

Article

Homomorphic Encryption as an Enabler for Secure Multi-Source Data Aggregation and Confidential Analytics

Cristina Regueiro *, Julen Bernabé-Rodríguez , Iñaki Seco-Aguirre  and Idoia Gamiz *

TECNALIA, Basque Research and Technology Alliance (BRTA), Bizkaia Science and Technology Park, 700, 48160 Derio, Bizkaia, Spain; julen.bernabe@tecnalia.com (J.B.-R.); inaki.seco@tecnalia.com (I.S.-A.)

* Correspondence: cristina.regueiro@tecnalia.com (C.R.); idoia.gamiz@tecnalia.com (I.G.)

Abstract

Homomorphic Encryption plays a key role in secure multi-source data aggregation because it enables computations to be performed directly over encrypted data, allowing distributed parties to contribute sensitive information while preserving confidentiality. However, its use in this context introduces three main challenges: existing approaches often focus on specific operations rather than supporting diverse analytics across multiple encrypted data sources; key generation and management frequently rely on trusted third parties or require private keys to be shared; and TEE-based solutions may avoid trusted third parties but often require computations to be partially executed inside the trusted environment, thereby limiting deployment flexibility. To address these limitations, this work makes three main contributions: (i) the proposal of a complete framework for secure multi-source data aggregation that leverages homomorphic encryption and enables any data consumer to securely run multi-source data aggregations over data previously registered by untrusted data providers; (ii) the integration of secure enclaves for the secure generation, distribution, and management of homomorphic keys, addressing challenges related to coordinated key synchronization in multi-party aggregation environments and removing the need for a trusted third party; and (iii) the introduction of a hybrid key management protocol that combines secure enclave-based key generation with efficient key distribution and secure aggregation outside the enclave, in the data consumer, minimizing trust assumptions and computational overhead. The implementation and evaluation on small-scale aggregated datasets show that the proposed approach effectively addresses the identified challenges by providing, to the best of the authors' knowledge, the first practical and privacy-preserving solution that supports different algorithms without relying on any trusted third party, while improving over existing solutions through the integration of secure enclaves and a hybrid key management protocol.



Academic Editors: Danda B. Rawat and Yangguang Tian

Received: 9 February 2026

Revised: 8 July 2026

Accepted: 16 July 2026

Published: 21 July 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: homomorphic encryption; secure enclave; data aggregation; confidential analytics; multi-source data; homomorphic key management

1. Introduction

In an era where data is often referred to as the “new oil”, data-driven systems have become central to decision-making across industries [1]. While larger and more diverse datasets improve model accuracy, generalization, and fairness [2], the integration of confidential data from multiple sources remains a major barrier due to legal, ethical, and competitive constraints [3]. Regulations such as General Data Protection Regulation (GDPR) [4] enforce strict data minimization and purpose limitation principles [5], making traditional

data sharing and centralized processing increasingly infeasible. This creates a fundamental tension between the benefits of multi-source data aggregation and the need to preserve privacy and confidentiality.

From a technical perspective, data aggregation presents several inherent challenges, including the alignment and synchronization of heterogeneous data formats. Recent advancements in the field of data aggregation have explored different techniques [6,7] that have significantly improved analytical capabilities. However, their implementation in privacy-sensitive environments remains problematic. Most state-of-the-art data aggregation frameworks assume a trusted central aggregator, which represents a single point of failure and a high-risk target for data breaches.

Existing privacy-preserving solutions partially address these limitations but introduce new challenges. Federated learning and differential privacy [8] enable privacy-preserving analytics but often require data to be processed in a distributed manner, which may still expose model updates or intermediate results. Trusted execution environments (TEEs) offer strong isolation but rely on hardware trust assumptions and do not inherently support computation on encrypted data [9]. Meanwhile, homomorphic encryption (HE) enables computations on encrypted data without decryption, offering a powerful mechanism for preserving confidentiality throughout the entire data lifecycle [10,11]. However, even if multi-key HE solutions exist [12–14], their practical deployment remains challenging, particularly due to the lack of secure and scalable key management and the high computational overhead of fully homomorphic encryption (FHE) [15,16].

The core challenges that arise when using Homomorphic Encryption for secure multi-source data aggregation are:

- Supporting diverse HE-based analytics over multiple distributed data sources remains challenging. Existing solutions typically focus on specific operations or algorithms, rather than providing a general platform where different aggregations can be executed over multiple previously registered encrypted datasets.
- Using HE in multi-source aggregation introduces complex key management and trust issues. Several existing approaches require trusted third parties, such as a trusted cloud [17–20], while others rely on cryptographic protocols that may be computationally prohibitive for practical deployments [21].
- Combining HE with secure enclaves for key generation and management remains insufficiently explored. Although TEEs can reduce trust assumptions, existing TEE-based approaches often require part of the computation to be executed inside the enclave, which limits deployment flexibility and scalability.

Each challenge is directly linked to a specific contribution of this work. The contributions are summarized below:

1. The design of a comprehensive framework for secure multi-source data aggregation and confidential analytics based on FHE. The framework enables data consumers to execute different aggregation algorithms over encrypted datasets previously registered by multiple untrusted data providers, ranging from basic statistical operations (mean, standard deviation, etc.) to more complex machine learning tasks (logistic regression training), without requiring access to raw data.
2. The integration of secure enclaves into the framework for homomorphic key generation and management. This addresses the key management and trust challenges of HE-based multi-source aggregation by enabling key generation and distribution without relying on a trusted third party.
3. The introduction of a hybrid key management protocol that combines secure enclave-based key generation with efficient key distribution and secure aggregation outside

the enclave, in the data consumer, minimizing computational overhead. This avoids executing the main homomorphic computations inside the trusted environment, reducing computational overhead, limiting security exposure, and improving deployment flexibility compared with existing TEE-based solutions.

The framework has been implemented and evaluated on small-scale aggregated datasets, showing the feasibility of the approach for the execution of several algorithms. This implementation differs from prior work in the literature in several respects, as previous studies evaluated performance only for a single aggregation or domain-specific algorithm [22,23]. Concretely, the evaluation measures the performance of key generation, encryption, homomorphic computation, and decryption for a range of operations, for both simple and complex operations. Results show that while simple operations incur minimal overhead, complex computations require substantial resources, yet remain feasible with optimized FHE parameter selection.

To the best of the authors' knowledge, this is the first framework to present a practical and privacy-preserving solution for secure multi-source data aggregation across different algorithms. In contrast to the most related works [22,23], in which performance was evaluated only for a single aggregation or domain-specific algorithm, this solution ensures full encryption of the data during aggregation execution and allows distributed data consumers to run the aggregations, instead of assuming a server computing them. Furthermore, it enables confidential analytics without the need for any trusted third party by integrating secure enclaves, and improves upon existing HE and TEE combinations by implementing a hybrid key management protocol and computing the aggregation outside the TEE, in the data consumer.

The paper is organized as follows: Section 2 starts with a description of the state of the art in data aggregation, as well as HE and trusted execution environments, the two key technologies to be combined in the suggested solution. Section 3 theoretically describes the multi-source data aggregation and confidential analytics framework. This section also introduces a proposal of implementation. In addition, performance measurements are also discussed in Section 4. Finally, Section 5 draws the key conclusions and future work from the research.

2. Background

2.1. Secure Multi-Source Data Aggregation

Data aggregation is the process of integrating multiple data sources to produce more accurate, complete, and consistent information than that provided by any single source [24]. Despite its advantages, data aggregation faces significant technical challenges, particularly regarding data heterogeneity and synchronization. When dealing with multi-source integration, providers often employ different sampling rates, formats, or semantic definitions, requiring complex alignment and normalization processes. However, the most complex challenge remains to be integrating privacy to these systems. Aggregating multi-source data typically requires access to the raw data or high-dimensional features, which often violates privacy regulations or exposes corporate secrets. Existing state-of-the-art solutions have attempted to address this using techniques such as secure multi-party computation (SMPC) [25–29] and differential privacy [30–32], but these often introduce significant communication overhead or degrade the quality of the aggregated output. Therefore, there is a pressing need for a framework that enables high-fidelity data aggregation without requiring a trusted central authority and the exposure of raw sensitive data.

2.2. Homomorphic Encryption for Multi-Source Aggregation

Homomorphic Encryption (HE) enables arithmetic operations to be applied directly on ciphertexts while preserving the correctness of the final decrypted result [33]. This property makes HE particularly powerful for collaborative analytics scenarios in which

privacy regulations, organizational boundaries, or commercial constraints prevent direct data sharing.

Among the existing schemes Fully Homomorphic Encryption (FHE) is specially relevant because it supports unlimited number of both addition and multiplication operations, allowing arbitrary computations on encrypted data. This capability is essential for complex data aggregation tasks, such as training machine learning models across encrypted datasets. However, practical adoption remains challenging due to the significant computational overhead associated with ciphertext operations and noise management.

Several modern FHE libraries, including OpenFHE [34] and TFHE-rs [17], provide practical implementations for numerical and logical computations over encrypted data. In particular, CKKS [35] is commonly used for approximate arithmetic on real-valued data, while TFHE [36] is well suited for Boolean and comparison-based operations.

While HE protects the confidentiality of aggregated data, multi-source deployments introduce additional challenges related to key management. Multiple independent data providers must use compatible cryptographic contexts and encryption keys to enable joint homomorphic computation. At the same time, no single participant should obtain access to the master secret key, as this would compromise the confidentiality of all aggregated data.

Moreover, FHE systems require the management of auxiliary cryptographic material such as evaluation and rotation keys [37]. These additional keys increase deployment complexity and create new risks associated with key exposure [38]. Existing approaches have explored secure key generation and distribution mechanisms, including SMPC-based protocols and distributed key establishment techniques [19,39,40]. However, many of these solutions remain difficult to deploy in practice due to synchronization requirements, communication overhead, and operational complexity. Consequently, secure and practical key management remains one of the main barriers to the adoption of HE in multi-source aggregation environments.

2.3. Trusted Execution Environments for HE Key Management

In order to facilitate distributed key generation in multi-source FHE aggregations, Trusted Execution Environments (TEEs), also known as secure enclaves, have started to be explored [17,41]. TEEs refer to a technique that isolates an environment (processor, memory and storage) with high levels of security, limiting the exchange of information with other untrusted contexts [9,42]. For example, a secure enclave at the hardware and operating system level could create and store private keys and perform operations with them, such as ciphertexts decryption, without allowing the private keys to leave the secure enclave or to be loaded into memory of some untrusted process [43]. TEEs could serve as the “root of trust” for key management, ensuring that the generation and distribution of homomorphic keys for multiple parties occur in a protected environment.

While Hardware Security Modules (HSMs) provide secure key storage, they are primarily designed for static cryptographic operations. However, TEEs combine hardware-enforced isolation with programmable execution capabilities, making them better suited to support the generation, management, and use of homomorphic keys in multi-source aggregation environments.

Several works have explored the integration of TEEs and FHE. For example, TEE-assisted FHE systems have been proposed to protect private keys and accelerate cryptographic operations [17,18]. However, many of these approaches still rely on trusted cloud infrastructures or execute computations inside the enclave, limiting deployment flexibility. Other proposals demonstrate the feasibility of combining FHE and TEEs for secure computation [44], but do not address the key management challenges introduced by multi-source aggrega-

tion. Consequently, the use of TEEs for practical homomorphic key generation, distribution, and management across multiple independent data providers remains an open challenge.

3. Materials and Methods

Secure multi-source data aggregation demands a framework that enables collaborative computation on encrypted data while ensuring privacy, confidentiality, and trust among distributed data providers. To address this, this section presents a privacy-preserving multi-source aggregation and confidential analytics framework that supports secure, decentralized data integration without exposing raw data or private keys.

The proposed architecture and operational flow focus on addressing key challenges in distributed HE systems, particularly those related to key management, by leveraging TEEs for secure key generation and distribution. This design enables controlled trust assumptions, end-to-end confidentiality, and flexibility to support different computation types within multi-source settings in sensitive domains.

3.1. Reference Use Case

Consider an organization that aims to develop a predictive algorithm capable of identifying early signs of failures in a specific type of industrial machine. The machine collects its own operational data, including sensor readings, performance logs, and historical incident reports. However, when these datasets are analyzed independently, they provide only a partial perspective. The variability between locations, environmental conditions, and operational practices of other machines makes it difficult for a model trained on a single dataset to generalize effectively.

To overcome this limitation, the organization decides to search for a unified larger training dataset by combining operational information from other machines of the same model from different organizations. By integrating multi-source data, the algorithm benefits from a much richer and more diverse pool of examples. Rare failure modes that may occur only under specific conditions become visible in the combined dataset while patterns that were previously indistinguishable due to noise or insufficient sample size begin to emerge consistently across sources. This diversity helps the model learn more robust and representative relationships, leading to improved accuracy and reduced false alarms.

In addition, the larger aggregated dataset enhances the model's ability to generalize. Instead of memorizing the unique characteristics of a single site, the algorithm now learns common operational patterns shared across the network. This makes the predictions more reliable when deployed in new or evolving environments. The inclusion of multiple viewpoints also helps mitigate biases that could arise from incomplete or unbalanced local datasets. Ultimately, by training on integrated multi-source data, the organization produces a more resilient and high-performing predictive system. The model is better equipped to detect anomalies, anticipate failures, and support proactive decision-making.

Yet this integration introduces a critical challenge: each organization machines' operational data may contain sensitive or proprietary information that cannot be shared directly due to regulatory constraints, contractual agreements, and internal security policies.

Taking this reference use case as motivation, the following formal problem statement can be defined. Let $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ be a set of n data providers, each holding a private dataset $D_i \in \mathbb{R}^{m_i \times d}$, where m_i is the number of records and d is the number of features. Let $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ be a set of k data consumers, each requesting a homomorphic computation $f: \mathbb{R}^{m \times d} \rightarrow \mathbb{R}^p$, where m denotes the total number of input records, d the number of features of the input records, and p is the output dimension. The goal is to compute $f(D_1 \cup D_2 \cup \dots \cup D_n)$ without revealing any private dataset D_i , private

keys, or intermediate results to any party. Therefore, the function is computed over the concatenated dataset, and $m = \sum_{i=1}^n m_i$.

To address this, a concrete security model has been defined for this scenario. Let λ denote the security parameter, and let $\text{negl}(\lambda)$ denote a negligible function in λ . To rigorously analyze the security of the proposed framework, instead of assuming absolute security (i.e., zero probability of attack), any advantage in breaking confidentiality or integrity is bounded by a negligible function in the security parameter λ . The security goals are:

- **Confidentiality:** No raw data or private keys can be exposed to the cloud, data consumer, or other organizations. This includes protecting: (i) data confidentiality, meaning that raw data must remain encrypted; (ii) key confidentiality, meaning that private keys used for decryption must be protected from unauthorized access; and (iii) result confidentiality, meaning that the final prediction output must not reveal sensitive information about individual data providers.

