MOZAIK: A Privacy-Preserving Analytics Platform for IoT Data Using MPC and FHE

Michiel Van Kenhove[1*†], Erik Pohle[2†], Leonard Schild[2†], Martin Zbudila[2†], Merlijn Sebrechts[1], Filip De Turck[1], Bruno Volckaert[1], Aysajan Abidin[2]

[1*]IDLab, Ghent University - imec, Department of Information Technology, Technologiepark-Zwijnaarde 126, Ghent, 9052, Belgium.
[2]COSIC, KU Leuven, Department of Electrical Engineering, Kasteelpark Arenberg 10, Leuven-Heverlee, 3001, Belgium.

*Corresponding author(s). E-mail(s): michiel.vankenhove@ugent.be;
Contributing authors: erik.pohle@esat.kuleuven.be; leonard.schild@esat.kuleuven.be; martin.zbudila@esat.kuleuven.be; merlijn.sebrechts@ugent.be; filip.deturck@ugent.be; bruno.volckaert@ugent.be; aysajan.abidin@esat.kuleuven.be;
†These authors contributed equally to this work.

Abstract

The rapid increase of Internet of Things (IoT) systems across several domains has led to the generation of vast volumes of sensitive data, presenting significant challenges in terms of storage and data analytics. Cloud-assisted IoT solutions offer storage, scalability, and computational resources, but introduce new security and privacy risks that conventional trust-based approaches fail to adequately mitigate. To address these challenges, this paper presents MOZAIK, a novel end-to-end privacy-preserving confidential data storage and distributed processing architecture tailored for IoT-to-cloud scenarios. MOZAIK ensures that data remains encrypted throughout its lifecycle, including during transmission, storage, and processing. This is achieved by employing a cryptographic privacy-enhancing technology known as computing on encrypted data (COED). Two distinct COED techniques are explored, specifically secure multi-party computation (MPC) and fully homomorphic encryption (FHE). The paper includes a comprehensive analysis of the MOZAIK architecture, including a proof-of-concept implementation and performance evaluations. The evaluation results demonstrate the feasibility of the

MOZAIK system and indicate the cost of an end-to-end privacy-preserving system compared to regular plaintext alternatives. All components of the MOZAIK platform are released as open-source software alongside this publication, with the aim of advancing secure and privacy-preserving data processing practices.

Keywords: Internet of Things (IoT), Secure Data Collection, Privacy-Preserving Machine Learning Inference, Secure Multi-Party Computation (MPC), Fully Homomorphic Encryption (FHE), Data Intermediaries

1 Introduction

The pervasive deployment of versatile Internet of Things (IoT) systems in daily life, such as in healthcare [1], smart cities [2, 3], and smart homes [4], continuously generate vast amounts of high-definition images, videos, and sensor data. Massive IoT data demands substantial storage and high-performance computation power, often exceeding the capabilities of typical users or IoT devices [5]. To address this, cloud-assisted IoT leverages the cloud's computational and storage capabilities, providing additional conveniences as a data intermediary. However, this added convenience also introduces new security risks, including data loss and unauthorized access, which present significant challenges in developing secure cloud-assisted IoT systems. Conventional security mechanisms [6] are insufficient to address these risks, and trust-based [7, 8] approaches fall short in providing adequate security, exposing users to privacy risks and companies to liability and reputational risks. Additionally, as data intermediaries, companies must fulfill their obligations under the Data Governance Act (DGA) while also adhering to the General Data Protection Regulation (GDPR) when handling personal data from users inside the European Union. Moreover, ensuring privacy through data anonymization and obfuscation in IoT environments proves impractical, particularly for dynamic operations such as insertion and deletion. This challenge becomes even more pronounced when outsourcing storage or computation to the cloud, where these privacy measures fail to provide provable security [9]. Consequently, the absence of secure and reliable data-sharing and analytics platforms, coupled with the lack of privacy-compliant analytics for personal and proprietary data, hinders the growth of a data-driven market and economy. In turn, this significantly restricts data sharing and technological innovation.

