-DO cost by dividing the values in each of the previous graphs by the number of participants. This normalization reveals which components scale linearly (or otherwise) and which remain constant. Most notably:

- The bars corresponding to the Data Customer (Figure 4.4d) clearly *decreases* with increasing DOs, highlighting that the cost is amortized across participants. In fact, when looking at this graph itself, there is a clear constant time-trend regardless of the amounts of DOs, which makes sense as it always decodes a constant amount of data, exactly aggregated as desired.

- In contrast, the Server's decryption, computation, and memory costs (Figures 4.4b, 4.4c, and 4.4e) remain mostly flat, demonstrating a direct linear scaling with DO count.

On Timing Variance: Interpreting Error Bars
Each of the barplots includes a small black mark extending from the top edge of each bar. These markers represent the standard error of the measurements obtained over multiple runs. In most cases, these error bars are so small they are barely visible, a desirable outcome that confirms consistent execution times. Larger deviations, such as in the DC graph (Figure 4.4d) have been attributed in their corresponding paragraph.
Taken together, these six graphs provide a robust and granular understanding of how each actor in the system behaves under load, and where future optimizations can yield the greatest benefit. They also confirm that the transciphered FHE pipeline exhibits predictable, linear performance characteristics, with certain roles (e.g., DC) benefitting from cost sharing as the system scales.

Comparison between different setups

The results indicate that `pasta2_3` (Figure 4.4 on page 43) and `pasta_3` (Figure 4.3 on page 43) exhibit highly similar computational behavior, with both achieving consistent and predictable performance. However, `pasta2_3` was found to be marginally faster in runtime. This improvement, however, comes at a cost: it incurs a higher memory footprint and significantly larger ciphertext sizes during transfer. In scenarios where available memory and bandwidth are constrained, `pasta_3` may be the more viable choice, whereas `pasta2_3` is more suited for environments optimized for throughput.
In contrast, the version using no FHE-friendly block cipher (Figure 4.2 on page 42), i.e., pure FHE encryption, demonstrated dramatically faster encryption and evaluation times. Nevertheless, this apparent benefit is heavily outweighed by the substantial increase in memory usage and data transfer requirements. The system consumed upwards of 17 GB of RAM in later experiments, compared to approximately 4 GB for the transciphered implementations. For large-scale scenarios involving 200, 300, 400, and 500 Data Owners, the pure FHE pipeline exceeded the memory capacity of the test system, resulting in failed or incomplete runs. This highlights a critical scalability limitation when using direct FHE on large datasets. However, this result is also suspect: transciphering only changes the ciphertext in-transit, and after the pre-HE-processing has taken place, the transciphered and pure ciphertexts should be exactly, apart from noise, the same. And in these results, that is not the case as can be seen in the Graphs of the required memory. In the Appendix, there are results of differing mod degrees and one can easily see that the "`HE_ONLY`" Memory requirements are almost an order of magnitude bigger than the other FHE-Friendly Block Ciphers.
These trend in the results are compatible with the trend in the results found in [54], where they looked at the data aggregation time (here the Homomorphically Decrypting and computing step) for Prida version one and two, using BFV and CKKS. It should be noted that these computations on the server side are very different, it is a good indication that these display similar trends. While the FHE-friendly block cipher and function to evaluate is quite different, the linear trend is present there as well.
Additional graphs and experiments, such as Pasta-4 and Hera-5, yield interesting results but were shifted to the Appendix which can be found as Figure 2 until Figure 10.

3.2 Transfer Delay and Communication Overhead in PPS

In the preceding analysis of the system's performance, we assumed that communication between the Data Owners (DOs) and the Server was instantaneous. This assumption allowed us to isolate and evaluate the computational costs associated with transciphering and homomorphic evaluation. However, in practice, especially in distributed environments, network communication introduces significant latency, particularly when dealing with large encrypted payloads.
To model and analyze this aspect more realistically, we extend our evaluation to include the **per-DO communication delay**, based on the data transfer time from each DO to the Server. Let s denote the ciphertext size in bytes and ν the available network bandwidth (in kilobytes per second). Then, the transfer time T_{transfer} for a single DO can be approximated as: $T_{\text{transfer}} = c + \frac{s}{\nu}$, where c is the time it takes to encrypt, either

homomorphically, or through FHE-friendly Block cipher and some trivial initialization time.
This transfer time is incurred individually for each DO. In our current pipeline, the Server (or aggregator) must wait to receive all encrypted inputs before initiating homomorphic operations. As such, the total delay before computation can begin grows linearly with the number of participating data owners (assuming the connections happen consecutively instead of concurrently). If we approximate transfer time across all DOs as roughly similar, the expression simplifies to:

$$T_{\text{total-transfer-wait}} = \sum_{i=1}^{n} T_{\text{transfer}}^{(i)} \approx n \cdot T_{\text{transfer}}$$

This introduces a critical bottleneck, particularly when dealing with hundreds of DOs or under limited bandwidth conditions. Therefore, the size of the ciphertext becomes extremely important, and an argument can be made that the time required to transcipher is easily gained back by the savings in having to transfer less and therefore be quicker overall. This, we can observer in Figure 4.5 clearly as the PASTA solutions are consistently faster than their HE only counterparts. For further analysis of the results with the different mod degrees, please look at Figure 1 in the Appendix (Page 55).

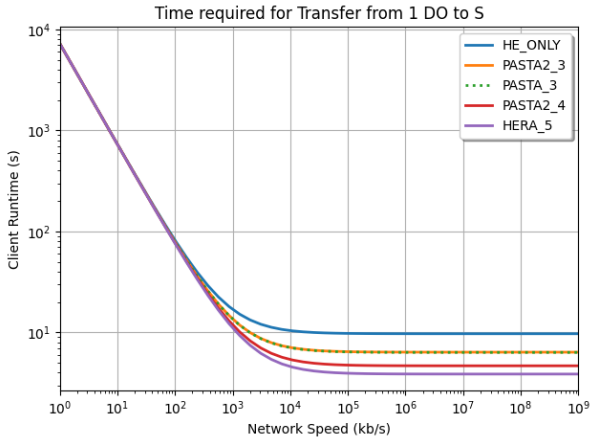


Figure 4.5: Timing per DO based on Transfer speed

An additional factor to consider is the cost. Depending on the setup or scale, transferring huge amounts of data could cost more than the additional computational requirements that are gained through transciphering. These findings further underline that privacy-preserving systems must be evaluated not only in terms of computation but also with respect to their communication topology and real-world deployment constraints.

Chapter 5

Conclusion

The thesis investigated the use of transciphering, combining FHE-friendly symmetric ciphers with homomorphic encryption, to build efficient privacy-preserving computation pipelines. We implemented and evaluated a prototype where each DO first encrypts its input using an FHE-friendly block cipher (e.g., Pasta), and a Server then homomorphically converts those ciphertexts and evaluates a function on them. DC holds the FHE secret key and recovers only the final result.

Contributions: The main contributions of this work are

1. The design and implementation of the transciphering pipeline itself, making use of available libraries.
2. An analysis of its privacy-efficiency trade-offs.
3. Implemented a general usable function and its possible use cases.