Formally, for any provider $P_i \in \mathcal{P}$ with dataset D_i ($i \in \{1, \dots, n\}$) and any consumer $C_j \in \mathcal{C}$ ($j \in \{1, \dots, k\}$), that acts as an adversary, $\mathcal{A}$, the probability of learning unauthorized information about D_i is negligible, as shown in Equation (1).

$$\Pr[\mathcal{A} \text{ learns information about } D_i] \leq \text{negl}(\lambda) \quad (1)$$

Likewise, the probability that $\mathcal{A}$ obtains the private key, sk , is negligible too. This can be expressed as in Equation (2).

$$\Pr[\mathcal{A} \text{ obtains } sk] \leq \text{negl}(\lambda) \quad (2)$$

- **Integrity:** Data cannot be tampered with during computation. This includes: (i) data integrity, ensuring that the encrypted data is not altered during transmission or storage; (ii) computation integrity, ensuring that the data processing generates correct results and that no malicious code is executed; and (iii) result integrity, ensuring that the final result is accurate and has not been manipulated.

Formally, for any adversary $\mathcal{A}$, the probability of successfully causing an unauthorized modification of an encrypted dataset, the evaluated computation, or the final result without detection must be negligible. We express this in Equations (3), (4) and (5), respectively.

$$\Pr[\mathcal{A} \text{ causes an undetected modification of } D_i] \leq \text{negl}(\lambda) \quad \forall D_i, i \in \{1, \dots, n\} \quad (3)$$

$$\Pr[\mathcal{A} \text{ causes an undetected modification of } Eval(f)] \leq \text{negl}(\lambda) \quad (4)$$

$$\Pr[\mathcal{A} \text{ causes an undetected modification of } R_f] \leq \text{negl}(\lambda) \quad (5)$$

- **Availability:** The system must remain operational even under denial-of-service attacks. This includes: (i) data availability: the encrypted data must be accessible for computation; and (ii) computational availability: the system must be able to perform computations without interruption. Availability is treated as an operational security goal rather than as a confidentiality property.

The most central threat is private key exposure, which would allow an adversary (e.g., a malicious data consumer) to decrypt all encrypted data and access sensitive information.

To address this issue, a privacy-preserving multi-source aggregation approach is required. In this context, a solution which allows a consumer (the requester organization) to execute a given algorithm (predictive algorithm) over multi-source confidential datasets (machine operational data) from different providers (organizations) is needed, avoiding raw data disclosure. Instead of pooling raw datasets, the proposed solution is based on the use of HE to guarantee the confidential execution of the predictive algorithm over

encrypted confidential datasets. As a result, data confidentiality is guaranteed between data providers and data consumer (or against any additional third party). This solution allows all providers to contribute knowledge without exposing confidential information resulting in a better trained predictive algorithm.

3.2. Overview

HE allows computations to be performed directly on encrypted data, ensuring that sensitive information from each data source remains protected at all times. By using this approach, multiple organizations can collaboratively contribute encrypted data to a shared computation without ever exposing the underlying raw information.

One of the important limitations of HE is that all the data considered for a given execution should share a common cryptographic context (CC), which includes parameters such as the polynomial degree, plaintext modulus, and multiplicative depth. These parameters define the operational boundaries of the HE scheme and ensure that encrypted values from different sources are compatible when performing joint computation [45]. Alongside the context, all data sources must also use the same public HE key, which allows them to produce ciphertexts that can be securely aggregated, compared, and processed [46]. However, this requirement introduces a fundamental challenge: the need for a shared encryption key and unified CC across all participating data sources. If each source were to use its own independent key pair or incompatible parameters, the resulting ciphertexts would be mathematically incompatible and could not be jointly evaluated. This would break the multi-source computation pipeline and prevent the system from producing global results [47].

To address this issue, the proposed workflow establishes a centralized trusted entity where HE keys (public pk and private sk) are generated and managed. A single public key together with a consistent CC is securely generated there (i.e., $\text{KeyGen} \rightarrow (CC, pk, sk)$), and then distributed (i.e., $\text{KeySharing} \rightarrow (CC, pk) \rightarrow P_i \quad \forall P_i \in \mathcal{P}$) to all data sources.

Each source uses pk to locally encrypt its data before securely sending the resulting ciphertexts to the analytics engine. The corresponding sk needed for the result decryption is held strictly by the designated trusted entity, ensuring that no single party can decrypt the results unilaterally. In this context, for a plaintext $x \in \mathbb{R}^d$, encrypting it is written as $c = \text{Enc}(pk, x)$, and then decryption $\text{Dec}(sk, c) = x$ follows only when pk was derived from sk in KeyGen.

By unifying the encryption context and key material, homomorphic operations become feasible across encrypted datasets originating from different sources. In this context, the first step is the concatenation of the ciphertexts under a shared public key. For two encrypted datasets $c_1 = \text{Enc}(pk, D_1)$, $c_2 = \text{Enc}(pk, D_2)$, this concatenation can be denoted as in Equation (6)

$$C_{\text{concat}} = c_1 \| c_2 \text{ such that } \text{Dec}(sk, C_{\text{concat}}) = \text{Dec}(sk, c_1) \| \text{Dec}(sk, c_2) = D_1 \| D_2, \quad (6)$$

where $\|$ denotes the usual concatenation operation.

Once the encrypted datasets from different sources have been concatenated under the shared public key, the aggregation function f , which is a combination of additions and products, is applied as $c_f = \text{Eval}(f, C_{\text{concat}})$, such that the decrypted result satisfies $R_f = \text{Dec}(sk, c_f) = f(D_1 \cup D_2)$.

This can be achieved if the centralized trusted entity has a secure enclave, so that the private key is stored securely and is never accessible. Consequently, none of the originating organizations will be able to access it or, therefore, decrypt the data of the others. In this way, in addition to ensuring confidentiality vis-à-vis the executor and other third parties (which is already ensured through the use of secret-key encryption), managing the private key within a secure enclave guarantees confidentiality among all source organizations and the secure decryption of the result. For an encrypted result c_f , compute $\text{Dec}(sk, c_f) = R_f$.

To illustrate the aggregation process, consider a simple example with two data providers. Provider P1 holds a dataset D1 with patient records:

$$D1 = \{(age = 45, bloodpressure = 120), (age = 50, bloodpressure = 130)\}$$

Provider P2 holds a dataset D2 with the same structure:

$$D2 = \{(age = 60, bloodpressure = 140), (age = 55, bloodpressure = 135)\}$$

Before encryption, both datasets are assumed to be aligned in terms of schema and data representation (i.e., same attributes and units). Each provider encrypts its dataset using the shared public key, obtaining ciphertexts c_1 and c_2 . These ciphertexts are then concatenated into a single ciphertext structure:

$$C_{\text{concat}} = \text{Concatenate}(c_1, c_2) = \{(45, 120), (50, 130), (60, 140), (55, 135)\}$$

A given function f , such as the average blood pressure across all records, is then evaluated over the concatenated ciphertext:

$$c_f = \text{Eval}(f, C_{\text{concat}}) = (52.5, 131.25)$$

Finally, the trusted entity decrypts the result to obtain the plain values. In this example, the decrypted result corresponds to the average of the four blood pressure values (120, 130, 140, 135), obtained without revealing any individual data points. This workflow assumes that schema alignment is performed prior to encryption.

3.3. Architecture

Based on the proposed approach, the suggested privacy-preserving multi-source aggregation and confidential analytics framework is organized into three core components, as shown in Figure 1. The architecture is designed to ensure the three fundamental security goals: confidentiality, integrity, and availability. Confidentiality is guaranteed through homomorphic encryption and secure key management within a secure enclave. Integrity is ensured by restricting decryption and key usage to controlled environments and by preserving correctness responsibilities at the computation layer. Availability is addressed through the distributed nature of data providers and consumers, avoiding centralized bottlenecks.

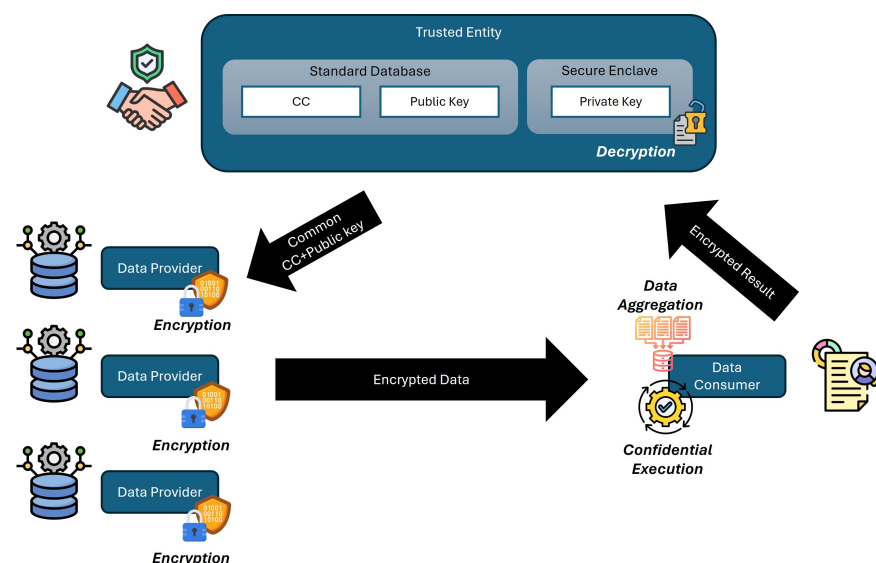


Figure 1. Privacy-preserving multi-source aggregation framework architecture and components.

- **Trusted entity:** This subcomponent is in charge of the general computation management. Additionally, it generates the common HE keys (including the public key for the encryption process and the private key for the decryption process) as well as the suitable common HE context depending on the specific algorithm to be applied. The HE context and public keys will be used for the data encryption of all data providers involved in the aggregation and confidential information. In addition, it will be securely generated, maintaining the private key in a secure cryptography module (secure enclave) while storing the public key and the HE context on a standard database. As the private key remains securely stored inside the secure enclave, the decryption process which requires the use of this private key also happens inside it to always maintain the privacy of the private key. The trusted entity is not trusted for correctness of computations, system availability, or access control beyond key management. Its trust is strictly limited to secure key generation and protection. Private keys are never exposed outside the secure enclave under any circumstances, and all decryption operations are strictly confined within the secure enclave.
- **Data provider:** This subcomponent, required by each data source, is in charge of the local HE process with the common public key and HE context (obtained from the standard database). As a result, all source datasets will be homomorphically encrypted in a proper common way to be concatenated under a shared public key and considered for a given operation. This subcomponent needs to be distributed over the different involved data sources.
- **Data consumer:** This subcomponent is in charge of the encrypted data aggregation as well as the operations/algorithms confidential execution on the homomorphic domain considering the aggregated homomorphically encrypted datasets from different data providers (sources). As a result, the output of the operation is obtained, also encrypted, and needs to be sent back to the trusted entity for the decryption process (as it is where the private key is securely stored). This subcomponent needs to be distributed over the different involved data consumers/executors. The data consumer is responsible for ensuring the correctness and reliability of computations performed over encrypted data, as the trusted entity does not verify computational integrity.

In this architecture, it is essential that the execution of computations does not occur within the trusted entity, as doing so could undermine the fundamental privacy guarantees the system aims to provide. The trusted entity is designed with a minimal and controlled role: it generates the homomorphic keys, establishes the CC, and securely stores the private key inside the protected TEE for the final decryption. By confining the trusted entity's responsibilities to key-related operations, the system ensures that no sensitive encrypted datasets (the original datasets) from the data providers ever need to be uploaded or processed within the same environment that holds the decryption capability. If computations were executed inside the trusted entity, encrypted data from multiple sources would have to be transferred into a location where the private key is physically stored. Even if the encrypted data remained encrypted during computation, co-locating ciphertexts and the decryption key would significantly increase the security risk. A vulnerability or misconfiguration of the secure enclave could give an adversary an opportunity to access both the data and the decryption capability. This violates a core design principle of homomorphic architectures: the entity holding the private key must never process the data. Separating computation from the trusted entity also reduces the attack surface. The data consumer performs all operations on ciphertexts in a domain where no secret key exists, ensuring that even a full compromise of the computation environment would not expose any plaintext. Only the final encrypted result, which is significantly less sensitive than raw datasets, is transferred back to the trusted entity for decryption inside the secure enclave. Regarding

failure and threat assumptions, the system considers the possibility of a partial compromise of the trusted entity. However, even under such conditions, hardware-level protections provided by the secure enclave ensure that private keys remain protected and inaccessible. Therefore, confidentiality of the data is preserved even in degraded trust scenarios.

Additionally, isolating the trusted entity from computation supports avoids concentrating computational effort on the component responsible for key management. Data-intensive operations, such as machine learning model training, require considerable computational resources. If different data consumers make simultaneous multi-source computation requests, the workload would be heavy and the system efficiency would be severely degraded, even becoming saturated and limiting the number of operations it can support. On the contrary, the proposed design distributes the computational effort by allowing each data consumer to perform confidential computations locally on encrypted data. This architectural choice supports a more flexible distribution of computational resources, as the most demanding operations are executed at the consumer side rather than within the trusted entity.

Overall, keeping computation outside the trusted entity is crucial for maintaining strong privacy guarantees, reducing security risk, and improving performance. This architectural separation ensures that no single component has access to both sensitive data and the private key, thereby preserving the confidentiality of all data sources throughout the entire multi-source aggregation workflow.

Taking everything into consideration, Figure 2 illustrates the lifecycle of data and cryptographic material across the different domains: the Provider, the Untrusted Zone, and the Consumer. The Trusted Zone (secure enclave), shaded in blue, establishes a hardware-enforced boundary where sensitive operations such as HE key generation and decryption are performed. Data transitions are explicitly marked: plaintext local data is encrypted by the Provider before crossing into the untrusted zone, while the private key never leaves the trusted boundary. The red dotted lines highlight the critical boundary crossings, where public crypto-material enters or exits the enclave, and where the final clear result is delivered to the consumer, ensuring that plaintext sensitive information is only exposed within the secure enclave or at the final authorized endpoint.