To tackle these challenges, the MOZAIK research project [10] proposes an end-to-end confidential data storage and distributed processing solution for IoT-to-cloud scenarios. This paper presents an in-depth exploration and evaluation of MOZAIK's novel architecture, which ensures that data from IoT devices is encrypted before leaving the user's control, remains encrypted while being securely stored in an IoT-data-focused cloud-based data store, and is processed using privacy-preserving machine learning inference without revealing the plaintext to any single entity in the system. To achieve this, MOZAIK leverages privacy-enhancing technologies (PETs), enabling data owners to securely share sensitive data with data processors without

compromising privacy. While real-world systems often rely on statistical PET techniques such as differential privacy, synthetic data generation, and federated learning, MOZAIK employs cryptographic PET techniques. More specifically, computing on encrypted data (COED), that offers robust cryptographic security guarantees to data owners. MOZAIK integrates two key COED techniques, multi-party computation (MPC) and fully homomorphic encryption (FHE), ensuring end-to-end secure privacy-preserving data processing. To the best of our knowledge, MOZAIK is the first fully end-to-end confidential IoT data storage and privacy-preserving processing platform.

This paper substantially expands upon two of our previously published conference papers [11, 12], providing an in-depth and detailed analysis of the MOZAIK system, presenting significant advancements and novel evaluation results and insights. It includes notable progress and improvements made to the system's architecture, and a proof-of-concept implementation of all its components. We further provide both end-to-end and micro-benchmark evaluations to assess the performance and overhead of the proposed end-to-end privacy-preserving confidential data storage and processing solution. Specifically, our contributions in this paper are as follows:

- Design a novel system architecture to enable scalable and end-to-end secure privacy-preserving computation and analytics on sensitive IoT data.
- Demonstrate the system's feasibility by applying a specific real-life use case as an implementation on top of the system's architecture.
- Evaluate the performance overhead of the designed system in comparison to non-privacy-preserving processing techniques by conducting end-to-end benchmarks and micro-benchmarks of specific system components.
- Provide the complete MOZAIK platform, its implementation, and all its components as open-source software in the following GitHub organization [13].

The remainder of this paper is structured as follows. We begin by positioning our work within the existing literature in Section 2. Then, we introduce key concepts and the required background, including mathematical notation, in Section 3. Section 4 describes the design and implementation of the system's architecture, detailing the functionality of each component and how they communicate with each other. In Section 5, we introduce a specific use case that was implemented as a practical proof of concept to validate MOZAIK's architecture design. The implementation details specific to this use case are described in Section 6. In Section 7, we assess the performance and overhead of the system using both end-to-end benchmarks and micro-benchmarks. Finally, we conclude the paper in Section 8.

2 Related Work

A broad spectrum of platforms has been proposed for privacy-preserving data sharing and processing, particularly in contexts of healthcare, edge devices, and decentralized marketplaces. These works often address only isolated phases of the data lifecycle or rely on centralized trust models, limiting their applicability in fully privacy-preserving systems.

The "MHMD: My Health, My Data" project [14] and the work by Shafagh et al. [15] both present a distributed ledger-based platform focusing on access control features and data management for medical and IoT data, respectively. However, neither system offers mechanisms for protecting the privacy of the underlying data during processing. In contrast, our work explicitly prioritizes data confidentiality, ensuring end-to-end secure data processing while the user maintains control over their data throughout its lifecycle.

A more privacy-focused approach is taken by Agora [16], which leverages functional encryption to achieve input privacy during smart contract execution on blockchain. However, Agora depends on a trusted third party for key generation, and like many such systems, it does not address the problem of secure data ingestion or long-term storage privacy. Niu et al. [17] similarly employ partially homomorphic encryption to enable privacy-preserving data markets, but their reliance on a central authority holding the decryption keys weakens output privacy and introduces a single point of trust. In our work we propose an end-to-end secure data processing platform based on cryptographic techniques, removing the need for a central entity with access to any unencrypted plaintext data.

Several projects use secure multi-party computation to enable collaborative, privacy-preserving processing. The KRAKEN platform [18, 19] adopts MPC to protect data during processing. Likewise, Veeningen et al. [20] present an MPC-based system for secure medical analysis, supplemented by GDPR compliance considerations. However, they rely on intermediate entities, such as hospitals or general practitioners, where data is stored or transmitted in the clear prior to computation, thus leaving the early stages of the data pipeline unprotected. Januszewicz et al. [21] explore secure aggregation in streaming scenarios using functional encryption-like techniques, but their work focuses on a narrow computation model and does not incorporate secure data collection.