We introduced practical protocol variants to enhance privacy, Choice Vectors to let DOs opt in or out, and thresholding to require a minimum number of participants. These ideas help achieve anonymity and access-control goals, as illustrated in our privacy tables (3.1 and 3.2). We also benchmarked multiple FHE-friendly ciphers (two Pasta instances) using established libraries, comparing them to a baseline FHE-only system. Our evaluation showed that, with optimized ciphers like Pasta, the homomorphic decryption overhead can be somewhat mitigated. Pasta has been shown in prior work to outperform other ciphers by large factors [5]. Incorporating such a cipher meant that the homomorphic decryption and evaluation steps, while still costly, were significantly faster than naive choices. In essence, we contributed both a concrete system prototype and empirical data on its performance.

Key Results: Our experiments confirmed that transciphering dramatically reduces the communication overhead compared to using FHE alone. For example, symmetric encryptions have negligible expansion, so the size of data sent by each DO is much smaller than that of full FHE ciphertexts. This matches observations in prior work that "HHE is to drastically reduce bandwidth requirements when using homomorphic encryption, at the cost of more expensive computations in the encrypted domain" [5]. In our setting, encrypting with Pasta (a cipher optimized for FHE use) maintained correctness and

privacy. However, the Server's computation cost increased significantly. Homomorphic evaluation of the decryption circuit (followed by the actual function evaluation) dominated the runtime. For instance, in our graphs the server needed on the order of one hour of processing for each 100 data owners (about 0.5 min per owner), dwarfing the time for encoding/encryption at the clients. This confirms that, as expected, the transciphering pipeline trades much lower bandwidth for a higher computational load. Nevertheless, it does preserve correctness (the function outputs match those from a plain HE pipeline) and strong confidentiality (the Server never learns private inputs or outputs).

Challenges and Limitations: We discovered that the main bottleneck is computational overhead at the server. Fully homomorphic evaluation of any nontrivial circuit remains expensive, especially when it includes even a few multiplications. In our pipeline the addition of the symmetric decryption circuit doubled the server's work (decrypt + compute), leading to very long runtimes. In practice, this means transciphering may not be suitable for latency-sensitive tasks. Another challenge is the potential impact of dummy values on accuracy: inserting trivial or padded data can skew functions (e.g., averages), so careful handling (or use of noise-aware encoding schemes) is needed. The security analysis also assumes a semi-honest server; defending against active (malicious) adversaries would require extra tools (e.g., verifiable computing), which adds complexity. Finally, our experiments used large polynomial moduli (degree 32768) to get enough precision, which incurs heavy computation; smaller parameters or approximate schemes (like CKKS) might improve performance but introduce approximation error. These factors limit the immediate practicality of the approach, though they do not violate the fundamental correctness or security guarantees of the design.

Future Work: To further this research, one direction is optimizing the homomorphic decryption step. For example, using more efficient algorithms or hardware acceleration (GPUs or FPGAs) could reduce the server's workload. Investigating other FHE-friendly ciphers (beyond Pasta) or co-designing ciphers specifically for the target FHE library could yield speed-ups, as suggested by the HHE literature [5]. From an algorithmic perspective, one could explore bootstrapping or ciphertext packing to amortize work across many inputs. On the privacy side, implementing active security measures (malicious-server protections) or formalizing the security proofs for threshold/dummy techniques would strengthen the model. Finally, applying this pipeline to real-world workloads (such as private machine learning inference or statistics) would test its viability; for instance, integrating batched encoding to process vectors of data may improve throughput. Overall, while our results show that transciphering offers clear communication advantages, future work must address the computational trade-offs to make such systems efficient and practical in realistic settings.

Bibliography

[1] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the forty-first annual ACM symposium on Theory of Computing*. STOC '09 (May 2009), pp. 169–178. DOI: 10.1145/1536414.1536440.

[2] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption without Bootstrapping*. Cryptology ePrint Archive, Paper 2011/277. https://eprint.iacr.org/2011/277. 2011. URL: https://eprint.iacr.org/2011/277.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. Cambridge, Massachusetts: Association for Computing Machinery, Jan. 2012, pp. 309–325. ISBN: 9781450311151. DOI: 10.1145/2090236.2090262. URL: https://doi.org/10.1145/2090236.2090262.

[4] Zvika Brakerski and Vinod Vaikuntanathan. *Efficient Fully Homomorphic Encryption from (Standard) LWE*. Cryptology ePrint Archive, Paper 2011/344. Oct. 2011. DOI: 10.1109/focs.2011.12. URL: https://eprint.iacr.org/2011/344.

[5] Christoph Dobraunig et al. "Pasta: A Case for Hybrid Homomorphic Encryption". In: *IACR Cryptol. ePrint Arch.* 2021 (2021), p. 731.

[6] Christoph Dobraunig et al. *Rasta: A cipher with low ANDdepth and few ANDs per bit*. Cryptology ePrint Archive, Paper 2018/181. 2018. DOI: 10.1007/978-3-319-96884-1_22. URL: https://eprint.iacr.org/2018/181.

[7] Tomer Ashur, Mohammad Mahzoun, and Dilara Toprakhisar. *Chaghri - an FHE-friendly Block Cipher*. Cryptology ePrint Archive, Paper 2022/592. https://eprint.iacr.org/2022/592. 2022. URL: https://eprint.iacr.org/2022/592.

[8] Martin Albrecht et al. *Ciphers for MPC and FHE*. Cryptology ePrint Archive, Paper 2016/687. 2016. URL: https://eprint.iacr.org/2016/687.

[9] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Homomorphic Evaluation of the AES Circuit*. Cryptology ePrint Archive, Paper 2012/099. 2012. URL: https://eprint.iacr.org/2012/099.

[10] *Microsoft SEAL (release 4.1)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Jan. 2023.

[11] Shai Halevi and Victor Shoup. *HElib: An Implementation of Homomorphic Encryption*. https://github.com/homenc/HElib. 2013.

[12] Ahmad Al Badawi et al. *OpenFHE: Open-Source Fully Homomorphic Encryption Library*. Cryptology ePrint Archive, Paper 2022/915. https://eprint.iacr.org/2022/915. 2022. URL: https://eprint.iacr.org/2022/915.

[13] Alfred J Menezes, Paul C van Oorschot, and Scott A Vanstone. *Handbook of Applied Cryptography*. CRC Press, Dec. 1996. ISBN: 9780429466335. DOI: 10.1201/9780429466335. URL: http://dx.doi.org/10.1201/9780429466335.

[14] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, 1996.

[15] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, Jan. 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4.

[16] National Institute of Standards and Technology. *FIPS PUB 197: Advanced Encryption Standard (AES)*. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf,https://doi.org/10.6028/NIST.FIPS.197-upd1. 2001.

[17] Douglas R Stinson. *Cryptography: Theory and Practice*. Vol. 91. 520. Chapman & Hall/CRC, Mar. 2005, pp. 189–189. DOI: 10.1017/s002555720018146x. URL: http://dx.doi.org/10.1017/s002555720018146x.

[18] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. 2008. DOI: 10.2139/ssrn.3977007. URL: http://dx.doi.org/10.2139/ssrn.3977007.

[19] Daniel J Bernstein. "ChaCha, a variant of Salsa20". In: *Workshop Record of SASC* (2008). URL: https://cr.yp.to/chacha/chacha-20080128.pdf.

[20] Mendel F. et al. Dobraunig C. Eichlseder M. "Ascon v1.2: Lightweight Authenticated Encryption and Hashing". In: *J cryptol* 34.3 (June 2021). ISSN: 1432-1378. DOI: https://doi.org/10.1007/s00145-021-09398-9. URL: http://dx.doi.org/10.1007/s00145-021-09398-9.