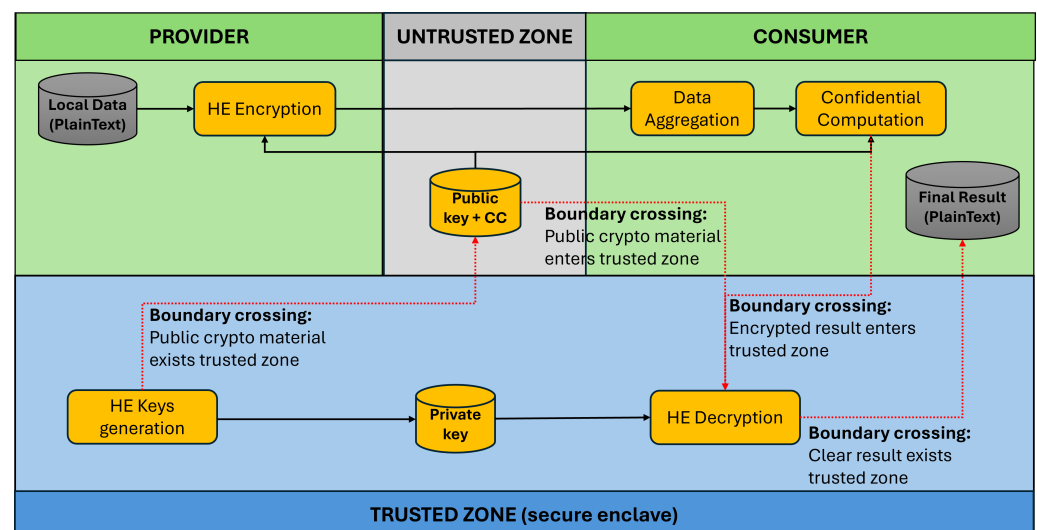


Figure 2. End-to-end security data flow and trust boundary analysis for multiple data providers participating in a common aggregation process.

Although a single provider is represented for simplicity, the same workflow applies to every participating data provider. The framework is designed to support an arbitrary number of independent providers, all of which share the same homomorphic context and public key for a given aggregation process.

3.4. Operation Flow

The suggested privacy-preserving multi-source aggregation and confidential analytics framework requires two main operations.

The first process, depicted in Figure 3, is executed by data providers to register their datasets within the framework, thus enabling their availability for future confidential operations. To this end, data providers must first locally deploy and execute the Data provider subcomponent of the framework. Once operational, the dataset intended for sharing is stored within this subcomponent (step 1). In parallel, the dataset is registered within the trusted entity (step 2), where its specifications and descriptive metadata, such as size, data type, structure, and semantics, are defined. This registration enables future data consumers to discover the dataset and to access it in a correct and controlled manner. The endpoint information of the deployed Data provider subcomponent is also registered and associated with the corresponding dataset, allowing the framework to notify the data provider whenever interactions or operations involving that dataset are required. Once all the information has been successfully registered, the trusted entity responds with a positive acknowledgment (step 3) to inform the data provider that the registration was successful.

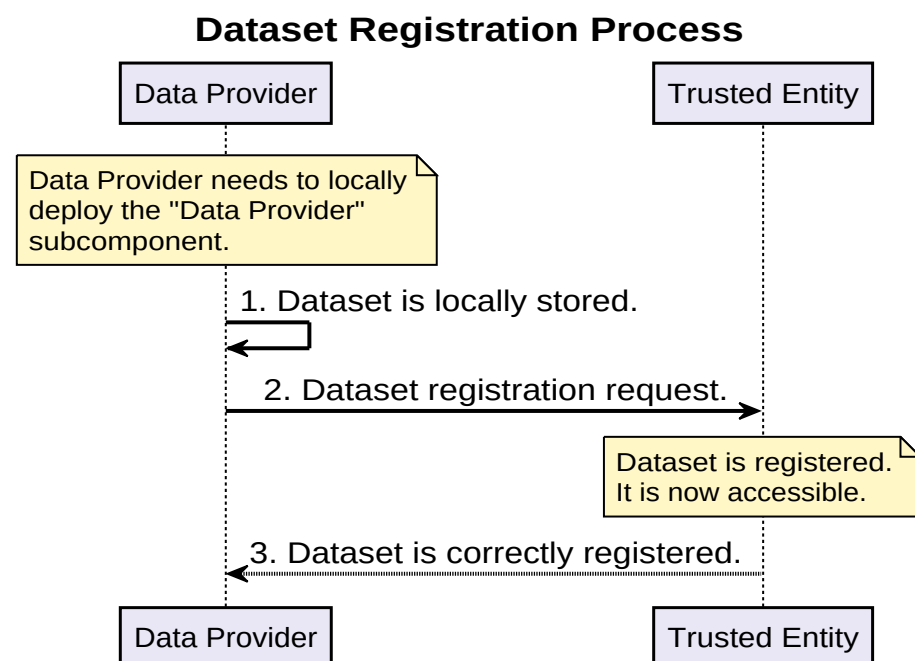


Figure 3. Multi-source dataset registration process.

The registration process is independently executed by each participating data provider. Although Figure 3 illustrates the registration of a single dataset for clarity, the framework supports the registration of multiple datasets originating from different organizations, which can later participate in the same aggregation workflow.

The second process is related to the confidential execution of algorithms or computations over concatenated datasets from different sources. In this case, the data consumer must first deploy and run locally the Data consumer subcomponent of the framework. Additionally, the registered data providers still need to keep the Data provider subcomponent locally running. Figures 4 and 5 show the main phases and steps of this process:

- **Computation Request:** The process begins when the data consumer submits a specific computation request to the trusted entity (step 1). This request includes the specifications of the desired datasets, expressed in terms of metadata. The trusted entity then searches among the registered datasets to identify those that match both the data

consumer's specifications and the requirements of the selected computation (e.g., data size, data type, data semantics) (step 2). The resulting list of suitable datasets is returned to the data consumer (step 3) for final selection (step 4).

At this stage, the trusted entity generates an HE key pair for the specific computation and selected data providers (step 5). The private key is securely stored within the secure enclave, while the public key and the CC are stored in the standard database. Once the key pair is generated, the trusted entity notifies the selected data providers so they can begin the encryption process (step 6).

- **Local datasets HE Encryption:** Once the selected data providers are notified, they retrieve the required CC and public key from the trusted entity's standard database (step 7) to perform the local HE encryption of their datasets (step 8). After completing the encryption, each data provider notifies the trusted entity (step 9). This notification also specifies the method for retrieving the encrypted data from each provider. The trusted entity then waits until notifications from all selected data providers have been received.
- **Local Confidential Computation:** Once the trusted entity has received notifications from all selected data providers, it notifies the data consumer (step 10) to initiate the data aggregation process and provides the corresponding mechanisms for retrieving the encrypted data from each provider. At this stage, the data consumer requests the required public cryptographic material to the trusted entity (step 11) and directly retrieves the encrypted data from each selected data provider using the specified method in each case (step 12).

After all encrypted datasets have been collected (step 13), the multi-source data aggregation process is carried out (step 14). This process consists of merging all datasets from different sources into a single unified dataset based on their data types and semantic information. If only one dataset was selected, the aggregation is bypassed. Then, the data consumer locally executes the confidential computation over the concatenated encrypted data (step 15).

- **Secure Result HE Decryption:** Once the confidential computation is completed, an encrypted result is obtained by the data consumer. This result must then be sent to the trusted entity for secure decryption within the secure enclave (step 16). The decryption is performed inside the enclave (step 17), and the plaintext result is returned to the data consumer (step 18). In this way, the data consumer obtains the final computation result over the multi-source aggregated data without ever accessing any of the raw datasets.

Overall, this scheme enables a data consumer to request a specific aggregation, which is then securely executed over data previously registered by multiple data providers. In this way, the framework supports confidential multi-source analytics without requiring data providers to disclose their raw data or relying on a trusted central aggregator.

3.5. Security Considerations

3.5.1. Security Model

To ensure the robustness of the proposed framework, the security model is described in terms of adversary capabilities, collusion risks, and integrity threats. It is important to note that this section provides an informal security analysis of the composed system rather than a complete formal cryptographic proof. The analysis is modular: it inherits the security guarantees of the underlying primitives (homomorphic encryption, digital signatures, TLS) and of the trusted execution environment (TEE) under its declared threat model, and reasons about how these guarantees compose in the proposed architecture.

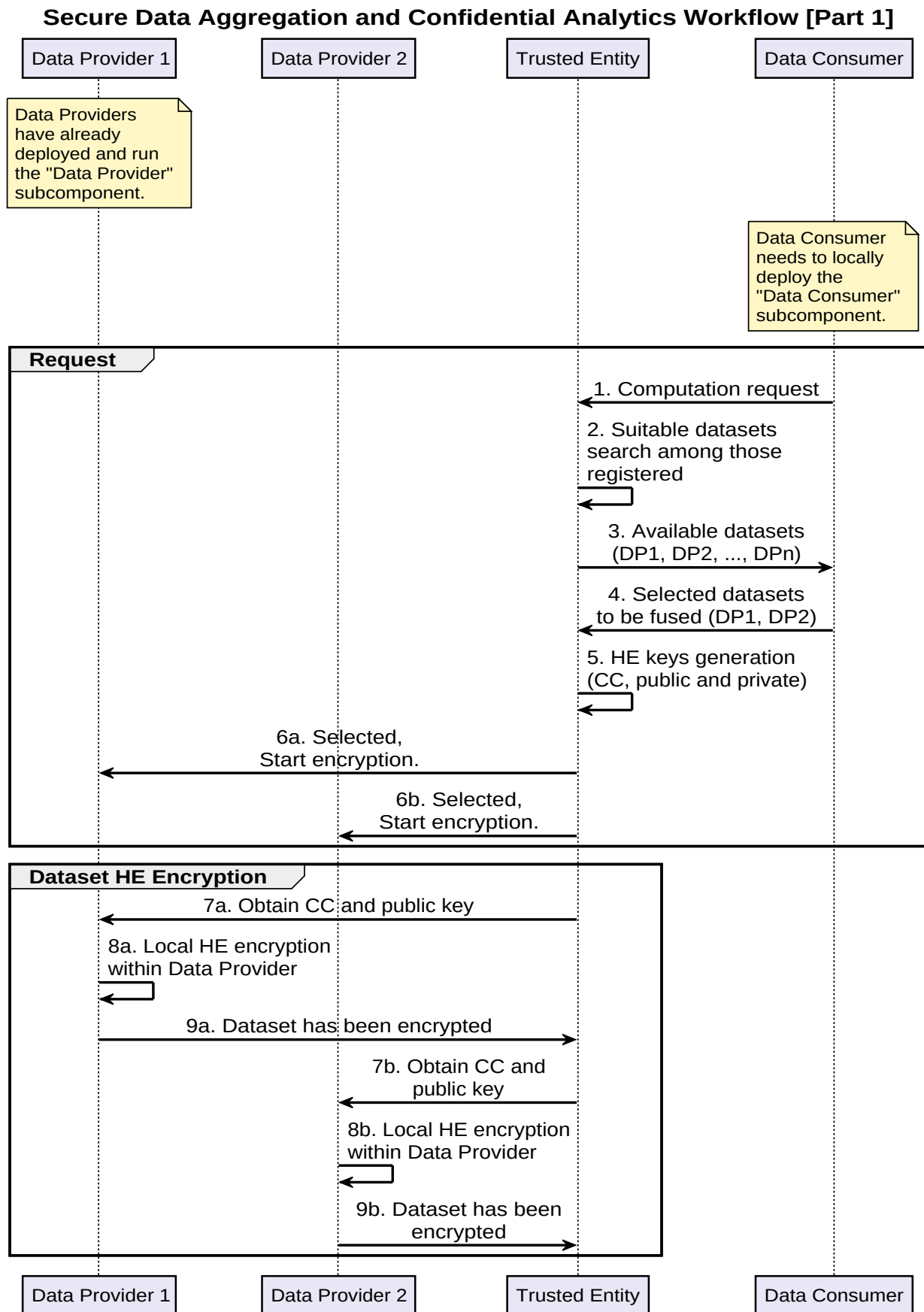


Figure 4. Distributed confidential computation process [Part 1].

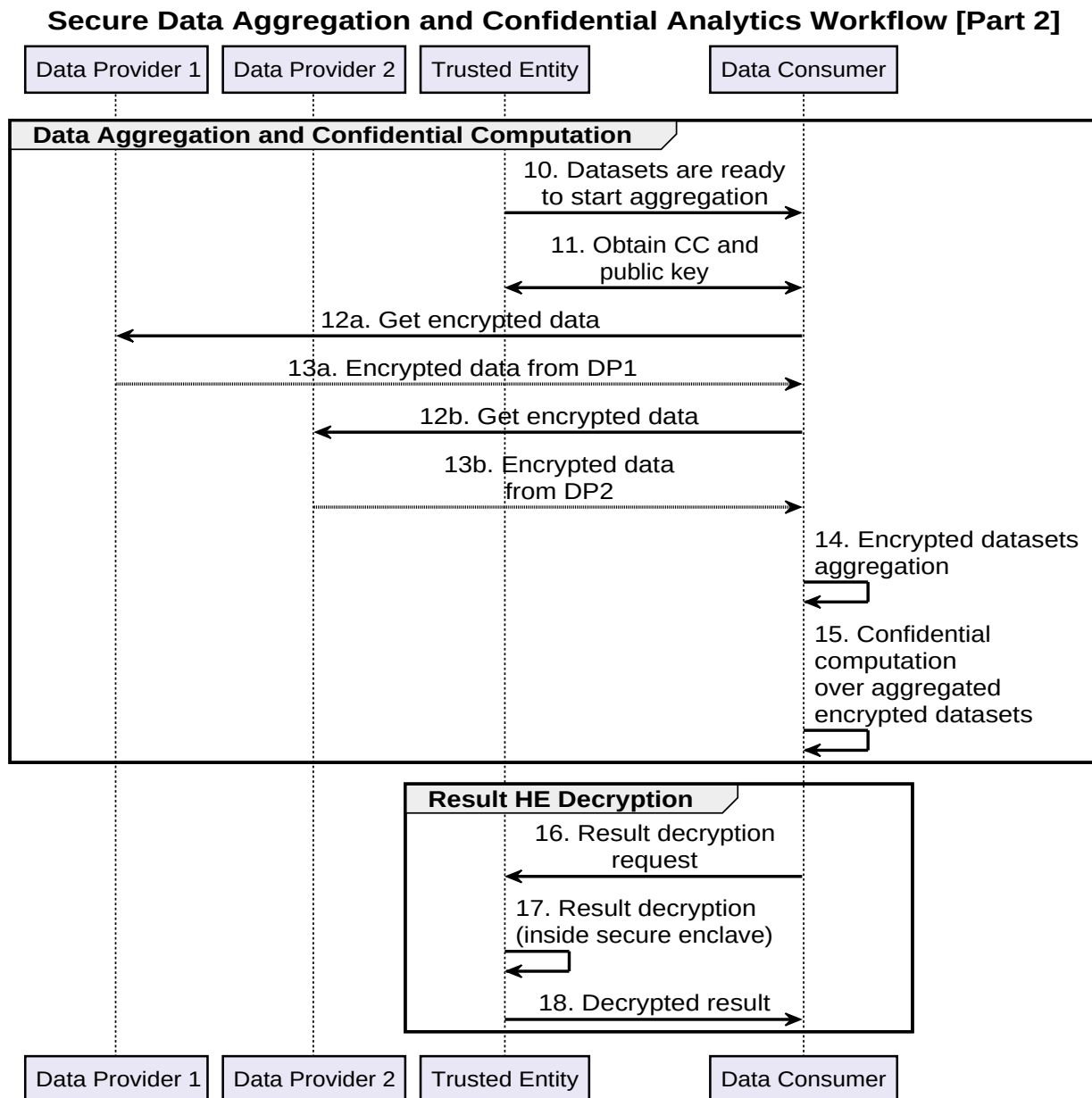


Figure 5. Distributed confidential computation process [Part 2].