A more comprehensive privacy model is discussed by Golob et al. [22], who combine MPC with differential privacy to protect both input and output privacy during processing in a decentralized marketplace. Nevertheless, their platform does not specify secure data collection mechanisms and remains focused on processing in isolation.

With regard to secure computation over encrypted data, a tremendous number of works have recently emerged, offering solutions across various deployment settings [23, 24]. Many MPC-based solutions target the outsourcing setting, where inputs are distributed among computing parties performing machine learning (ML) inference. While often more efficient, such works typically lack integration with data providers. Another line of research considers the client-server scenario, in which a client holding the query and a server owning the ML model engage in secure computation directly, typically via FHE or hybrid FHE-MPC schemes [23, 24]. However, these approaches include the client device as part of the computation cluster, thereby imposing undesirable overhead on the client side. Our work integrates these approaches by leveraging the more efficient outsourced secure processing and tackling the data lifecycle end-to-end, from collection and storage to processing, while minimizing the overhead on the client-edge devices.

Several works focus specifically on distributed computation involving IoT devices and cloud servers. CryptDNN [25] presents a cloud-edge-client architecture based on a hybrid model combining fully homomorphic encryption with searchable symmetric encryption. The authors focus particularly on the efficiency and accuracy of the underlying computation primitives but provide limited analysis of the complete data lifecycle. In contrast, our work emphasizes a scalable, end-to-end secure, privacy-preserving storage and analytics platform for sensitive IoT data.

Similarly, the work of Yang et al. [26] is set in the cloud-edge setting under a hybrid FHE-MPC model combined with differential privacy. Their focus is ML training: an edge server performs shallow training on encrypted data using FHE, after which cloud servers combine models via weighted averaging to form an aggregated model. This aggregated model is then used to perform learning over secret shares to obtain the final target model. While their approach supports privacy-preserving model learning, it does not address secure data collection, storage, or privacy-preserving analysis via model inference. Our approach, by design, does not train models, but focuses on end-to-end secure collection, storage, and analysis of sensitive data using a pre-trained model supplied by a model provider.

PrivESD [27] proposes a privacy-preserving logistic regression framework in a cloud-edge collaborative architecture, targeting encrypted streaming data. It leverages multi-key fully homomorphic encryption to enable computation over data encrypted under different keys, mitigating collusion risks. By distributing computation between edge nodes and the cloud, PrivESD reduces computational complexity and communication overhead while maintaining confidentiality. However, it primarily targets privacy-preserving model training on streaming data. Our work, instead, supports continuous analysis of encrypted sensitive streams by enabling privacy-preserving model inference on individual data points or small micro-batches as they arrive.

In contrast to the above works, our approach addresses privacy and security across the entire data lifecycle, from the moment data is generated and collected, through encrypted transmission and privacy-preserving processing, to secure and verifiable storage. Unlike systems that isolate data protection to a single phase or depend on trusted third parties, our architecture ensures that no stage of the pipeline is exposed to unprotected handling or centralized control. Furthermore, our design effectively minimizes the computational and communication overhead on client-edge devices through the use of lightweight cryptographic schemes. This end-to-end, cryptographically grounded design provides a scalable and secure framework suitable for real-world deployments in untrusted environments, filling a critical gap left by existing solutions.

3 Background

This section provides essential background information and key concepts to help the reader better comprehend the continuation of the paper. To begin, we briefly familiarize the reader with the various notations used throughout this paper. Afterwards, we provide background information on data processing platforms, secure multi-party computation, and fully homomorphic encryption.

3.1 Notation

For this paper we use the notation of $P_i \in \mathcal{P}$ for party i. Let $\ell, n > 1$, we denote the ring of integers modulo ℓ with $\mathbb{Z}_{2^\ell}$ and let $GF(2^n)$ be a field extension of $\mathbb{Z}_2$. We use $x \| y$ to denote concatenation of bits/strings. We denote the uniformly random sampling from a finite set A between parties i and j as $\xleftarrow{\$_{i,j}} A$. Such sampling is implemented using pseudorandom functions (PRFs), as described by Furukawa et al. [28]. We refer to an MPC protocol that performs a functionality f as Π_f.