[21] Claude E Shannon. "Communication theory of secrecy systems". In: *Bell system technical journal* 28.4 (Oct. 1949), pp. 656–715. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1949.tb00928.x. URL: http://dx.doi.org/10.1002/j.1538-7305.1949.tb00928.x.

[22] Junfeng Fan and Frederik Vercauteren. "Somewhat practical fully homomorphic encryption". In: *IACR Cryptol. ePrint Arch.* 2012 (2012), p. 144. URL: https://eprint.iacr.org/2012/144.

[23] Jung Hee Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: *Advances in Cryptology - ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 409–437. ISBN: 978-3-319-70694-8. DOI: 10.1007/978-3-319-70694-8_15. URL: http://dx.doi.org/10.1007/978-3-319-70694-8_15.

[24] Tomer Ashur and Siemen Dhooghe. *MARVELlous: a STARK-Friendly Family of Cryptographic Primitives*. Cryptology ePrint Archive, Paper 2018/1098. 2018. URL: https://eprint.iacr.org/2018/1098.

[25] TU Graz Christian Rechberger. *On FHEMPCZK-friendly symmetric crypto*. In Slide 8. June 2023.

[26] Dilara Toprakhisar. "Behaviour of Algebraic Ciphers in Fully Homomorphic Encryption". In: *Master's Thesis, Eindhoven University of Technology* (2021).

[27] Christoph Dobraunig et al. "Pasta: A Case for Hybrid Homomorphic Encryption". In: (2022). URL: %5Curl%7Bhttps://rwalch.at/talk/fhe_org22/pasta.pdf%7D.

[28] U.S. National Institute of Standards and Technology. *FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. 2015. DOI: 10.6028/nist.fips.202. URL: http://dx.doi.org/10.6028/nist.fips.202.

[29] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. "Can homomorphic encryption be practical?" In: *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*. CCSW '11. Chicago, Illinois, USA: Association for Computing Machinery, Oct. 2011, pp. 113–124. ISBN: 9781450310048. DOI: 10.1145/2046660.2046682. URL: https://doi.org/10.1145/2046660.2046682.

[30] Cynthia Dwork. "Differential privacy: A survey of results". In: *Theory and Applications of Models of Computation*. Springer, 2008, pp. 1–19. ISBN: 9783540792284. DOI: 10.1007/978-3-540-79228-4_1. URL: http://dx.doi.org/10.1007/978-3-540-79228-4_1.

[31] Arvind Narayanan and Vitaly Shmatikov. "Robust de-anonymization of large sparse datasets". In: *IEEE Symposium on Security and Privacy*. IEEE. IEEE, May 2008, pp. 111–125. DOI: 10.1109/sp.2008.33. URL: http://dx.doi.org/10.1109/sp.2008.33.

[32] Manuel Freire-Garabal y Núñez. *General Vision of the Regulation (EU) 2016/679 of the European Parliament and of the Council of April 27, 2016*. General Data Protection Regulation. June 2020. DOI: 10.21428/18d9181c.39ae71fc. URL: http://dx.doi.org/10.21428/18d9181c.39ae71fc.

[33] Dan Boneh et al. *Private Database Queries Using Somewhat Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2013/422. 2013. DOI: 10.1007/978-3-642-38980-1_7. URL: https://eprint.iacr.org/2013/422.

[34] David F Ferraiolo and D Richard Kuhn. "Proposed NIST standard for role-based access control". In: *Proceedings of the 15th National Computer Security Conference*. Vol. 4. 3. NIST. Association for Computing Machinery (ACM), Aug. 2001, pp. 1–12. DOI: 10.1145/501978.501980. URL: http://dx.doi.org/10.1145/501978.501980.

[35] O. Goldreich, S. Micali, and A. Wigderson. "How to play ANY mental game". In: *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. STOC '87. ACM. ACM Press, 1987, pp. 218–229. DOI: 10.1145/28395.28420. URL: http://dx.doi.org/10.1145/28395.28420.

[36] Ilaria Chillotti et al. "TFHE: Fast Fully Homomorphic Encryption Over the Torus". In: *Journal of Cryptology* 33.1 (Apr. 2020), pp. 34–91. ISSN: 1432-1378. DOI: 10.1007/s00145-019-09319-x. URL: http://dx.doi.org/10.1007/s00145-019-09319-x.

[37] Martin Albrecht et al. “Ciphers for MPC and FHE”. In: *Advances in Cryptology-EUROCRYPT 2015*. Springer, 2015, pp. 430–454. ISBN: 9783662468005. DOI: 10.1007/978-3-662-46800-5_17. URL: http://dx.doi.org/10.1007/978-3-662-46800-5_17.

[38] Vibhor Rastogi and Dan Suciu. “Relationship between k-anonymity and differential privacy”. In: *Proceedings of the 2007 ACM symposium on Principles of database systems*. ACM, 2007, pp. 43–52.

[39] Cynthia Dwork. “Calibrating noise to sensitivity in private data analysis”. In: *Theory of Cryptography Conference* 7.3 (May 2006), pp. 265–284. ISSN: 2575-8527. DOI: 10.29012/jpc.v7i3.405. URL: http://dx.doi.org/10.29012/jpc.v7i3.405.

[40] Qiang Yang et al. “Federated machine learning: Concept and applications”. In: *ACM Transactions on Intelligent Systems and Technology* 10.2 (Jan. 2019), pp. 1–19. ISSN: 2157-6912. DOI: 10.1145/3298981. URL: http://dx.doi.org/10.1145/3298981.

[41] Martin Albrecht et al. *MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity*. Cryptology ePrint Archive, Paper 2016/492. 2016. DOI: 10.1007/978-3-662-53887-6_7. URL: https://eprint.iacr.org/2016/492.

[42] Reza Shokri et al. “Privacy-preserving deep learning”. In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. CCS’15 (Oct. 2015), pp. 1310–1321. DOI: 10.1145/2810103.2813687. URL: http://dx.doi.org/10.1145/2810103.2813687.

[43] Sara Halawi, Yuriy Polyakov, et al. “Bootstrapping Fully Homomorphic Encryption in Sublinear Time”. In: *IACR Cryptol. ePrint Arch.* 2022.134 (Nov. 2022), pp. 501–537. ISSN: 2569-2925. DOI: 10.46586/tches.v2023.i1.501-537. URL: https://eprint.iacr.org/2022/134.

[44] Dennis Günther and et al. “CHIMERA: Combining Ring-LWE-Based Fully Homomorphic Encryption Schemes”. In: *IACR Cryptol. ePrint Arch.* 2020.1 (Aug. 2020), p. 1218. ISSN: 1862-2976. DOI: 10.1515/jmc-2019-0026. URL: https://eprint.iacr.org/2020/1218.

[45] Peter W. F. Wilson et al. “Prediction of Coronary Heart Disease Using Risk Factor Categories”. In: *Circulation* 97.18 (May 1998), pp. 1837–1847. ISSN: 1524-4539. DOI: 10.1161/01.CIR.97.18.1837. URL: https://www.ahajournals.org/doi/abs/10.1161/01.CIR.97.18.1837.

[46] Shai Halevi and Victor Shoup. “Bootstrapping for HElib”. In: *IACR Cryptol. ePrint Arch.* 2014.1 (Jan. 2014), p. 873. ISSN: 1432-1378. DOI: 10.1007/s00145-020-09368-7. URL: https://eprint.iacr.org/2014/873.

[47] M. Sadegh Riazi et al. “Chameleon: A hybrid secure computation framework for machine learning applications”. In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ASIA CCS ’18 (May 2018), pp. 707–721. DOI: 10.1145/3196494.3196522. URL: http://dx.doi.org/10.1145/3196494.3196522.