Different types of attackers can be considered controlling the following roles in the framework: (i) data providers, (ii) data consumers, and (iii) the untrusted infrastructure of the trusted entity. However, the TEE is assumed to provide confidentiality and integrity guarantees as defined by its threat model, excluding known side-channel attacks (e.g., Spectre, Meltdown, Foreshadow, and L1TF) as the proposed framework is designed under the assumption that the underlying hardware is deployed with the latest microcode updates and software mitigations provided by the manufacturer.

Within the trust assumptions above, the following classes of adversaries are considered:

- Semi-honest adversaries: follow the protocol but attempt to infer additional information from observed data.
- Malicious adversaries: arbitrarily deviate from the protocol, including sending malformed inputs or manipulating execution.
- Compromised components: system entities (e.g., cloud operator) whose software stack is fully controlled by the adversary, excluding the TEE boundary.

Under these assumptions, the framework aims to achieve the following properties. The claims are stated informally; their negligible-advantage formulations are inherited from the underlying primitives and are not proven end-to-end in this work.

- **Data confidentiality:** The adversary learns no information about plaintext inputs beyond what is revealed by the final output. This follows from the semantic security of the HE scheme, under which any probabilistic polynomial-time adversary has advantage negligible in the security parameter λ in distinguishing ciphertexts.
- **Computation integrity:** The result returned corresponds to the correct evaluation of the function over the provided inputs, except with probability negligible in λ , assuming the correctness of the HE evaluation and the integrity guarantees of the enclave.
- **Key confidentiality:** Secret keys remain confined within the TEE and are never exposed to external components, under the enclave isolation assumption.

Limited collusion among system participants has been taken into account. The following cases have been identified as priorities.

- **Consumer–Provider Collusion:** A data consumer colluding with one or more data providers may attempt to infer other providers' inputs. However, under the semantic security of HE, ciphertexts do not reveal plaintext information, and the decryption key is not accessible outside the enclave. Thus, the adversary's advantage remains negligible.
- **Consumer-Trusted Entity Collusion:** A malicious consumer colluding with a compromised trusted entity may attempt to extract keys or tamper with computation. The TEE enforces isolation of code and data, and remote attestation ensures that only verified enclave code is executed. Therefore, confidentiality of keys and correctness of execution hold under the TEE security assumptions.
- **Provider-Trusted Entity Collusion:** A malicious provider colluding with the trusted entity may attempt to manipulate inputs or influence computation. While such adversaries can inject malformed ciphertexts, they cannot decrypt other participants' data. Integrity protections limit the impact of such attacks.

Full compromise of the TEE or advanced side-channel attacks have not been considered, as they remain an open challenge in the literature.

The adversary assumptions are explicitly assigned per system role. Unless otherwise stated, data providers and data consumers are assumed to be semi-honest, while the infrastructure outside the trusted execution environment is considered potentially compromised. The TEE itself is assumed to provide confidentiality and integrity guarantees according to its threat model. The security guarantees achieved under each assumption are summarized in Table 1.

Table 1. Adversary assumptions and security guarantees per system role.

Role	Adversary Model	Assumptions	Security Guarantees
Data Providers	Semi-honest	Follows protocol, may try to infer information	Confidentiality of other providers data
Data Consumers	Semi-honest or malicious	May deviate from protocol, no access to private key	No access to private keys or plaintext results
Trusted Entity (outside TEE)	Compromised	Full software control by adversary	No access to private keys or plaintext results
TEE	Trusted	Enforces isolation and integrity	Key confidentiality and correct decryption

3.5.2. Security Solutions

To ensure robustness against malicious inputs, the system enforces:

- **Authenticated submissions:** All ciphertexts must be signed by authorized data providers. This prevents unauthorized data injection.
- **Ciphertext validity checks:** Inputs are verified to conform to expected HE parameters, preventing malformed ciphertext attacks.
- **Availability protection:** Input size limits and validation mitigate denial-of-service attacks.

These mechanisms ensure that any deviation from correct execution can be detected with high probability.

Keys are generated inside the TEE and bound to a specific computation instance.

- **Ephemeral keys:** Each computation uses fresh HE keys, limiting exposure in case of compromise.
- **Secure storage:** Secret keys remain inside the enclave throughout their lifecycle.
- **Secure distribution:** Public keys are distributed over authenticated and encrypted channels (e.g., TLS).
- **Freshness guarantees:** Nonces and timestamps can be used to prevent replay attacks and ensure uniqueness of computation sessions.

Standard cryptographic primitives (HE, signatures, TLS) are assumed to be secure. In addition, two fundamental security primitives provided by TEEs architectures are considered:

- **Enclave Isolation:** The framework relies on the hardware-enforced isolation that partitions the application into a trusted enclave and an untrusted host. This ensures that the enclave's memory is encrypted and inaccessible to any software running outside the enclave, including privileged layers such as the OS, Hypervisor, or BIOS. Consequently, private keys are only exposed within the processor's secure boundary.
- **Remote Attestation:** To establish trust with external entities, remote attestation can be considered. This mechanism allows a remote party to verify that the correct, untampered code has been instantiated within a genuine enclave, ensuring that the execution environment has not been compromised before sensitive data is generated.

Portability and Migration Path

Although this implementation utilizes Intel SGX, the architecture is designed to be TEE-agnostic. The trust boundaries established in this work can be mapped to alternative TEEs with minimal modification to the core logic. For instance, a migration to AMD SEV-SNP would involve shifting the isolation boundary from the application level to the Virtual Machine (VM) level (Confidential VMs). Similarly, a transition to ARM TrustZone would require partitioning the system into a "Secure World" and "Normal World," maintaining the same logical separation between the encrypted data flow and the secure processing unit.

For the key management, the standard database plays a central role in the architecture, storing the public key, HE context, and metadata for data providers. To ensure its security, the following mechanisms are assumed:

- **Authentication:** The database is access-controlled and requires secure authentication (e.g., OAuth, API keys) to prevent unauthorized access.
- **Authorization:** Only authorized data providers and data consumers can access the database. Role-based access control (RBAC) ensures that only authorized entities can retrieve or modify data.
- **Tamper resistance:** The database can be protected with encryption or integrity checks to ensure that no unauthorized modifications can be made. Any attempt to tamper with the data will be detected and rejected.

In practical deployments, the system must handle security failures such as delayed or missing inputs, node unavailability, or malformed ciphertexts. In the proposed design, these situations can be addressed through simple system-level mechanisms, including timeout-based exclusion of unresponsive participants, validation of ciphertext structure and parameters prior to computation, and optional retry mechanisms for failed submissions. Furthermore, the framework could be extended to support partial participation, where computations proceed with a subset of available input datasets.

The argument above is an informal, modular security analysis: each property is justified by appealing to the standard guarantees of the underlying primitives (HE semantic security, signature unforgeability, TLS channel security) and of the TEE threat model, and by reasoning about how these compose in the proposed architecture. A reductionist proof of the full protocol in a formal security model is not provided.

3.6. Suggested Implementation

3.6.1. Selected Technologies

The core of the framework is the HE layer, which guarantees confidentiality both during data transfer between data providers and data consumers and during the execution of aggregated computations over the providers' data. On the one hand, it enables secure processing without requiring access to the underlying plaintext, thereby preserving data privacy throughout the entire workflow. On the other hand, it supports data aggregation across multiple data providers, provided that the same HE public key is used to encrypt all data sources.

In this context, symmetric cryptographic schemes are not suitable, since the same private key would be used for both encryption and decryption. This would allow any participating data provider to decrypt the data of other providers, resulting in potential data leakage. To mitigate this risk, asymmetric cryptographic mechanisms are employed, where a public key is used for encryption and the corresponding private key is securely managed by a trusted entity. This design ensures that data providers cannot decrypt each other's data while still enabling joint computation over encrypted datasets.

Taking this into consideration, two different asymmetric HE libraries have been considered to support a broad range of confidential operations: OpenFHE v.1.1.2 [34] and TFHE-rs v.0.10.0 [17]. OpenFHE is particularly well suited for complex numerical and mathematical computations. It supports advanced functionalities such as Chebyshev polynomial approximations [48], which enable the secure evaluation of non-trivial operations including divisions, square roots, and logistic functions over encrypted data. Moreover, OpenFHE benefits from a large and active developer community, with frequent updates and continuous improvements. In particular, the CKKS scheme is used as it is specifically designed for the efficient handling of real and approximate numbers, maintaining a strong balance between precision, performance, and resource efficiency, which is essential for data analytics or machine learning. However, OpenFHE does not natively support the evaluation of logical gates using public keys, which are required for comparison operations or binary logic. To address this limitation, TFHE-rs has been also integrated in the framework. TFHE-rs is currently the only HE framework that efficiently supports the evaluation of logical gates and binary operations under public-key encryption, making it especially suitable for comparison-based computations. The complementary capabilities of OpenFHE and TFHE-rs allow the privacy-preserving multi-source aggregation and confidential analytics framework to support a wide range of confidential computation scenarios, combining efficient numerical processing with secure logical and comparison operations.

A notable limitation of HE is the large size of some of the HE keys, which depends on the HE configuration parameters. For example, a public key in the case of TFHE-rs can

reach up to several GBs [49]. This poses challenges for key distribution among participants and for key generation within secure enclaves, which are generally not designed to handle such large files. Among all the existing options for a secure enclave (e.g., ARM TrustZone [50], Intel SGX [51], or the use of a Hardware Security Module [52]), Intel SGX Intel Corporation, Santa Clara, CA, USA) was considered due to its high potential to support HE algorithms [53,54] (it fits in the description of security level 3 and hybrid software module from Section 2.3). Although Intel SGX has been officially deprecated, its use is still justified in this context due to several factors. First, SGX provides a mature and well-understood TEE that supports the secure generation, storage, and processing of sensitive cryptographic keys. Second, it offers a practical solution for the secure management of large homomorphic keys within a controlled environment, a functionality that is not yet fully supported by alternative TEEs or secure enclave technologies. Finally, ongoing research and platform updates allow for mitigation of known vulnerabilities, enabling SGX to provide a secure environment for experimental and prototype deployments until fully supported alternatives become available.

While the proposed implementation considers Intel SGX, the underlying architecture is designed to be TEE-agnostic. The security logic based on the isolation of cryptographic keys and the execution of the sensitive decryption operation within a trusted boundary is not tied to a specific hardware vendor but to the general properties of TEEs. Consequently, the framework can be migrated to alternative TEE architectures with minimal modifications to the core protocol:

- **Migration to AMD:** Transitioning to AMD's Secure Encrypted Virtualization-Secure Nested Paging (SEV-SNP) would shift the isolation granularity. While Intel SGX operates at the application level (enclaves), SEV-SNP operates at the Virtual Machine (VM) level. In this scenario, the entire VM becomes a "Confidential VM," simplifying the development process by removing the need to partition the application into trusted and untrusted components, although it expands the Trusted Computing Base (TCB) to include the guest OS.
- **Migration to ARM TrustZone:** A migration to ARM TrustZone would require a different partitioning strategy. TrustZone implements a system-wide approach by creating two parallel worlds: the "Secure World" and the "Normal World." The framework's key management and decryption logic would be hosted in the Secure World, while the rest of the data aggregation pipeline would remain in the Normal World, interacting via a secure monitor call (SMC).

Table 2 provides a comparative summary of how these alternative TEEs affect the trust boundaries and the TCB of the proposed framework.

By decoupling the framework's security logic from the specific implementation of the enclave, the system remains resilient to hardware-specific vulnerabilities and adaptable to the evolving landscape of confidential computing.

Table 2. Comparison of TEE Architectures.

Feature	Intel SGX	AMD SEV-SNP	ARM TrustZone
Isolation Level	Process/Application	Virtual Machine (VM)	System-wide (World)
Trust Boundary	Enclave Boundary	VM Boundary	Secure World Boundary
TCB Size	Smallest (Application)	Larger (Guest OS)	Moderate (Secure OS)
Migration Effort	Baseline	Low (VM-based)	Medium (Partitioning)
Key Protection	Hardware-enforced	Hypervisor-isolated	Hardware-partitioned

3.6.2. Trusted Entity

The Trusted Entity is the subcomponent responsible for managing multi-source data aggregation and coordinating confidential computations. In addition, it generates and securely stores the HE key pairs required for each computation. To this end, it exposes a RESTful API that provides the following main functionalities:

- **Data Registration Request:** The trusted entity receives data registration requests from data providers (data sources), which include all the information required for the accurate characterization of the dataset. This information comprises metadata, column descriptions, data types, and semantic definitions. Upon successful validation, the dataset is registered within the framework, thereby becoming discoverable and accessible for subsequent confidential analytics.
- **Computation Request:** The trusted entity receives computation requests from data consumers, specifying the desired computation together with the requirements that the input data must satisfy in terms of metadata and other constraints. Based on this information, the trusted entity searches among the registered datasets to identify those that are compatible with the selected computation and that meet the specified requirements.
- **Dataset Selection Notification:** The trusted entity receives from the data consumer the list of datasets selected among those previously identified as applicable to the requested computation. As a result, the trusted entity generates a new HE key pair and CC specific to this computation and internally verifies the information of the data providers associated with the selected datasets (in the case of TFHE-rs, the CC is not required as the parameters equivalent to a CC are directly linked to the key, and each key contains all the information necessary to operate). This information is then used to notify the corresponding data providers to initiate the encryption process.
- **Public Key and CC Request:** The trusted entity receives requests from the selected data providers to obtain the public key and CC (when needed) which are required to correctly encrypt the selected datasets. This request is also received from the data consumer to obtain the public cryptographic material required for the confidential computation. In response, the trusted entity provides the requested cryptographic material, which has been previously generated for that computation.
- **Finished Encryption Notification:** The trusted entity receives a notification from each selected data provider indicating that the encryption of the corresponding dataset has been successfully completed. The trusted entity waits until such notifications have been received from all selected data providers before proceeding. Once all encrypted datasets are available, it notifies the data consumer who initiated the process, thereby enabling the beginning of the computation phase.
- **Result Decryption Request:** The trusted entity receives a request to decrypt the encrypted result produced by the confidential computation executed by the data consumer. Upon reception, the decryption process is initiated within the secure enclave of the trusted entity, ensuring that the plaintext result is recovered in a protected and trusted execution environment.
- **Other Endpoints:** The trusted entity exposes additional endpoints that enable, for instance, the retrieval of registered information, the monitoring of the computation process status, and the execution of health checks. Although these endpoints are not strictly required for the core operational workflow, they significantly enhance the usability, transparency, and manageability of the component.