We briefly recall the syntax and semantics of symmetric and asymmetric *authenticated* encryption. Informally, a nonce-based symmetric authenticated encryption scheme consists of two algorithms:

- $\mathsf{ct}_m^k \leftarrow \mathsf{Enc}_k(m, N, ad)$: Encrypts a message m under the secret key k with associated data ad and nonce N, and produces a ciphertext ct_m^k.
- $m/\perp \leftarrow \mathsf{Dec}_k(\mathsf{ct}_m^k, N, ad)$: Decrypts the ciphertext ct_m^k with the secret key k and associated data ad. This operation only returns the original message m if ct_m^k was encrypted under the same (k, N, ad). Otherwise, it returns failure ($\perp$).

Similarly, an asymmetric authenticated encryption scheme for a private-public key pair $(\mathsf{SK}, \mathsf{PK})$ consists of:

- $\mathsf{ct}_m^{\mathsf{PK}} \leftarrow \mathsf{Enc}_{\mathsf{PK}}(m, ad)$: Encrypts a message m under the public key PK with associated data ad and produces a ciphertext $\mathsf{ct}_m^{\mathsf{PK}}$.
- $m/\perp \leftarrow \mathsf{Dec}_{\mathsf{SK}}(\mathsf{ct}_m^{\mathsf{PK}}, ad)$: Decrypts the ciphertext $\mathsf{ct}_m^{\mathsf{PK}}$ with the private key SK and associated data ad. This operation only returns the original message m if $\mathsf{ct}_m^{\mathsf{PK}}$ was encrypted under the same PK and ad. Otherwise, it returns failure ($\perp$).

In the remaining part of this paper, we will always denote a symmetric secret key as k and public/private keys as PK/SK. We use the notation ct_m^K to denote the ciphertext of message m under the key K, where K is either a public key or a symmetric key.

3.2 Data Processing Platforms

Large-scale data is everywhere, with significant data growth in both commercial and scientific databases due to advancements in data generation and collection technologies. Data is gathered with the expectation that it will hold value, either for its intended purpose or for unforeseen future uses. High-quality data has become even more valuable with the recent advances in artificial intelligence (AI) and large language models (LLMs) [29]. It is clear that simply storing data is insufficient to unlock its full potential value. To extract valuable insights, data must be actively processed. This need has driven the development of data processing platforms [30]. These platforms are designed to efficiently manage, process, store, and analyze data from various sources. They enable us to make data-driven decisions, improve overall operational efficiency, and reduce costs. A data processing platform typically integrates multiple components and leverages various technologies to handle the entire data lifecycle.

To ensure the effectiveness of data processing platforms, they need to be reliable, scalable, and maintainable. The system should maintain its essential performance levels despite of adversity, such as hardware or software faults, or even human mistakes.

As the load on the system increases, whether due to larger data volumes, higher traffic, or greater complexity, there should be reasonable methods to scale the system to maintain the desired performance without significantly increasing costs.

Data processing can be accomplished through batch processing, stream processing, or a combination of both. Batched processing involves the collection and processing of data in large predefined groups. This method is particularly effective for handling tasks that do not require immediate processing results. Batch processing systems operate by accumulating data over a period of time and start processing it all during off-peak times, thereby optimizing the use of computing resources. Stream processing, on the other hand, is used when near real-time results are required. A stream processor operates on each data point shortly after they are ingested in the system, whereas a batch job processes a fixed set of input data. Stream processing systems generally have lower latency to process a single data point compared to batch systems, but typically offer lower throughput than batch systems. A hybrid approach, known as micro-batch processing, tries to find a balance between latency and throughput. Micro-batch processing accumulates a smaller amount of data inputs and groups them in a batch before processing them. This improves the throughput of the data processing platform at the cost of a higher latency.