[48] Payman Mohassel and Yupeng Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning". In: *IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2017, pp. 19–38. DOI: 10.1109/sp.2017.12. URL: http://dx.doi.org/10.1109/sp.2017.12.

[49] Ashequr Rahman, Md Risalat, and Md Risalat Hossain Ontor. "PRIVACY-PRESERVING MACHINE LEARNING: TECHNIQUES, CHALLENGES, AND FUTURE DIRECTIONS IN SAFEGUARDING PERSONAL DATA MANAGEMENT". In: 4 (Dec. 2024), p. 2024. DOI: 10.55640/ijbms-04-12-03.

[50] Jung Hee Cheon et al. *Bootstrapping for Approximate Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2018/153. https://eprint.iacr.org/2018/153. 2018. DOI: 10.1007/978-3-319-78381-9_14. URL: https://eprint.iacr.org/2018/153.

[51] Ilaria Chillotti et al. *TFHE: Fast Fully Homomorphic Encryption Library*. https://tfhe.github.io/tf 2016.

[52] Wikipedia. *Homomorphic encryption - Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Homomorphic%20encryption&oldid=1269605530. [Online; accessed 27-January-2025]. 2025.

[53] Ilaria Chillotti et al. *Faster Fully Homomorphic Encryption: Bootstrapping in less than 0.1 Seconds*. Cryptology ePrint Archive, Paper 2016/870. https://eprint.iacr.org/2016/870. 2016. DOI: 10.1007/978-3-662-53887-6_1. URL: https://eprint.iacr.org/2016/870.

[54] Beyza Bozdemir, Betül Aşkın Özdemir, and Melek Önen. *PRIDA: PRIvacy-preserving Data Aggregation with multiple data customers*. Cryptology ePrint Archive, Paper 2024/074. 2024. DOI: 10.1007/978-3-031-65175-5_4. URL: https://eprint.iacr.org/2024/074.

Appendix

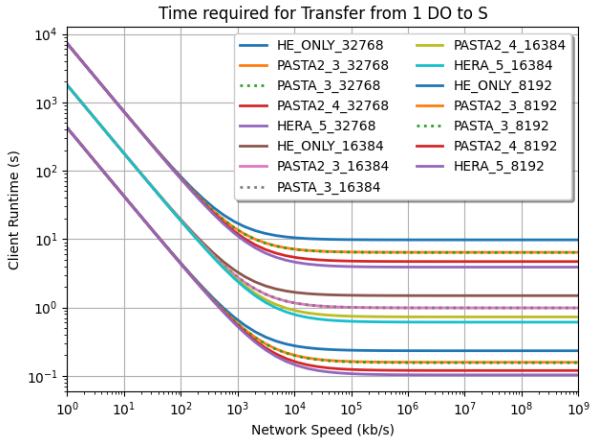
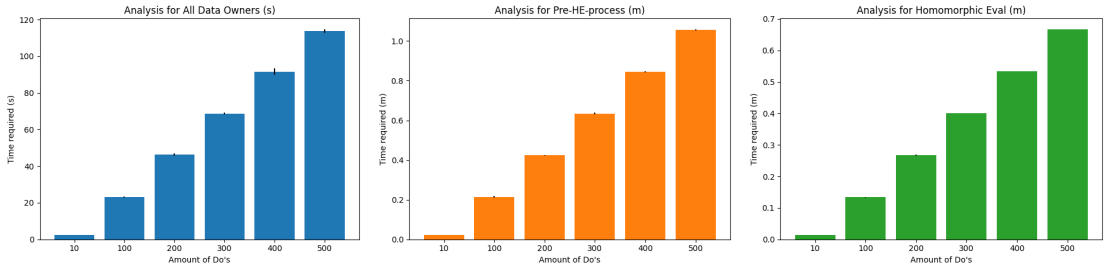
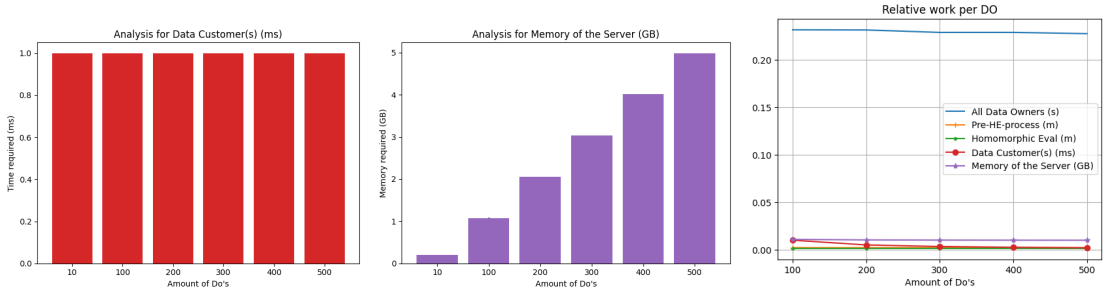


Figure 1: Timing per DO based on Transfer speed

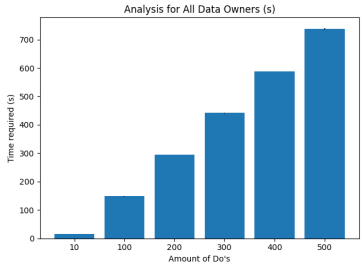


(a) All Data Owners (b) Pre-HE-Processing (c) Homomorphic Evaluation

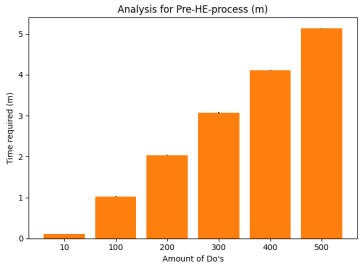


(d) Data Customer(s) (e) Memory of the Server (f) Scale with DO

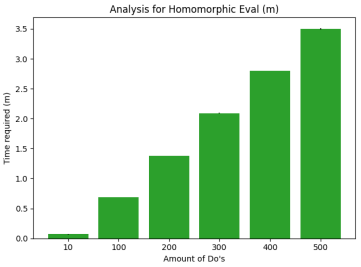
Figure 2: Results for 1 DC, Cipher: `None`, mod degree: 8192



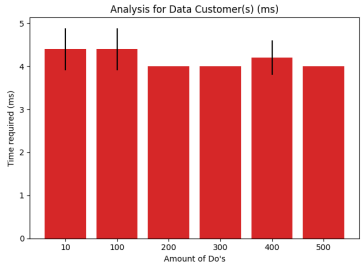
(a) All Data Owners



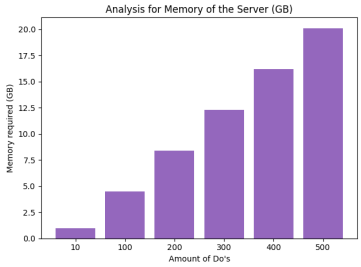
(b) Pre-HE-Processing



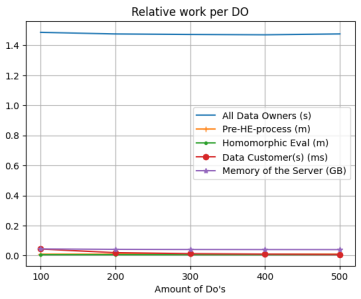
(c) Homomorphic Evaluation



(d) Data Customer(s)