3.6.3. Data Provider

The Data provider subcomponent is in charge of the encryption and provision of datasets. For this purpose, it exposes a REST API with the following main functionalities:

- **Start Encryption Notification:** The data provider receives this notification from the trusted entity when a dataset under its ownership has been selected for a computation. Upon reception, the data provider requests from the trusted entity the public key and the CC required to initiate the encryption process.
- **Get Encrypted Data Request:** The data provider receives this request from the data consumer to provide the encrypted data needed to initiate the computation. In response, the data provider provides the corresponding encrypted dataset to the data consumer.
- **Other Endpoints:** The data provider exposes additional endpoints that enable, for instance, the monitoring of the encryption process status, and the execution of health checks. Although these endpoints are not strictly required for the core operational workflow, they significantly enhance the usability, transparency, and manageability of the component.

3.6.4. Data Consumer

The Data consumer subcomponent is responsible for executing multi-source data aggregation and confidential analytics. To support these operations, it exposes a RESTful API with the following primary functionalities:

- **Available Datasets Notification:** The data consumer receives this notification from the trusted entity, identifying the datasets registered on the platform that are suitable for the requested computation and meet the consumer's specified requirements. The data consumer then selects the datasets to be considered for the computation.
- **Start Data Aggregation Notification:** The data consumer receives this notification from the trusted entity to indicate that the selected datasets have been encrypted and are ready for the aggregation process. In response, the data consumer obtains the public cryptographic material from the trusted entity and issues a request for the encrypted data to each of the relevant data providers.
- **Decrypted Result Notification:** The data consumer receives this notification from the trusted entity to obtain the decrypted result of the confidential computation. Consequently, the data consumer gains access to the computation output in plaintext.
- **Other Endpoints:** The data consumer exposes additional endpoints that enable, for example, the monitoring of the aggregation or computation process and the execution of health checks. Although these endpoints are not strictly required for the core operational workflow, they substantially enhance the component's usability, transparency, and manageability.

All REST-based interactions between framework components are assumed to be carried out over authenticated and encrypted communication channels, such as TLS. The concrete authentication mechanism, e.g., mutual TLS, API tokens, or integration with an external identity provider, is deployment-dependent and orthogonal to the FHE/TEE-based aggregation protocol.

Additional implementation details, including example API requests, responses, and communication assumptions, are provided in Appendix A to improve reproducibility.

3.6.5. Computations

The implementation of algorithms within the homomorphic domain allows computations to be performed directly on encrypted data, thereby preserving confidentiality while enabling meaningful operations. Moreover, homomorphic computations enable the outsourcing of processing tasks to untrusted entities, such as external data consumers, without compromising sensitive information. Nonetheless, the deployment of algorithms in this domain introduces significant challenges. Algorithm design must account for inherent constraints, including noise accumulation, limits on operation depth, and the specific

arithmetic operations supported by homomorphic schemes. Consequently, careful consideration of the trade-offs between computational performance, accuracy, and security is essential to achieve practical and effective implementations [48,55].

Implementing an algorithm in the homomorphic domain includes a series of critical steps that differ substantially from conventional algorithm design, due to the inherent constraints and properties of HE. The implementation process typically involves the following stages:

- **Algorithm Selection:** The initial step consists of identifying the target algorithm and analyzing its operations in terms of fundamental arithmetic primitives (addition and multiplication). Given that HE schemes efficiently support only specific operations, it is essential to map the algorithm's logic to operations compatible with the chosen scheme.
- **Mapping to Homomorphic Operations:** The algorithm must be adapted to operate on encrypted data. This often requires restructuring computations to minimize multiplicative depth and mitigate noise accumulation in ciphertexts, as excessive noise may compromise correct decryption [56]. In some cases, approximation techniques are employed when exact arithmetic is infeasible [16].
- **Execution:** Homomorphic operations can be executed sequentially or in parallel, depending on the scheme and algorithmic structure. During execution, careful management of noise growth and ciphertext expansion is crucial to preserve both correctness and computational efficiency.

Overall, implementing algorithms in the homomorphic domain demands careful adaptation of conventional computational workflows, consideration of scheme-specific constraints, and strategic optimizations to ensure secure, accurate, and practical execution. Specifically, the implemented Data consumer subcomponent encompasses a range of generic computations, including both lightweight operations and more complex processing tasks:

- **Statistics (mean, range and standard deviation).** These fundamental statistical operations allow for the characterization and summarization of datasets, providing essential descriptive insights that form the basis for further analysis. They are particularly useful in identifying trends, anomalies, and variability within encrypted data.
- **Summation and Production.** These arithmetic operations are the building blocks of more advanced computations, including aggregation tasks and cumulative metrics. They support a variety of analytical procedures, ranging from simple totals to more complex cumulative computations in machine learning and statistical modelling.
- **Number Comparisons.** Comparison operations enable the evaluation of relative relationships between data points, which is critical for decision-making processes, conditional logic, and threshold-based analyses within encrypted domains.
- **Linear Regression Training and Inference.** Linear models provide an efficient framework for predictive analytics and regression tasks. Their implementation allows the Data Consumer to perform both model training and inference directly on encrypted data, enabling secure predictions while preserving the confidentiality of both training and input datasets.
- **Logistic Regression Training and Inference.** Logistic models extend the predictive capabilities of the system to classification tasks. By implementing both training and inference homomorphically, the Data Consumer can perform privacy-preserving classification, which is particularly relevant in sensitive domains such as healthcare, finance, and security applications.

Collectively, these operations form a versatile computational toolkit within the Data consumer subcomponent, enabling the execution of both basic and advanced analytical tasks while maintaining strong data confidentiality guarantees. The combination of statistical analysis, arithmetic processing, and machine learning capabilities ensures that the subcomponent can handle a broad spectrum of practical applications in secure data processing.

4. Results

This section presents the experimental evaluation of the proposed privacy-preserving multi-source aggregation and confidential analytics framework, focusing on key performance challenges in secure multi-source data aggregation: computational overhead and practical feasibility. The goal is to assess how the framework addresses the core challenge of secure key management and computation in decentralized settings, while providing measurable insights into the trade-offs between security, performance, and usability.

4.1. Experimental Setup

The evaluation was conducted to validate the feasibility of the framework in a realistic multi-source environment. The setup consists of four Docker containers representing the system components: one trusted entity, one data consumer, and two data providers, deployed across three independent machines (Figure 6). This configuration enables the validation of data aggregation across multiple sources while preserving the decentralized trust model.

The objective of this evaluation is not to study scalability with respect to the number of participating data providers, but rather to validate the feasibility of the proposed key management and secure aggregation workflow in a realistic multi-source setting involving independent providers and encrypted computation.

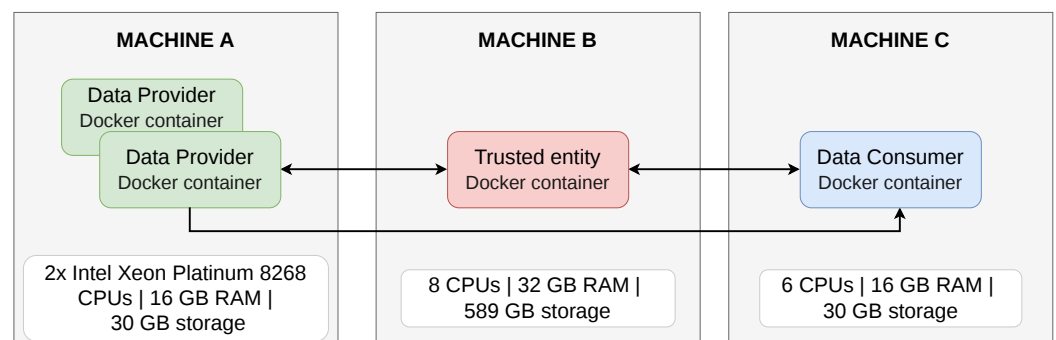


Figure 6. Multi-source data aggregation experimental setup.

The data provider platform is equipped with two Intel® Xeon® Platinum 8268 CPUs operating at 2.90 GHz, 16 GB of RAM, and 30 GB of storage. In contrast, the data consumer platform comprises six identical CPUs, 16 GB of RAM, and 30 GB of storage. This configuration reflects the significantly higher computational demands of the processing phase compared to the encryption phase; therefore, a larger number of CPUs is allocated to the data consumer to accommodate the increased workload. For the trusted entity, a platform with 8 CPUs, 32 GB RAM and 589 GB of storage shared with other services has been considered. An SGX TEE [51] has been considered for the secure enclave.

The following HE libraries have been considered: OpenFHE v.1.1.2 (for floats) and TFHE-rs v.0.10.0 (for Boolean operations). In the particular case of OpenFHE, the following configuration parameters have been considered after qualitative validations by comparing

the encrypted results with the expected outcomes for the representative test cases, ensuring that the approximation error remains within an acceptable range (always lower than 10%):

- Security level: 128 bits. The security level defines the bit-level hardness of the underlying lattice problem and drives the selection of parameters to achieve a target security against known attacks. 128 bits is the standard value considered in OpenFHE.
- Ring Dimension: 32,768. The ring dimension determines the size of the polynomial ring used in the encryption process; it directly influences ciphertext size, computational complexity, and the number of data points that can be encrypted simultaneously with larger values enhancing security at the cost of performance.
- Plaintext Modulus: 50. The plaintext modulus determines the range of values that can be encrypted and processed; in BFV, it directly influences precision and security, while in CKKS, it is replaced by a scaling factor to represent real numbers.
- Scaling Factor: 2^{50} . The scaling factor enables the representation of real numbers as fixed-point integers with larger values enabling higher precision but increasing computational overhead.
- Batch Size: 16,384. The batch size refers to the number of data points that can be encrypted and processed simultaneously in a single ciphertext; it is directly determined by the ring dimension and significantly impacts computational efficiency, with larger batch sizes enabling higher throughput but requiring more memory and processing power.
- Multiplication depth: Variable. The multiplication depth (multDepth) parameter has the greatest impact on the performance. The multDepth defines the maximum number of consecutive homomorphic multiplications that can be performed without refreshing the ciphertext. As multDepth increases, the encryption scheme requires larger cryptographic parameters, such as higher-degree polynomials, to ensure that ciphertexts can support deeper computations without compromising correctness. Additionally, more complex evaluation keys (e.g., relinearization keys) must be generated to accommodate these deeper multiplications. As a result, higher multDepth values substantially increase the computational effort required for HE operation, leading to longer times.

In the case of TFHE-rs, the interesting configuration parameters are different. For the experiments, the following values have been considered:

- Plaintext Modulus: 16. The plaintext modulus defines the range of values that can be encrypted. In this case, 16 distinct values (0 to 15) are allowed to be encrypted.
- Ring Modulus: 1024. The ring modulus determines the number of coefficients in the ciphertexts and the number of data points that can be encrypted simultaneously. The security increases with the ring modulus, but also the complexity.
- Batch size: 512. The batch size refers to the number of data points that can be encrypted and processed simultaneously in a single ciphertext. In TFHE, the batch size is half the ring modulus.

Based on these configuration parameters, the algorithms from Section 3.6.5 have been implemented with the encoding strategy and configuration particularities (mostly focused on the variable multDepth parameter).

- Statistics using real numbers column encoding strategy; OpenFHE with a multDepth of 7 has been considered.
- Summation/Production using real numbers column encoding strategy; OpenFHE with a multDepth of 8 has been considered.
- Comparisons over an integer value: TFHE-rs.

- Linear regression training to predict the value of a real dependent variable based on three independent real variables following a row encoding strategy; OpenFHE with a multDepth of 11 has been considered.
- Linear regression inference of a real variable following a row encoding strategy; OpenFHE with a multDepth of 11 has been considered. The model obtained from the linear regression training is considered.
- Logistic regression training to predict the value of a real dependent variable based on three independent real variables following a row encoding strategy; OpenFHE with a multDepth of 24 has been considered.
- Logistic regression inference of a real variable following a row encoding strategy; OpenFHE with a multDepth of 10 has been considered. The model obtained from the logistic regression training is considered.

All experiments were conducted on small-scale datasets (50 records per provider, 100 total after aggregation), chosen to ensure reproducibility and provide a controlled baseline for performance analysis. Each experiment was repeated 10 times, and the mean and standard deviation of the execution times were calculated to account for system variability. The current setup enables the evaluation of the computational complexity of the key HE phases (key generation, encryption, computation, and decryption) under realistic cryptographic and computational constraints. This approach allows for a more robust and reliable assessment of performance, as it captures variations in execution time due to system load, background processes, and other environmental factors.