3.2.1 Obelisk

Obelisk is a cloud-based time series database (TSDB) and storage platform designed and developed by researchers at the IDLab research group of Ghent University-imec. It is specifically designed for scalable IoT data ingestion and querying. Obelisk allows secured data ingestion, storage, streaming and retrieval through HTTPS REST (REpresentational State Transfer) APIs. With regards to data transport, Obelisk ensures server authentication, data confidentiality, and integrity towards its clients (IoT devices, computation servers, end users, etc.) against third-party adversaries through the use of the Transport Layer Security (TLS) protocol, a proven and widely adopted cryptographic protocol designed to secure communication over a computer network. Obelisk has been designed and implemented using an asynchronous, event-based, microservice software architecture, intended to be deployed on a container orchestration platform such as Kubernetes [31]. The microservice-based architecture enables Obelisk to scale individual components under load horizontally, improving reliability and performance [32].

In order to contextualize our proposed software architecture in the remainder of the paper, we include a detailed overview of Obelisk HFS (High-Frequency Streaming). Although Obelisk HFS itself is not a novel contribution of this work, it serves as a foundational component within our system design.

Obelisk HFS is a version of Obelisk specifically designed for high throughput. It has role-based access control (RBAC) built in, where specific permissions or dataset access needs to be explicitly granted to each entity interacting with the platform. Obelisk HFS consists of several microservice-based components, illustrated in the schematic overview of Figure 1. The functionality of each component will be briefly summarized below.

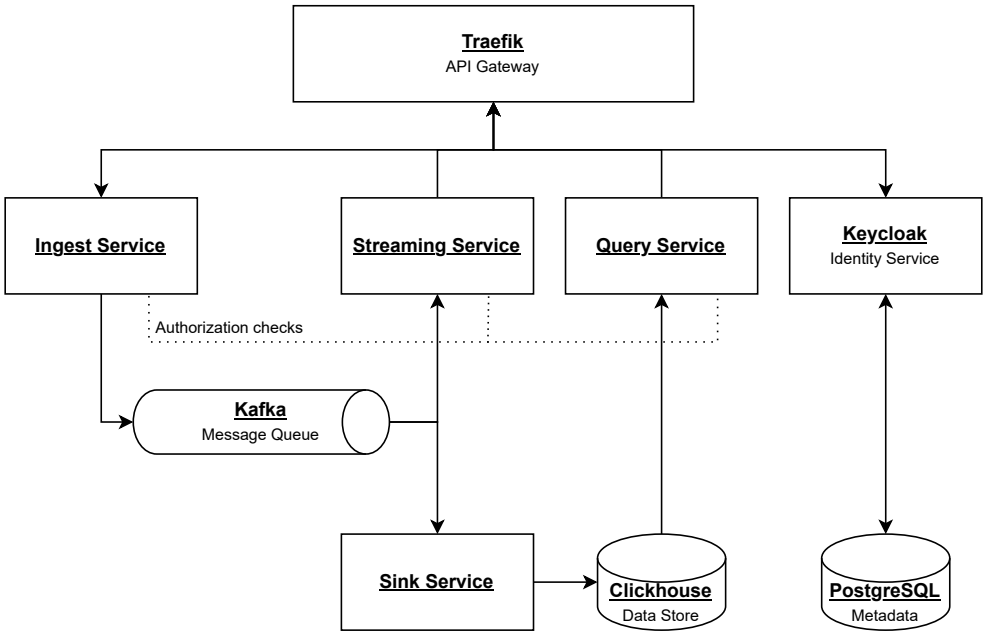


Fig. 1: Architectural overview of Obelisk HFS. The system consists of five distinct microservices: the ingest service, the sink service, the streaming service, the query service, and Keycloak as the identity service. Ingested data is placed on a Kafka message queue topic to be consumed by the sink and streaming service. Data is eventually stored in a Clickhouse data store. The Traefik API gateway ensures requests are directed to the correct service.

API Gateway

All requests made to Obelisk HFS arrive at its API gateway, which delegates each request to the correct service based on the HTTP URI of the request. For example, API requests with a URI of `/ingest` will be directed to Obelisk's ingest service, whereas requests with a URI or `/query` will be directed to the query service. The API gateway uses Traefik [33] that serves as the ingress controller [34, 35] in a Kubernetes cluster.