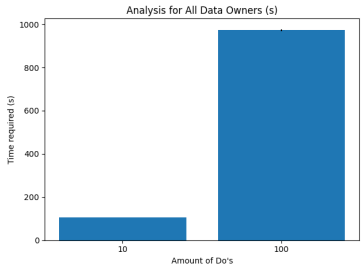


(e) Memory of the Server

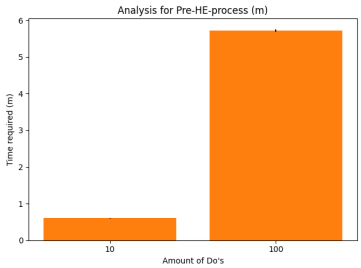


(f) Scale with DO

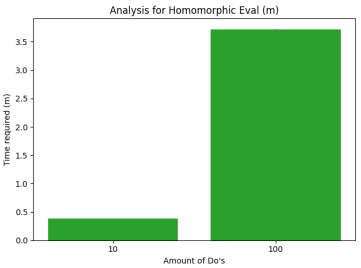
Figure 3: Results for 1 DC, Cipher: `None`, mod degree: 16384



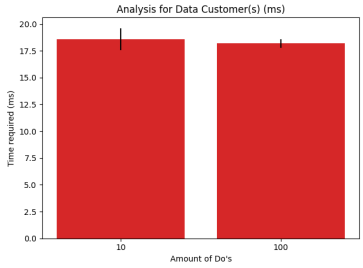
(a) All Data Owners



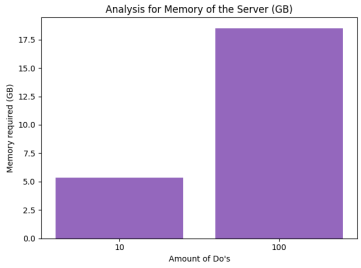
(b) Pre-HE-Processing



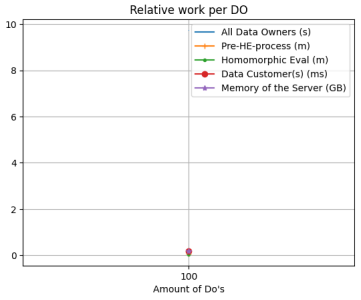
(c) Homomorphic Evaluation



(d) Data Customer(s)

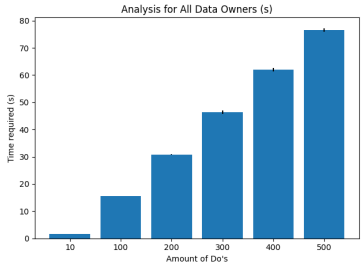


(e) Memory of the Server

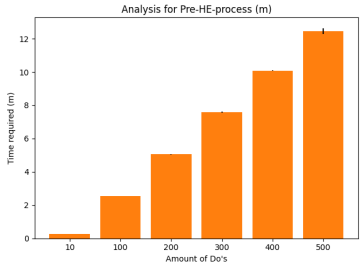


(f) Scale with DO

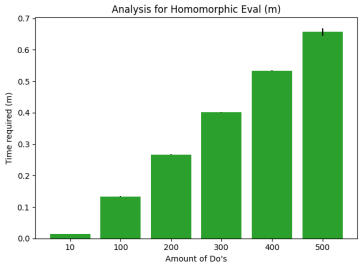
Figure 4: Results for 1 DC, Cipher: `None`, mod degree: 32768



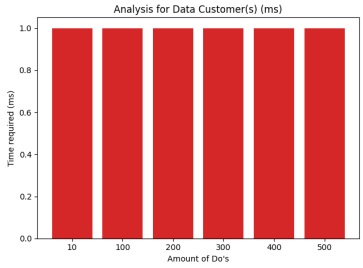
(a) All Data Owners



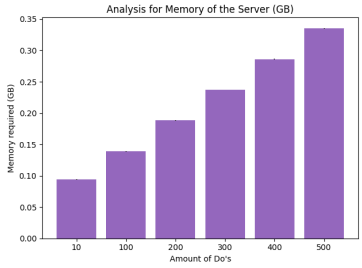
(b) Pre-HE-Processing



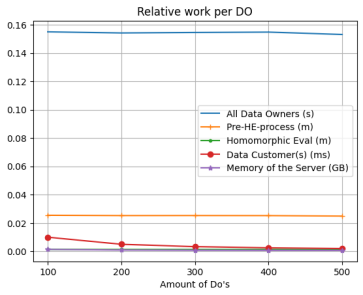
(c) Homomorphic Evaluation



(d) Data Customer(s)

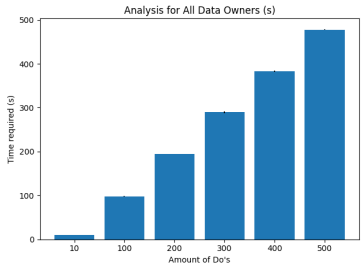


(e) Memory of the Server

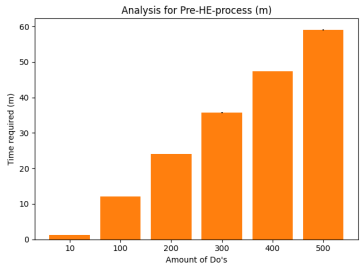


(f) Scale with DO

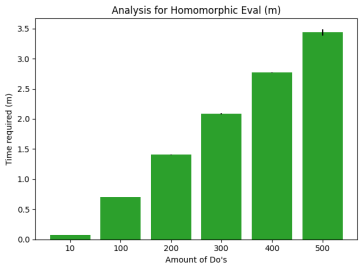
Figure 5: Results for 1 DC, Cipher: `Pasta-3`, mod degree: 8192



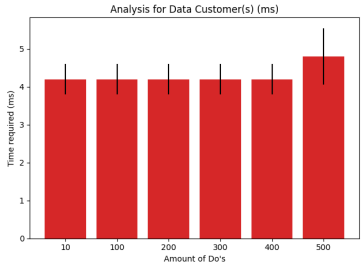
(a) All Data Owners



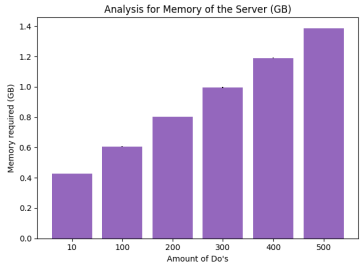
(b) Pre-HE-Processing



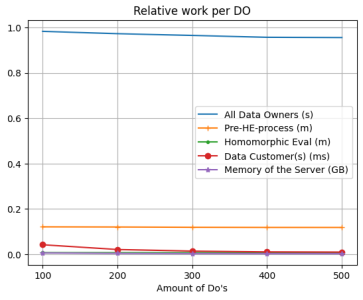
(c) Homomorphic Evaluation



(d) Data Customer(s)

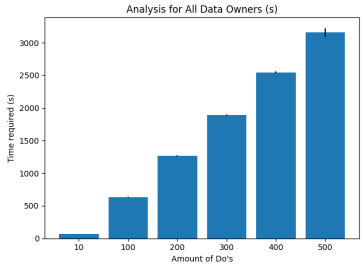


(e) Memory of the Server

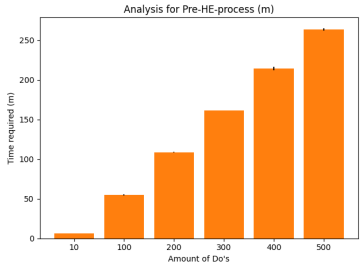


(f) Scale with DO

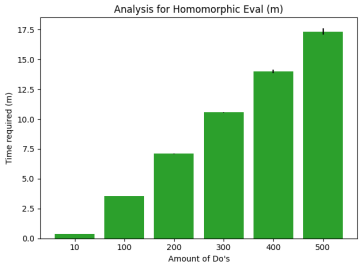
Figure 6: Results for 1 DC, Cipher: `Pasta-3`, mod degree: 16384