4.2. Computational Complexity of Key HE Phases

This section presents the computational complexity results of the key HE phases. Let T_{gen} , T_{enc} , T_{comp} , T_{dec} be the time for keys generation, encryption, computation, and decryption, respectively. The total time for a computation f is:

$$T_f = T_{\text{gen}} + T_{\text{enc}} + T_{\text{comp}} + T_{\text{dec}} \quad (7)$$

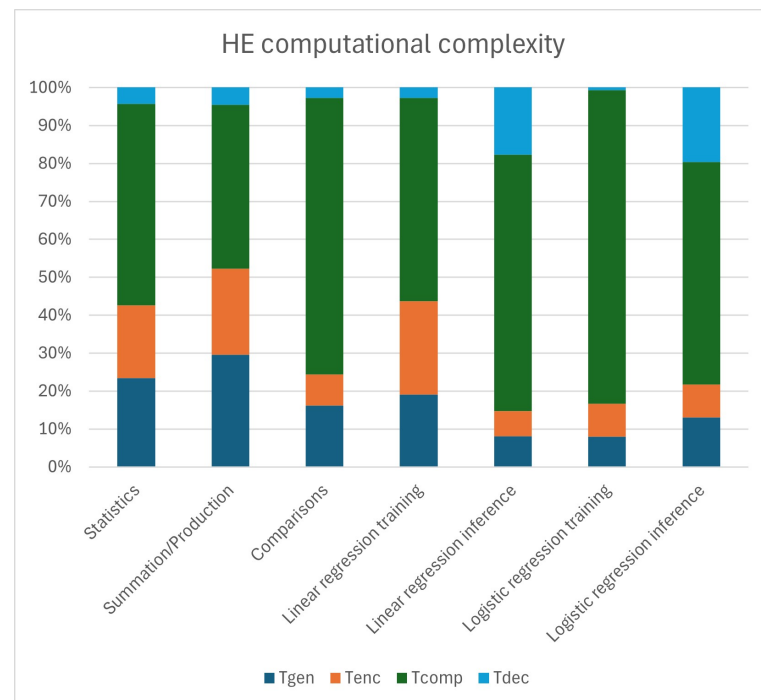
Table 3 summarizes the total end-to-end execution time (T_f) for each evaluated operation. The total time is computed as the sum of the mean key generation, encryption, homomorphic computation, and decryption times, in accordance with Equation (7). This table provides the absolute execution times associated with each operation. We also added their execution time equivalents for performing all these operations in the clear. Even if the overall overhead is considerable in general (see the slowdown ratio in Table 3), the end-to-end execution times with FHE show bearable execution times, most of them below the two minutes of execution.

Figure 7 shows how the computational complexity for each measured operation is distributed in the four main phases: keys generation, encryption, computation and decryption.

For all measured computations, the majority of time is spent on homomorphic computation (T_{comp}), which is represented by the largest green segment in each bar. This indicates that T_{comp} is the primary bottleneck in HE operations. The orange segment (T_{enc}) is not negligible, especially for linear regression training, where it accounts for 15–20% of the total time. The blue segment (T_{dec}) is very small, typically less than 10% of the total time. This is expected, as decryption is a single, fast operation compared to the complexity of homomorphic computation. Finally, the dark blue segment (T_{gen}) often requires a long time of around 20% depending on the configuration parameters.

Table 3. Total end-to-end execution time for each evaluated operation.

Computation	T_f (s)	T_f (s) Without FHE	Slowdown Ratio
Statistics	47.1	0.004654	10,099.84
Summation/Production	44.0	0.007232	6085.07
Comparisons	74.3	0.002910	25,430.55
Linear regression training	110.1	0.007834	14,042.36
Linear regression inference	136.1	0.008832	15,399.55
Logistic regression training	1123.9	0.008602	130,668.29
Logistic regression inference	138.1	0.013384	10,311.82

**Figure 7.** HE main phases computational complexity.

The results shown in the Figure 7 demonstrate that the seven operations can be performed without any issues using the proposed framework. Now, each HE phase is analyzed in detail, identifying the influence of different parameters on the measured results.

4.2.1. T_{gen} : HE Keys Generation

The HE keys (and CC when required) are generated inside the trusted entity. The time required for HE keys generation can differ significantly depending on the underlying library. In the experiments, OpenFHE generally demonstrates faster (and smaller) key generation compared to TFHE-rs. This difference is due to the fact that TFHE-rs is natively a symmetric key scheme; as such, it is possible to turn it into a public-key scheme by using standard conversions [57], but these techniques introduce overhead [58]. This does not happen in OpenFHE, since BGV, BFV and CKKS are public-key schemes per se.

Table 4 presents the mean and standard deviation values of T_{gen} (HE key generation times) for the different computations, considering the defined parameters. As previously noted, higher multDepth values lead to longer key generation times, with the linear and logistic regression training operations exhibiting the most significant impact (up to 90 s of mean time). The standard deviation values are consistently low, indicating high stability and reliability in the execution times across repeated experiments.

Table 4. HE keys (and CC when required) generation time.

Computation	Mean T_{gen} (s)	Std. Dev. T_{gen} (s)
Statistics	10.8	1.0
Summation/Production	13.3	1.0
Comparisons	12.5	1.0
Linear regression training	20.7	1.4
Linear regression inference	10.9	1.0
Logistic regression training	90.3	2.7
Logistic regression inference	18.1	1.4

4.2.2. T_{enc} : Encryption Time

The HE encryption time is a combined result of data volume, computational complexity, cryptographic parametrization and software optimization. First, the number of values to be encrypted has a direct impact: encrypting larger vectors requires more polynomial operations, increasing the encryption time. Second, the size of the data influence performance, as operations such as training involve encrypting a larger amount of information. Third, the cryptographic parameters, such as the multDepth, affect the computational cost: higher values increase the size of the ciphertexts and the complexity of arithmetic operations required for the encryption process. Finally, the library implementation (OpenFHE or TFHE-rs) and its internal optimizations, also plays a significant role in determining encryption speed.

Table 5 presents the measured mean and standard deviation values of T_{enc} (HE encryption time) for the different algorithms considered in the framework. The results show that operations requiring limited multDepth values (statistics, summation/production and inferences) require relatively short encryption times, around 9–12 s of mean time. In contrast, operations related to model training, particularly logistic regression, exhibit substantially higher encryption times (up to 97 s of mean time) due to the higher required multDepth value as well as to the higher amount of data being encrypted. The consistently low standard deviation values reflect stable and reliable execution times across multiple experimental runs.

Table 5. Encryption time.

Computation	Mean T_{enc} (s)	Std. Dev. T_{enc} (s)
Statistics	9.1	1.0
Summation/Production	9.7	1.0
Comparisons	5.6	1.0
Linear regression training	27.0	1.4
Linear regression inference	9.1	1.0
Logistic regression training	96.9	2.8
Logistic regression inference	12.3	1.0

4.2.3. T_{comp} : Aggregation and Computation Time

Table 6 presents the mean and standard deviation values of T_{comp} (HE computation time) for various algorithms applied over the aggregated datasets. The results show substantial differences across operations, reflecting both their computational complexity and the amount of data processed. Simple tasks, such as statistics and summation/production, exhibit relatively low computation times, ranging from 19 to 25 s of mean time, due to their straightforward arithmetic. Comparisons and linear regression training require more intensive processing, taking 54 and 59 s, respectively, as these computations involve multiple ciphertext multiplications and more complex linear algebra operations. Linear regression inference, with a mean time of 92 s, and logistic regression inference, at 81 s, demonstrate

that the number of encrypted values and operation structure can significantly influence performance. Logistic regression training is the most time-consuming, requiring 930 s, due to both the complexity of the algorithm and the depth of homomorphic operations involved. These results highlight that homomorphic computation time is primarily determined by algorithmic complexity, operation depth, and data volume, with training and inference on complex models representing the main contributors to computational overhead.

Table 6. Computation time over aggregated datasets.

Computation	Mean T_{comp} (s)	Std. Dev. T_{comp} (s)
Statistics	25.3	1.4
Summation/Production	19.2	1.0
Comparisons	54.0	1.7
Linear regression training	59.3	2.2
Linear regression inference	92.3	3.2
Logistic regression training	929.9	7.4
Logistic regression inference	80.8	2.7

Despite the computational overhead introduced by HE, the accuracy of the results obtained with encrypted computations is high, closely matching the results of plaintext operations. This is because HE enables exact or approximate arithmetic on encrypted data without decryption, preserving the mathematical integrity of the underlying algorithms. In the case of CKKS scheme, which operates with approximate arithmetic, the precision loss is influenced by multiple factors, including the scaling factor, multDepth, rescaling operations, and parameter choices such as the modulus size and ring dimension. As a result, parameter tuning is essential to balance accuracy and performance based on the application's requirements.

4.2.4. T_{dec} : Decryption Time

Table 7 reports the mean and standard deviation values of T_{dec} (HE decryption time) inside the trusted entity for the different computations considered in the framework. These results provide insight into the computational overhead introduced by the decryption phase as a function of the complexity of each operation. Basic tasks, such as statistics, summation/production, and comparisons, exhibit low and nearly constant mean decryption times of approximately 2 s, due to the limited amount of data involved in the result. In contrast, computations related to machine learning models, particularly during the inference stage, involve decryption of a much larger number of values, which results in significantly higher mean decryption times of 24 and 27 s for the linear and logistic models, respectively. This behavior is consistent with the results size of these operations.

Table 7. Decryption time.

Computation	Mean T_{dec} (s)	Std. Dev. T_{dec} (s)
Statistics	1.9	1.0
Summation/Production	1.8	1.0
Comparisons	2.2	1.0
Linear regression training	3.1	1.0
Linear regression inference	23.8	1.4
Logistic regression training	6.8	1.0
Logistic regression inference	26.9	1.4

4.3. Communication and Storage Overhead

The communication and storage requirements of the proposed framework were analyzed to better characterize its practical feasibility in multi-source deployments. Table 8 reports the size of the main cryptographic components involved in the evaluated operations, including public keys, evaluation keys, and ciphertexts. All values correspond to the configuration used in the experimental setup. In Table 8, all sizes are given per element, whereas ‘Full workflow size’ measures the total size per full execution of each of the algorithms with two providers. Note that for some operations, EvalSum and EvalRot keys are not required; in those cases, it is indicated with “—”. For measuring the total overhead of a full execution, let us observe that:

1. The public key is sent to providers (P1 and P2) and consumer (C1), so its size should be triplicated for computing the total overhead: $1 \times (P1) + 1 \times (P2) + 1 \times (C1)$
2. Each ciphertext is sent to the consumer, so when computing the total size overhead, each ciphertext should be counted twice (because it is both in the provider and in the consumer). In the case of two providers, this means the ciphertext size is quadruplicated: $1 \times (P1) + 1 \times (P2) + 2 \times (C1)$.
3. The evaluation keys are sent to the consumer, thus only counting their size once per execution.

In summary, the ‘Full workflow size’ corresponds to: $3 \cdot pk + 4 \cdot c + emk + esk + erk$.

Table 8. Communication and storage overhead for each evaluated operation (in MB).

Component	Stat.	Comp.	Linear Inf.	Linear Train	Log. Inf.	Log. Train	SumProd
Public key	6	2260	6	9	8	34	7
EvalMult key	18	117	17	27	24	102	21
EvalSum key	216	—	198	324	288	1190	252
EvalRot key	—	—	—	—	—	102	—
Ciphertext	450	6	400	1260	600	5200	500
Full workflow size	2052	6921	1833	5418	2736	22,296	2294

With this setting the full storage overhead is dominated by both the public key and the ciphertext sizes. Ciphertext size contributes significantly to the total cost, especially considering that its size should be scaled linearly over the number of data providers considered. The same happens with the public key. In the case of the evaluation keys, they only count once, so their final weight in the overall size is less significant.

What one can also see is the difference between the TFHE-rs implementations (used for the ‘Comparisons’ aggregation) and the OpenFHE ones (the rest of the aggregations). Indeed, TFHE-rs presents lightweight ciphertexts but a heavy public key, whereas the opposite happens for OpenFHE. Since both items scale linearly in the count for the full workflow size, there is no advantage in the scheme used in terms of size.

It is important to note that these values are highly dependent on the selected cryptographic parameters (e.g., ring dimension and multDepth configuration) and therefore may vary across deployments. However, the presented results provide a representative indication of the communication and storage footprint associated with the proposed architecture.

4.4. Comparison with Aggregations in the Clear

As a final section of the experimental evaluation, we compare the obtained results for each of the aggregations with the ones obtained when executed without FHE. Considering that FHE, and more concretely the CKKS scheme used, work with approximate arithmetic and noise over the real numbers, it is interesting to see how much error is introduced after

the aggregations, mostly over the more complex algorithms such as the linear and logistic regression trainings.

For this comparison, we executed all the algorithms with the same dataset: a two-column dataset with 100 rows of real numbers. Both the linear and logistic regression trainings use the first column as the input variable (X set in traditional ML literature) and the second as the label (y set). For all the other aggregations, only the first column was used for performing the aggregations. For comparing the FHE results with the real ones, we used the absolute and relative errors. The obtained results are shown in Table 9.

Table 9. Comparison of the results obtained both in executions with fully homomorphic encryption (FHE) and in plaintext (PT).

Computation	Operation	FHE Result	PT Result	Abs. Err.	Rel. Err.
Statistics	Mean	1.04775	1.04775	1.56×10^{-12}	1.49×10^{-12}
	Variance	1.54671	1.54671	2.45×10^{-13}	1.58×10^{-13}
	Std.	1.18405	1.24367	5.96×10^{-2}	4.79×10^{-2}
Summation Production	Summation	6.74999	6.75	9.97×10^{-13}	1.48×10^{-13}
	Production	0.07468	0.07468	3.33×10^{-12}	4.46×10^{-11}
Comparisons	Greater than	22	22	0	0
	Lower than	34	34	0	0
	Equal to	44	44	0	0
Linear regression training	Model weights	25.24370	24.02272	1.22	5.08×10^{-2}
	Bias	−870.72865	−828.61363	4.21×10^1	5.08×10^{-2}
Linear regression inference	Prediction of row 1	12.80097	12.80097	9.25×10^{-11}	7.23×10^{-12}
	Prediction of row 2	113.77578	113.77578	2.13×10^{-13}	1.87×10^{-15}
	Prediction of row 3	63.28838	63.28838	1.05×10^{-10}	1.66×10^{-12}
Logistic regression training	Model weights	0.01083	0.01076	7.86×10^{-5}	7.30×10^{-3}
	Bias	0.01484	0.01484	5.93×10^{-6}	3.9×10^{-4}
Logistic regression inference	Prediction of row 1	0.51574	0.51609	3.523×10^{-4}	6.81×10^{-4}
	Prediction of row 2	0.51587	0.516207	3.37×10^{-4}	6.52×10^{-4}
	Prediction of row 3	0.515999	0.51632	3.22×10^{-4}	6.24×10^{-4}

Since some of the aggregation algorithms contain more than one operation inside, we split them inside Table 9. Hence, for instance, inside the ‘Statistics’ algorithm, three computations are done: mean, variance and standard deviation; since the computations differ, the errors are distinct too. In the case of the inference results, the whole result is 100 predictions (one per row), so we picked the first to study their errors.