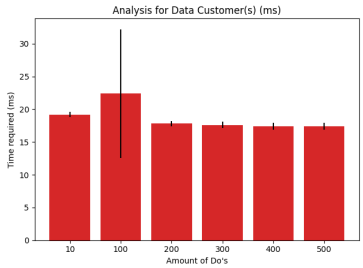
(a) All Data Owners



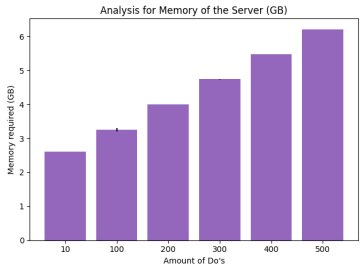
(b) Pre-HE-Processing



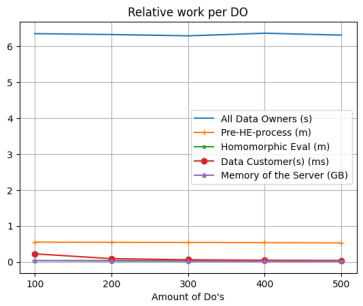
(c) Homomorphic Evaluation



(d) Data Customer(s)

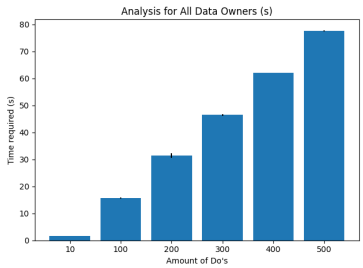


(e) Memory of the Server

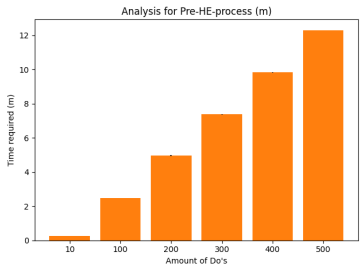


(f) Scale with DO

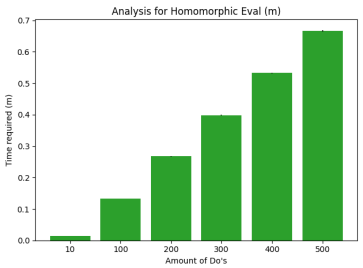
Figure 7: Results for 1 DC, Cipher: `Pasta-3`, mod degree: 32768



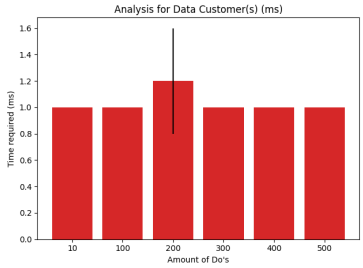
(a) All Data Owners



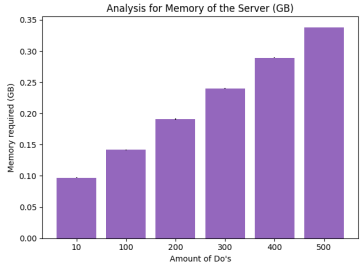
(b) Pre-HE-Processing



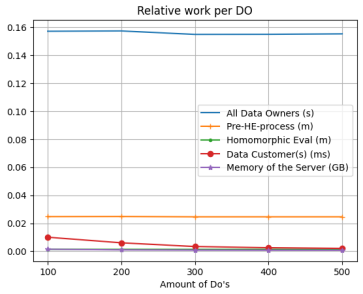
(c) Homomorphic Evaluation



(d) Data Customer(s)

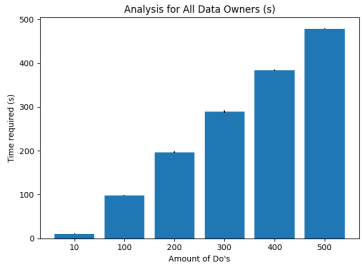


(e) Memory of the Server

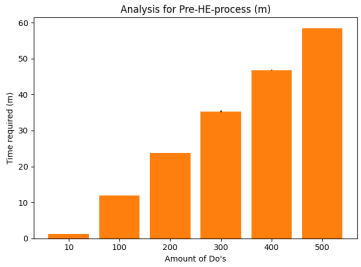


(f) Scale with DO

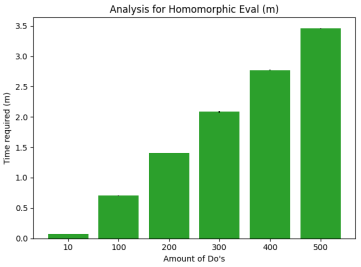
Figure 8: Results for 1 DC, Cipher: `Pasta2-3`, mod degree: 8192



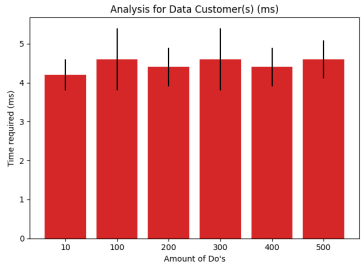
(a) All Data Owners



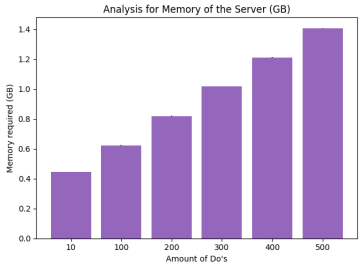
(b) Pre-HE-Processing



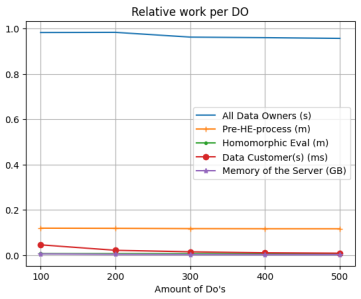
(c) Homomorphic Evaluation



(d) Data Customer(s)

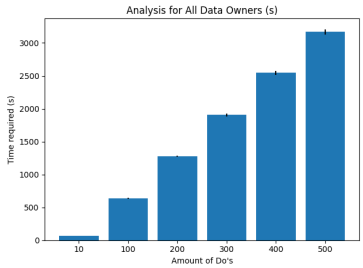


(e) Memory of the Server

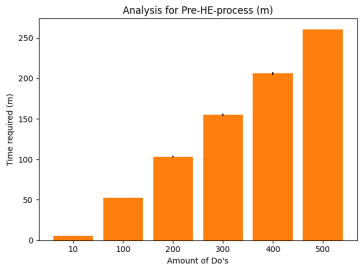


(f) Scale with DO

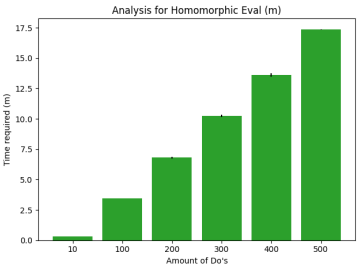
Figure 9: Results for 1 DC, Cipher: `Pasta2-3`, mod degree: 16384



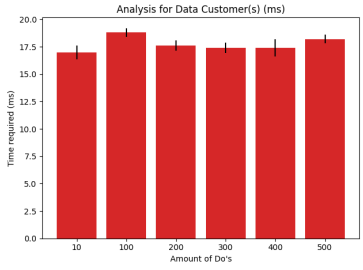
(a) All Data Owners



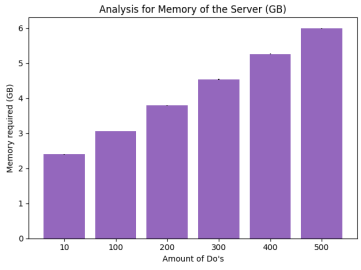
(b) Pre-HE-Processing



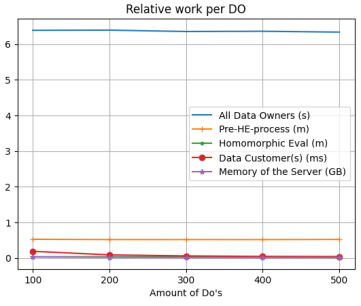
(c) Homomorphic Evaluation



(d) Data Customer(s)

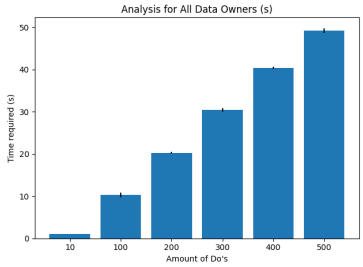


(e) Memory of the Server

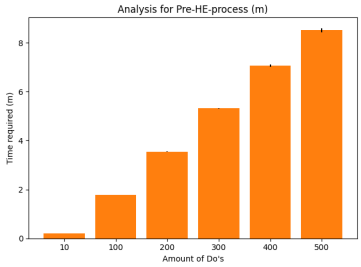


(f) Scale with DO

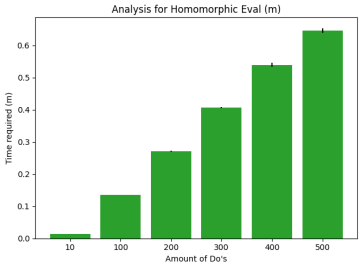
Figure 10: Results for 1 DC, Cipher: `Pasta2-3`, mod degree: 32768



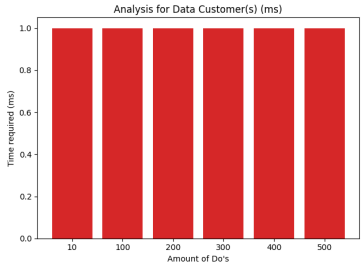
(a) All Data Owners



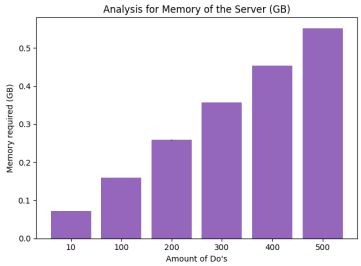
(b) Pre-HE-Processing



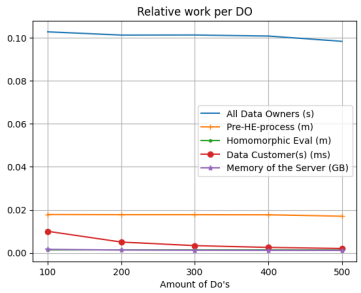
(c) Homomorphic Evaluation



(d) Data Customer(s)

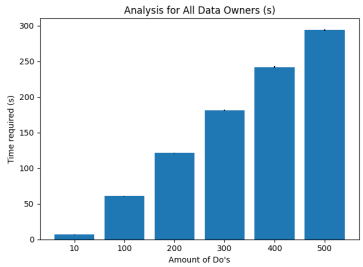


(e) Memory of the Server

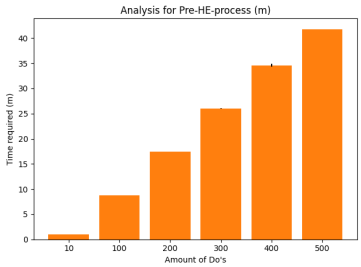


(f) Scale with DO

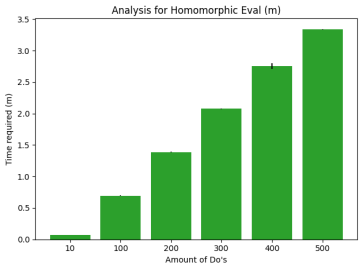
Figure 11: Results for 1 DC, Cipher: `Hera-5`, mod degree: 8192



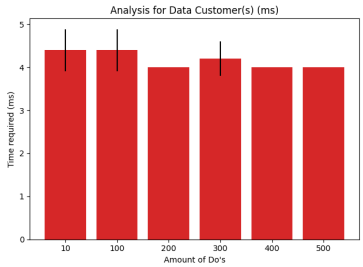
(a) All Data Owners



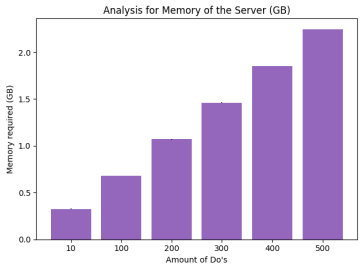
(b) Pre-HE-Processing



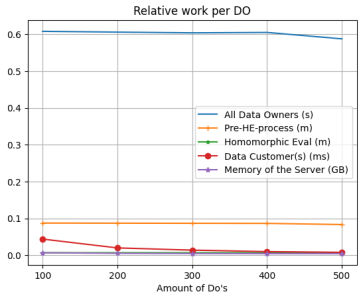
(c) Homomorphic Evaluation



(d) Data Customer(s)

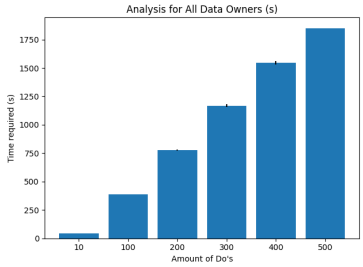


(e) Memory of the Server

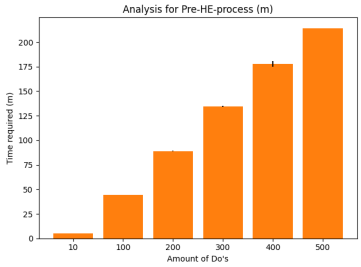


(f) Scale with DO

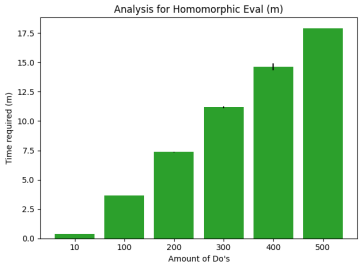
Figure 12: Results for 1 DC, Cipher: `Hera-5`, mod degree: 16384



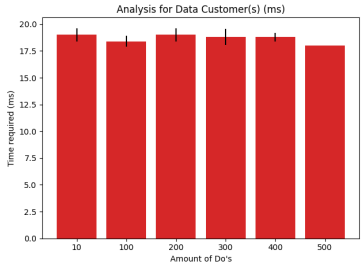
(a) All Data Owners



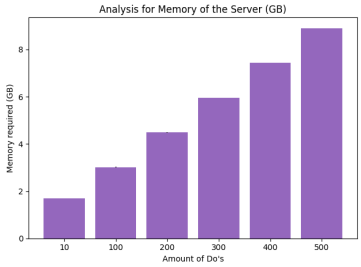
(b) Pre-HE-Processing



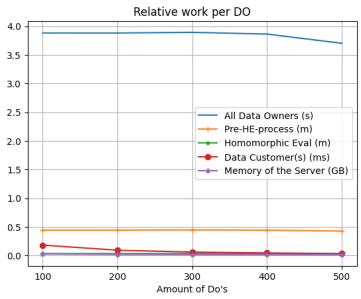
(c) Homomorphic Evaluation



(d) Data Customer(s)