As can be seen in Table 9, the algorithms with greater error are the one involving the usage of Chebyshev approximations. Chebyshev approximations were used for:

- Computing the square root in the standard deviation.
- Computing the logistic function in both the logistic regression training and inference.
- Computing the inverse ($1/x$ for some $x \in \mathbb{R}$) in the linear regression training.

As both the absolute and relative errors show, the results obtained inside FHE executions do not differ much from the original ones. Indeed, the complexity of the algorithm does not seem to be such a problem, but the good approximation of the Chebyshev polynomials. Let us recall that Chebyshev polynomials are optimal at approximating functions f inside a given interval (a, b) . However, once $f(x)$ approaches a or b , the interpolation of f starts to oscillate drastically, thus making the results to be more and more inaccurate. In fact, this is what happened in the execution of the linear regression training algorithm: the inverse function $f(x) = 1/x$ was approximated in the interval $(1, 5000)$, but some of

the dataset rows made $x < 1$, thus making the error to considerably increase. This is a real problem in FHE computations, because the consumer cannot know what interval to use in the Chebyshev approximations.

4.5. Discussion

4.5.1. Privacy and Performance Overhead Trade-Off

While the presented results demonstrate the feasibility of the proposed framework for the secure aggregation of multi-source datasets and their confidential use in generic computations, its overall performance remains constrained by the inherent computational overhead introduced by the employed security mechanisms, specially for complex algorithms such as the logistic regression training or inference. However, although the use of HE exhibits lower performance compared to conventional plaintext computation, its adoption offers significant advantages that justify the additional computational overhead. HE enables the direct aggregation and processing of sensitive data in encrypted form, thereby ensuring strong privacy preservation and facilitating collaboration among different organizations. This paradigm removes the necessity of trusting intermediate parties with access to raw data, substantially reducing the risks of data exposure, leakage, and misuse.

Furthermore, it is important to take into account that the current performance limitations of HE are neither intrinsic nor permanent; they can be progressively alleviated through ongoing advances in cryptographic research, improved parameter selection, optimized HE libraries, efficient parallelization techniques, and the incorporation of specialized hardware accelerators. Consequently, despite the moderate efficiency shown in current validation tests, HE constitutes a promising and forward-looking technology whose performance is expected to improve, establishing it as a robust foundation for secure multi-source data aggregation and confidential analytics systems.

As a result, the secure multi-source data aggregation and confidential analytics framework presented in this paper constitutes a foundational paradigm for enabling trusted collaboration across heterogeneous domains while preserving data confidentiality, integrity, and ownership. At its core, the multi-source data aggregation framework supports the aggregation of data originating from distinct organizations, each governed by its own security policies, trust assumptions, and regulatory constraints. Traditional centralized approaches to data aggregation are often incompatible with these settings, as they require the sharing of raw data and complete trust in a single processing entity. In contrast, the proposed framework adopts a decentralized trust model in which data remain under the control of their respective owners and are accessed or processed only under cryptographically and technically enforced guarantees.

The confidential analytics part of the framework is equally fundamental, as it prevents sensitive information from being exposed in plaintext to unauthorized parties, including system operators and infrastructure providers. By doing so, the framework effectively decouples trust in computation from trust in the underlying infrastructure, a property that is increasingly critical in modern cloud-based environments.

4.5.2. Comparative Analysis with State-of-the-Art

To contextualize our framework within the current landscape, Table 10 summarizes key aspects of our approach against some prominent frameworks from the state of the art [17–19,22,23]. This comparison is structured around the three main challenges addressed in this work: (i) the lack of general platforms supporting different aggregation algorithms over data provided by multiple untrusted sources, (ii) the reliance on trusted third parties or shared private keys for homomorphic key management, and (iii) the limited deployment flexibility of hybrid HE and TEE solutions that require part of the computation

to be executed inside the enclave. The proposed framework addresses these challenges through three corresponding contributions: a general framework for secure multi-source data aggregation over previously registered datasets, TEE-based homomorphic key generation and management that removes the need for a trusted third party, and a hybrid key management protocol that keeps the main homomorphic computation outside the enclave. Accordingly, the proposed framework advances the state of the art in the following aspects:

- **Trust Minimization:** Unlike [17–19], which assume a trusted cloud, the proposed TEE-based key generation ensures that no single entity holds both data and private keys, removing reliance on a trusted third party. This contribution directly addresses the key management challenge in multi-source FHE-based aggregation. In [23], authors use a similar TEE-based approach, but the TEE is placed in the same untrusted server that will perform the computations, introducing possible risks in terms of undesired decryptions.
- **Non-TEE Dependent Computations:** In terms of the aggregation executions, only this work and the one in [19] allow to perform the computations only with FHE. All the other works require the computation to be partially run in the TEE, which makes their solution more limited in terms of deployment. Not every data consumer or cloud has a TEE available for performing these executions. By keeping the main aggregation outside the enclave, the proposed framework improves deployment flexibility while preserving confidentiality.
- **Security Model:** While all works use TEEs or MPC, the proposed approach combines secure key generation (via TEEs) with decentralized computation, offering a more robust solution for real-world multi-source scenarios. In addition, security is achieved by hardware-enforced isolation under TEE assumptions, while the aggregation itself remains protected through FHE.
- **Implemented algorithms:** This framework also excels compared to its most similar ones [22,23] in the practical implementation of several aggregations. Indeed, authors in [23] only implemented the linear regression algorithm, whereas the performance evaluation in [22] only uses the mean aggregation. Here, performance results of different easy-to-complex algorithms are shown, assessing the feasibility of implementing and running them in real situations. This supports the contribution of providing a general platform rather than a solution focused on a single operation.

Table 10. Comparative analysis with state-of-the-art approaches.

Reference	Key Management	Sec. Model	Computation	No. of Implemented Algorithms	Formal Security
[17]	Centralized keygen	Trusted Cloud	Inside TEE (no FHE)	1	Informal security analysis
[18]	Centralized keygen	Trusted Cloud	Inside TEE (no FHE)	1	Informal security analysis
[19]	MPC-based	TTP for keygen	FHE	1	Formal security model
[22]	TEE-based	Trust-minimized	FHE-TEE (depending on the operation)	1	Informal security analysis
[23]	TEE-based	TEE in untrusted server	FHE-TEE (depending on the operation)	1	Informal security analysis
This work	TEE-based	Trust-minimized	FHE	7	Informal security analysis

In addition, although there are already other approaches based on the combination of HE and TEE for secure computations [18,53,54], they differ in the scope and system-level perspective. While proposals from the literature focus on combining HE and SGX to enable secure computation, they remain largely single-application or single-workflow oriented. In contrast, the proposed framework targets a broader setting of multi-party data aggregation and confidential analytics across independent data providers, where data from

multiple sources can be securely combined and processed. This introduces additional challenges not addressed in the prior work, such as cross-entity trust, federated data ownership, provenance, and workflow-level security guarantees. In essence, whereas the existing papers focus on how to securely compute on encrypted data, the proposed approach focuses on how to securely orchestrate, fuse, and execute confidential computations across multiple untrusted parties, making it a more general and system-oriented contribution.

This comparison demonstrates that the proposed framework addresses the identified challenges by enabling any data consumer to securely run different multi-source data aggregations over datasets registered by untrusted data providers, without requiring access to raw data or relying on a trusted central aggregator. In addition, it increases the number of aggregation algorithms already available within a common privacy-preserving framework, while combining TEE-based key management with FHE-based computation outside the enclave.

4.5.3. Applicability in Different Domains

Analyzing its application to the reference use case described in Section 3.1, the secure multi-source aggregation framework addresses the need to integrate data coming from different machines and organizations in order to train a more robust predictive model (it could be based on a logistic regression training algorithm, for example). Each organization keeps full control over its own operational data (sensor readings, performance logs, failure reports) and contributes to the learning process only in a protected manner. In this way, diversity in environmental conditions, operational practices, and rare failure modes is incorporated into the model while preserving strict confidentiality boundaries between participants. The final result is a model trained on truly multi-source data, achieving higher accuracy, better generalization, and lower bias.

However, the applicability of the framework spans a wider range of domains. Although the framework is primarily designed around a set of generic operations, it is inherently extensible and can be enriched with additional functionalities, including both domain-agnostic and domain-specific operations. This modular design allows the integration of specialized processing primitives, analytical methods, or algorithms tailored to the requirements of particular application domains. Consequently, the framework provides a flexible foundation that supports general-purpose secure data aggregation while also enabling the incorporation of customized operations that address the semantic and computational characteristics of specific domains. The proposed framework enables secure, privacy-preserving collaboration across diverse domains where sensitive data must be jointly analyzed without exposure. In healthcare, hospitals, research institutions, and pharmaceutical companies can jointly train predictive models on encrypted clinical, genomic, and imaging data—enhancing diagnostics and accelerating research while ensuring compliance with regulations like Health Insurance Portability and Accountability Act and GDPR. In finance, institutions can collaborate on fraud detection, credit risk assessment, and anti-money laundering initiatives without sharing raw transaction data, preserving competitive advantages and customer privacy. In cybersecurity, organizations can securely share threat intelligence—such as indicators of compromise and network telemetry—while protecting internal data, enabling faster detection of emerging threats. Smart cities benefit from integrating data across transportation, energy, and public services to improve safety and efficiency, while industrial and manufacturing sectors can optimize predictive maintenance and supply chain operations without exposing proprietary production data. In environmental monitoring, the framework supports global climate modelling and biodiversity studies by enabling secure aggregation of satellite, sensor, and field data across institutions and countries, ensuring compliance with data sovereignty requirements.

Overall, the secure multi-source aggregation and confidential analytics framework provides a unifying architectural foundation for privacy-preserving collaboration in data-intensive ecosystems. Its domain-agnostic design makes it applicable wherever sensitive data must be jointly analyzed across organizational or jurisdictional boundaries. As data sharing becomes increasingly central to innovation, the proposed framework is likely to play a critical role in shaping the future of secure, ethical, and collaborative computing across domains.

5. Conclusions

This paper presents a framework for secure multi-source data aggregation and confidential analytics, addressing three main challenges identified in existing approaches: (i) the lack of platforms supporting different aggregation algorithms over data provided by multiple untrusted sources, (ii) the reliance on trusted third parties or shared private keys for homomorphic key management, and (iii) the limited deployment flexibility of hybrid HE and TEE solutions that require part of the computation to be executed inside the enclave. To address these challenges, the proposed framework enables data consumers to run different aggregation algorithms over encrypted datasets previously registered by untrusted data providers. To do so, it combines FHE with TEEs through a hybrid key management protocol that uses secure enclaves for homomorphic key generation, distribution, and management without exposing private keys or requiring a trusted third party, and keeps the main homomorphic computation outside the enclave, improving deployment flexibility. The implementation and experimental evaluation on small-scale synthetic datasets, covering key generation, encryption, homomorphic computation, and decryption for operations ranging from simple statistics to logistic regression training, demonstrate the feasibility of the framework and its potential to support privacy-preserving analytics in sensitive domains such as healthcare and finance.

Despite its contributions, the framework has limitations that require further investigation. First of all, specific security mechanisms need to be identified following the security model analysis. Besides, the current evaluation is based on small-scale synthetic datasets, which limit the assessment of scalability and long-term performance under large-scale data loads. Future work should extend the framework with a focus on the aggregation phase to larger datasets, real-world applications, and more complex machine learning models to better characterize its computational demands. Additionally, the framework could be enhanced by supporting additional cryptographic schemes (e.g., lattice-based or approximate FHE) to improve efficiency. Finally, another direction is the integration of hardware accelerators (e.g., GPUs, FPGAs) to reduce the high computational overhead of FHE. By addressing these challenges, the framework can evolve into a more scalable, robust, and widely applicable solution for secure multi-source data aggregation in increasingly complex and privacy-sensitive environments.

Author Contributions: Conceptualization, C.R.; methodology, C.R., J.B.-R., I.S.-A. and I.G.; software, J.B.-R., I.S.-A. and I.G.; validation, C.R. and J.B.-R.; informal analysis, C.R., J.B.-R. and I.S.-A.; investigation, J.B.-R., I.S.-A. and I.G.; resources, J.B.-R.; data curation, J.B.-R., I.S.-A. and I.G.; writing—original draft preparation, C.R.; writing—review and editing, C.R., J.B.-R., I.S.-A. and I.G.; visualization, C.R.; supervision, C.R.; project administration, C.R.; funding acquisition, C.R., J.B.-R., I.S.-A. and I.G. All authors have read and agreed to the published version of the manuscript.

Funding: The authors declare that this study received funding from the European Union's Horizon Europe Framework Programme for Research and Innovation, under the TRUSTEE project (Grant Agreement No. 101070214). The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: During the preparation of this manuscript/study, the authors used GPT-5 mini for the purposes of rewriting text in a more formal and concise manner as well as assisting in describing the contextualization of the research. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: Author Cristina Regueiro, Julen Bernabé-Rodríguez, Iñaki Seco-Aguirre and Idoia Gamiz were employed by the company TECNALIA. The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest

Appendix A. REST API Specification

This appendix summarizes the REST API used by the system components, including the main endpoints, interaction flow, and security assumptions required for reproducibility.

Appendix A.1. HEDF Trusted Entity

The Trusted Entity orchestrates the execution lifecycle, including process initialization, coordination, monitoring, and secure decryption.

POST /register-dataset

Allows for a data provider to register a new dataset inside the Trusted Entity. The data provider must introduce the main metadata information of the dataset to be registered:

```
{
  "field": "healthcare",
  "variables": [
    {
      "name": "var-1",
      "type": "integer",
      "column_index": 0
    },
    {
      "name": "var-2",
      "type": "float",
      "column_index": 1
    }
  ]
}
```

The response returns the assigned dataset ID after the registration process was successful.

POST /register-computation

Before this request, we assume several data providers have registered their datasets in the Trusted Entity. This POST is used for when a data consumer wants to start a new computation. The intention of the data consumer at this point is to introduce the general information of the computation to be done:

```
{
  "field": "healthcare",
  "computation": "linear_regression_training",
  "columns": ["var-1"]
}
```

```
}
```

With this information, the Trusted Entity searches among all the registered datasets for those who fulfill the conditions the consumer has introduced. Once found, the Trusted Entity returns the list of dataset IDs with those conditions. As an example, this list could be D1, D2, D3, D4.

POST /start

With the previous list, the data consumer can decide which of the datasets to use in the final execution. Once the final list is made, this call initializes a new computation by defining the operation, participating datasets and other metadata.

```
{
  "computation": "linear_regression_training",
  "dataset-IDs": ["D1", "D2"]
}
```

The response returns the assigned process ID and initial status (*CREATED*).