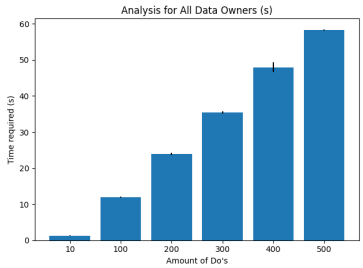


(e) Memory of the Server

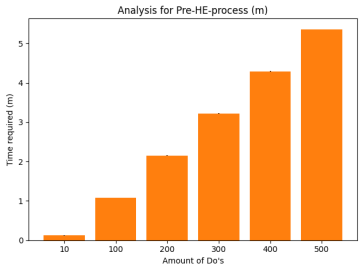


(f) Scale with DO

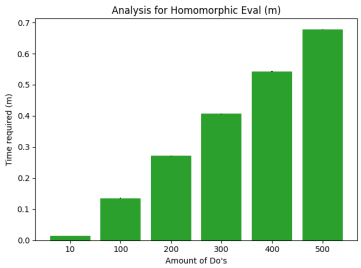
Figure 13: Results for 1 DC, Cipher: `Hera-5`, mod degree: 32768



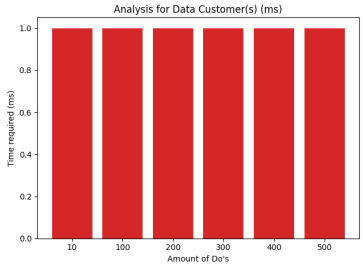
(a) All Data Owners



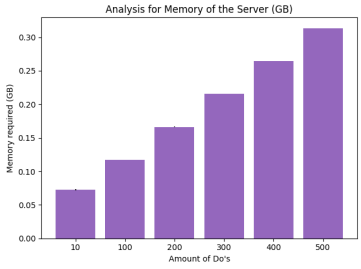
(b) Pre-HE-Processing



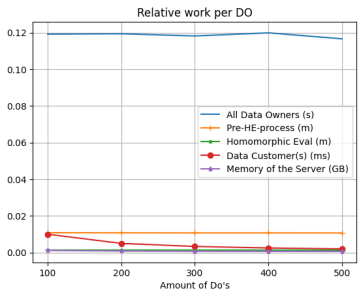
(c) Homomorphic Evaluation



(d) Data Customer(s)

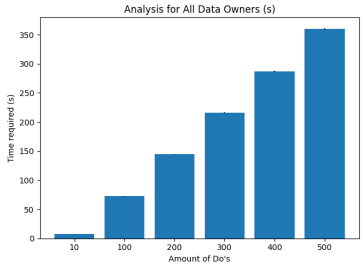


(e) Memory of the Server

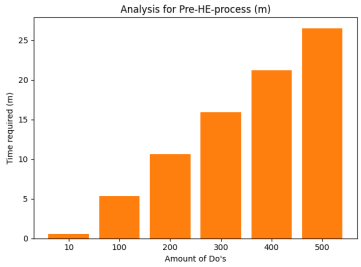


(f) Scale with DO

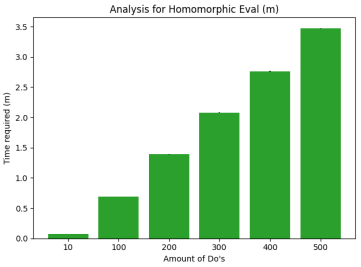
Figure 14: Results for 1 DC, Cipher: `Pasta2-4`, mod degree: 8192



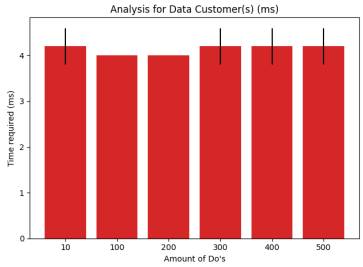
(a) All Data Owners



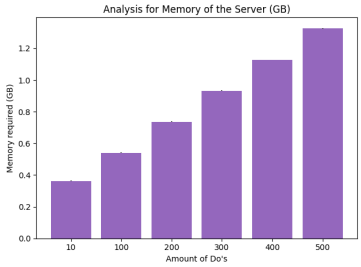
(b) Pre-HE-Processing



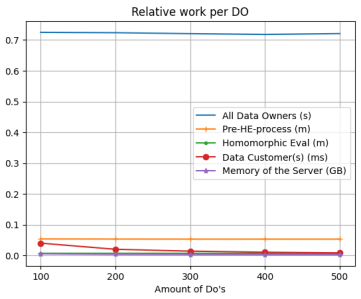
(c) Homomorphic Evaluation



(d) Data Customer(s)

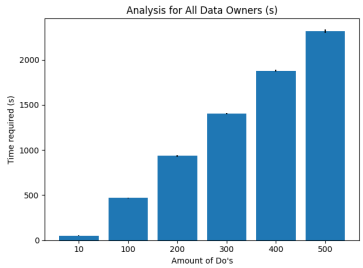


(e) Memory of the Server

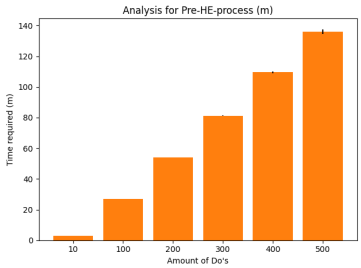


(f) Scale with DO

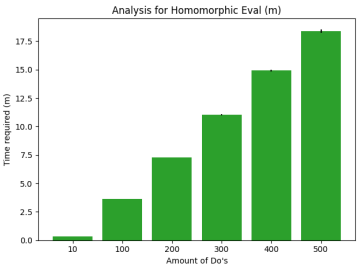
Figure 15: Results for 1 DC, Cipher: `Pasta2-4`, mod degree: 16384



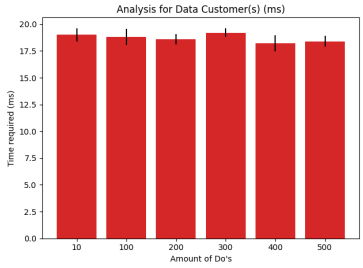
(a) All Data Owners



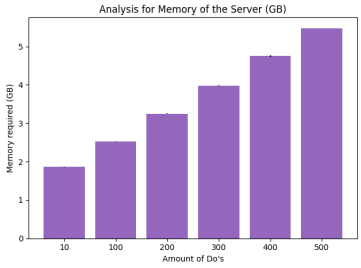
(b) Pre-HE-Processing



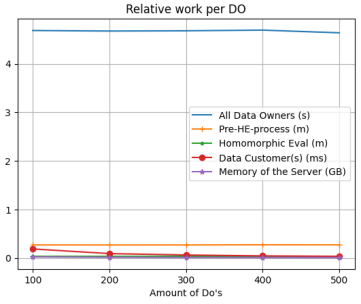
(c) Homomorphic Evaluation



(d) Data Customer(s)



(e) Memory of the Server



(f) Scale with DO

Figure 16: Results for 1 DC, Cipher: `Pasta2-4`, mod degree: 32768

AFDELING E
Kasteelpark Arenberg 10 - bus 2
3001 LEUVEN, BEL
tel. + 32 16 32 1
fax + 32 16 32 1
www.kuleuve