GET /{ID}/publicKey & /{ID}/context

Allow for both the data providers and data consumer to obtain the public key and cryptographic context. Similar requests exist for the evaluation keys.

POST /{ID}/encrypted-fl

Invoked by data providers after encryption. The request includes the provider identifier, dataset dimensions, and data endpoint. The Trusted Entity tracks progress and updates the process status.

POST /{ID}/decrypt

Performs secure decryption inside the TEE using the encrypted result and expected dimensions. No private keys or plaintext values are exposed outside this component.

GET /{ID}/result

Retrieves the encrypted output once the computation has finished, including the execution status.

The Trusted Entity implements timeout and retry mechanisms to handle delayed provider responses. All communication is secured via TLS.

Appendix A.2. Data Provider Service

Data Providers encrypt local datasets and expose ciphertexts for further processing.

POST /encrypt

Encrypts a dataset referenced by a URL or local source. The request includes dataset location, size, data type, and process identifier. Encryption is performed using the shared public key and cryptographic context.

GET /getct/{file}

Provides access to ciphertext files stored locally after encryption:

```
GET /getct/proc_001_ct.bin
```

The endpoint only serves encrypted binary data, it does not expose any plaintext information.

PUT /metadata

Updates contextual metadata associated with the dataset when required.

POST /{ID}/delete

Removes encrypted data and cryptographic material associated with a completed process. All encryption is performed locally before data sharing, and communication is authenticated and secured via TLS.

Appendix A.3. Data Consumer Service

The Data Consumer retrieves encrypted datasets and cryptographic material, and executes computations.

POST /execute

Coordinates the execution of algorithms over encrypted data from multiple providers. The request includes provider identifiers, computation type, data structure, dimensions, process ID, and encryption scheme.

```
{
  "DP-IDs": ["P1", "P2"],
  "computation": "linear_regression_training",
  "process_ID": "proc_001",
  "fhe_scheme": "CKKS"
}
```

The response confirms execution and returns the current process status.

GET /getct-result/{file} Allows for the Trusted Entity to obtain the encrypted result once the execution has finished.

PUT /clear-data

Processes or clears temporary input files used during execution.

POST /{ID}/delete

Removes locally stored ciphertexts and intermediate results after completion.

GET /status

Returns system health and availability information.

The Data Consumer operates exclusively on encrypted data. Ciphertexts are retrieved from providers, processed locally, and results are sent to the Trusted Entity for decryption. All communication is authenticated and secured via TLS.

References

1. Duan, Y.; Edwards, J.S.; Dwivedi, Y.K. Artificial intelligence for decision making in the era of Big Data—evolution, challenges and research agenda. *Int. J. Inf. Manag.* **2019**, *48*, 63–71. [CrossRef]
2. Yang, J.; Jiang, J.; Sun, Z.; Chen, J. A large-scale empirical study on improving the fairness of image classification models. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, Vienna, Austria, 16–20 September 2024; pp. 210–222.
3. Schäfer, F.; Gebauer, H.; Gröger, C.; Gassmann, O.; Wortmann, F. Data-driven business and data privacy: Challenges and measures for product-based companies. *Bus. Horiz.* **2023**, *66*, 493–504. [CrossRef]
4. European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation). *Off. J. Eur. Union* **2016**, *L119*, 1–88. Available online: <https://eur-lex.europa.eu/eli/reg/2016/679/oj> (accessed on 15 July 2026).
5. Makker, S.R. Overcoming foggy notions of privacy: How data minimization will enable privacy in the internet of things. *UMKC Law Rev.* **2016**, *85*, 895.
6. Jesus, P.; Baquero, C.; Almeida, P.S. A survey of distributed data aggregation algorithms. *IEEE Commun. Surv. Tutor.* **2014**, *17*, 381–404. [CrossRef]
7. Pourghebleh, B.; Navimipour, N.J. Data aggregation mechanisms in the Internet of things: A systematic review of the literature and recommendations for future research. *J. Netw. Comput. Appl.* **2017**, *97*, 23–34. [CrossRef]
8. Shenoy, D.; Bhat, R.; Krishna Prakasha, K. Exploring privacy mechanisms and metrics in federated learning. *Artif. Intell. Rev.* **2025**, *58*, 223. [CrossRef]
9. Boubakri, M.; Zouari, B. A Survey of RISC-V Secure Enclaves and Trusted Execution Environments. *Electronics* **2025**, *14*, 4171. [CrossRef]
10. Yi, X.; Paulet, R.; Bertino, E. Homomorphic encryption. In *Homomorphic Encryption and Applications*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 27–46.
11. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, Bethesda, MD, USA, 31 May–2 June 2009; pp. 169–178.

12. Chen, H.; Chillotti, I.; Song, Y. Multi-key homomorphic encryption from TFHE. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, 8–12 December 2019*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 446–472.
13. Chen, H.; Dai, W.; Kim, M.; Song, Y. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019*; pp. 395–412.
14. Asharov, G.; Jain, A.; López-Alt, A.; Tromer, E.; Vaikuntanathan, V.; Wichs, D. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, 15–19 April 2012*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 483–501.
15. Marcolla, C.; Sucasas, V.; Manzano, M.; Bassoli, R.; Fitzek, F.H.; Aaraj, N. Survey on fully homomorphic encryption, theory, and applications. *Proc. IEEE* **2022**, *110*, 1572–1609. [\[CrossRef\]](#)
16. Kim, A.; Papadimitriou, A.; Polyakov, Y. Approximate homomorphic encryption with reduced approximation error. In *Proceedings of the Cryptographers' Track at the RSA Conference, Virtual, 1–2 March 2022*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 120–144.
17. Brenna, L.; Singh, I.S.; Johansen, H.D.; Johansen, D. TFHE-rs: A library for safe and secure remote computing using fully homomorphic encryption and trusted execution environments. *Array* **2022**, *13*, 100118. [\[CrossRef\]](#)
18. Drucker, N.; Gueron, S. Achieving trustworthy Homomorphic Encryption by combining it with a Trusted Execution Environment. *J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl.* **2018**, *9*, 86–99.
19. Rahaman, M.; Arya, V.; Orozco, S.M.; Pappachan, P. Secure multi-party computation (SMPC) protocols and privacy. In *Innovations in Modern Cryptography*; IGI Global: Hershey, PA, USA, 2024; pp. 190–214.
20. Mittal, S. Fully homomorphic encryption-based optimal key encryption for privacy preservation in the cloud sector. *J. Inf. Secur. Appl.* **2025**, *91*, 104048. [\[CrossRef\]](#)
21. Fan, S.; Deng, X.; Tian, Z.; Hu, Z.; Chang, L.; Hou, R.; Meng, D.; Zhang, M. TaiYI: A high-performance CKKS accelerator for practical fully homomorphic encryption. *arXiv* **2024**, arXiv:2403.10188
22. Zhao, W.; Qi, S.; Qi, Y. SecDAF: An efficient secure multi-source data analysis framework. *Future Gener. Comput. Syst.* **2026**, *174*, 108020. [\[CrossRef\]](#)
23. Takeshita, J.; McKechney, C.; Pajak, J.; Papadimitriou, A.; Karl, R.; Jung, T. GPS: Integration of Graphene, PALISADE, and SGX for Large-Scale Aggregations of Distributed Data; Paper 2021/1155; Cryptology ePrint Archive: Bellevue, WA, USA, 2021.
24. Rajagopalan, R.; Varshney, P.K. Data Aggregation Techniques in Sensor Networks: A Survey. *IEEE Commun. Surv. Tutor.* **2006**, *8*, 48–63. [\[CrossRef\]](#)
25. Demmler, D.; Schneider, T.; Zohner, M. ABY-A framework for efficient mixed-protocol secure two-party computation. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium 2015, San Diego, CA, USA, 8–11 February 2015*.
26. Damgård, I.; Pastro, V.; Smart, N.; Zakarias, S. Multiparty computation from somewhat homomorphic encryption. In *Proceedings of the Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 643–662.
27. Keller, M.; Pastro, V.; Rotaru, D. Overdrive: Making SPDZ great again. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, 29 April–3 May 2018*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 158–189.
28. Keller, M.; Orsini, E.; Scholl, P. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016*; pp. 830–842.
29. Keller, M. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020*; pp. 1575–1590.
30. Dwork, C.; McSherry, F.; Nissim, K.; Smith, A. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*; Halevi, S., Rabin, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 265–284.
31. Dwork, C.; Roth, A. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* **2014**, *9*, 211–487. [\[CrossRef\]](#)
32. Dwork, C. Differential privacy. In *Encyclopedia of Cryptography, Security and Privacy*; Springer: Berlin/Heidelberg, Germany, 2025; pp. 649–652.
33. Rossman, B. Homomorphism preservation theorems. *J. ACM* **2008**, *55*, 1–53. [\[CrossRef\]](#)
34. Al Badawi, A.; Bates, J.; Bergamaschi, F.; Cousins, D.B.; Erabelli, S.; Genise, N.; Halevi, S.; Hunt, H.; Kim, A.; Lee, Y.; et al. OpenFHE: Open-Source Fully Homomorphic Encryption Library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Los Angeles, CA, USA, 7 November 2022*; WAHC'22; Association for Computing Machinery: New York, NY, USA, 2022; pp. 53–63. [\[CrossRef\]](#)
35. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Hong Kong, China, 3–7 December 2017*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 409–437.

36. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. TFHE: Fast fully homomorphic encryption over the torus. *J. Cryptol.* **2020**, *33*, 34–91.
37. Neda, N.; Ebel, A.; Reynwar, B.; Reagen, B. CiFlow: Dataflow analysis and optimization of key switching for homomorphic encryption. In *Proceedings of the 2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Indianapolis, IN, USA, 5–7 May 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 61–72.
38. Guo, Q.; Nabokov, D.; Suvanto, E.; Johansson, T. Key recovery attacks on approximate homomorphic encryption with Non-Worst-Case noise flooding countermeasures. In *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA, USA, 14–16 August 2024; pp. 7447–7461.
39. Zhao, C.; Zhao, S.; Zhao, M.; Chen, Z.; Gao, C.Z.; Li, H.; Tan, Y.a. Secure multi-party computation: Theory, practice and applications. *Inf. Sci.* **2019**, *476*, 357–372. [\[CrossRef\]](#)
40. Horng, G. An efficient and secure protocol for multi-party key establishment. *Comput. J.* **2001**, *44*, 463–470. [\[CrossRef\]](#)
41. Onishi, R.; Suzuki, T.; Sakai, S.; Yamana, H. Security and Performance-Aware Cloud Computing with Homomorphic Encryption and Trusted Execution Environment. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, Salt Lake City, UT, USA, 14–18 October 2024; pp. 36–42.
42. Jauernig, P.; Sadeghi, A.R.; Stapf, E. Trusted execution environments: Properties, applications, and challenges. *IEEE Secur. Priv.* **2020**, *18*, 56–60. [\[CrossRef\]](#)
43. Li, X.; Zhao, B.; Yang, G.; Xiang, T.; Weng, J.; Deng, R.H. A survey of secure computation using trusted execution environments. *arXiv* **2023**, arXiv:2302.12150.
44. Bouamama, J.; Hedabou, M.; Erradi, M. Cloud Key Management using Trusted Execution Environment. In *Proceedings of the SECUREPT, Virtual*, 6–8 July 2021; pp. 560–567.
45. Valera-Rodriguez, F.J.; Manzanares-Lopez, P.; Cano, M.D. Empirical study of fully homomorphic encryption using microsoft seal. *Appl. Sci.* **2024**, *14*, 4047. [\[CrossRef\]](#)
46. Yuan, M.; Wang, D.; Zhang, F.; Wang, S.; Ji, S.; Ren, Y. An examination of multi-key fully homomorphic encryption and its applications. *Mathematics* **2022**, *10*, 4678. [\[CrossRef\]](#)
47. Goyal, M.; Kar Gangopadhyay, A. Key Extension: Multi-Key FHE Utilizing LWR. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, Hanoi, Vietnam, 25–29 August 2025; pp. 58–71.
48. Lee, J.W.; Kang, H.; Lee, Y.; Choi, W.; Eom, J.; Deryabin, M.; Lee, E.; Lee, J.; Yoo, D.; Kim, Y.S.; et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access* **2022**, *10*, 30039–30054. [\[CrossRef\]](#)
49. Bergerat, L.; Chillotti, I.; Ligier, D.; Orfila, J.B.; Roux-Langlois, A.; Tap, S. New secret keys for enhanced performance in (T) FHE. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, Salt Lake City, UT, USA, 14–18 October 2024; pp. 2547–2561.
50. Li, W.; Xia, Y.; Chen, H. Research on arm trustzone. In *GetMobile: Mobile Computing and Communications*; Association for Computing Machinery: New York, NY, USA, 2019; Volume 22, pp. 17–22.
51. Zheng, W.; Wu, Y.; Wu, X.; Feng, C.; Sui, Y.; Luo, X.; Zhou, Y. A survey of Intel SGX and its applications. *Front. Comput. Sci.* **2021**, *15*, 153808.
52. Sommerhalder, M. Hardware security module. In *Trends in Data Protection and Encryption Technologies*; Springer: Cham, Switzerland, 2023; pp. 83–87.
53. Coppolino, L.; D’Antonio, S.; Formicola, V.; Mazzeo, G.; Romano, L. Vise: Combining intel sgx and homomorphic encryption for cloud industrial control systems. *IEEE Trans. Comput.* **2020**, *70*, 711–724. [\[CrossRef\]](#)
54. Xiao, H.; Zhang, Q.; Pei, Q.; Shi, W. Privacy-preserving neural network inference framework via homomorphic encryption and SGX. In *Proceedings of the 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, Virtual, 7–10 July 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 751–761.
55. Catak, F.O.; Aydin, I.; Elezaj, O.; Yildirim-Yayilgan, S. Practical implementation of privacy preserving clustering methods using a partially homomorphic encryption algorithm. *Electronics* **2020**, *9*, 229. [\[CrossRef\]](#)
56. Shuriya, B.; Kumar, S.V.; Bagyalakshmi, K. Noise-Resilient Homomorphic Encryption: A Framework for Secure Data Processing in Health care Domain. *arXiv* **2024**, arXiv:2412.11474.
57. Rothblum, R. Homomorphic encryption: From private-key to public-key. In *Proceedings of the Theory of Cryptography Conference*, Raleigh, NC, USA, 8–11 November 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 219–234.
58. Joye, M. *Guide to Fully Homomorphic Encryption over the [Discretized] Torus*; Cryptology ePrint Archive: Bellevue, WA, USA, 2021.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.