Ingest Service and Message Queue

The ingest service is responsible for data ingestion. To handle high-frequency data ingestion, only minimal processing is performed on each data point. Instead, the raw data is immediately published to an Apache Kafka [36] message queue topic, and a 204 HTTP response is returned to the data producer to acknowledge that it successfully received the data point. However, this acknowledgment does not guarantee that the data point has been stored in Obelisk's data store, as storage is handled asynchronously by the sink service and may require a significant amount of additional time. Ingesting data is thus an eventual-consistent operation. The rationale behind this design is to minimize the time required to acknowledge receipt of a data point, ensuring that the

data producer is not unnecessarily blocked. Additionally, using a message queue for incoming data enables support for streaming use cases.

Sink Service and Data Store

The sink service consumes data from the Kafka topic containing ingested data and ensures that these data points are reliably stored in the ClickHouse [37] data store. ClickHouse is a high-performance column-oriented database management system designed for online analytical processing. It excels at handling large volumes of data with fast query execution, making it well suited for real-time analytics and time-series data.

Query Service

Once ingested data points are stored in ClickHouse, they are available to query through the query service, which allows the filtering and retrieval of data from datasets. As mentioned above, since data points are initially placed on a message queue before being stored by the sink service, there is a delay between ingestion and when the data becomes queryable.

Streaming Service

Similar to the sink service, the streaming service functions as an additional consumer of the Kafka topic to which ingested data points are published. Rather than persisting the data in ClickHouse, the streaming service selectively forwards each data point that satisfies predefined criteria to subscribed clients using server-sent events [38]. Note that the sink service continues to consume all data points for storage, regardless of whether they are processed by the streaming service, ensuring eventual consistency. When a client creates a stream, they provide specific criteria, such as receiving only data points ingested in a certain dataset or those exceeding a threshold value. This selective forwarding requires real-time processing of each data point by the streaming service, thereby increasing system load relative to traditional poll-based querying. Nevertheless, it facilitates low-latency, real-time data delivery, which is essential for time-sensitive applications.

Identity Service

The identity service in Obelisk is responsible for securely authenticating clients. To achieve this, Obelisk uses Keycloak [39], an open-source identity and access management solution that provides a secure and low-effort authentication system. In addition to authentication, Keycloak supports RBAC and more fine-grained authorization features. However, in Obelisk HFS, Keycloak is used solely for authentication, while authorization is managed by each service individually through configuration files. This approach will be replaced in an upcoming version of Obelisk, where fine-grained authorization will be handled using OpenFGA [40].

3.3 Secure Multi-Party Computation

Secure multi-party computation (MPC) is a cryptographic technique allowing multiple parties to jointly compute a public function on their corresponding data, without

revealing their private inputs to any other parties. MPC protocols are designed to be secure in the presence of a certain number of corrupted parties. The set of corrupted parties denoted by $\mathcal{A}$ is assumed to be controlled by an adversary and may collude. Let $\mathcal{P}$ be the set of the computing parties. If the number of corrupted parties an MPC protocol tolerates is $|\mathcal{A}| < \frac{|\mathcal{P}|}{2}$, we say the MPC protocol is secure in the honest majority setting. Otherwise, if the protocol is able to tolerate $\frac{|\mathcal{P}|}{2} \leq |\mathcal{A}| < |\mathcal{P}|$, we say the MPC protocol is secure in the dishonest majority setting. We further differentiate protocols' security based on adversarial powers. A semi-honest, or passive, adversary is not allowed to deviate from the protocol. Such adversary strictly follows the protocol description but is curious to infer additional information from the data from their view. A malicious, or active, adversary can arbitrarily deviate from the protocol, thus manipulate any values or abort the execution at any time.

Many MPC frameworks work in the offline-online paradigm, where the computation can be split into two phases. The parties first engage in the input-independent preprocessing phase (sometimes called the offline phase), generating correlated randomness needed for the online phase. The parties can then run the online phase efficiently executing only input-dependent operations.

When realizing privacy-preserving machine learning (PPML) inference using MPC, we consider two types of approaches. In a 2-party computation (2PC), the client represents one MPC party with its private data sample to be queried, and the other MPC party represents the ML server holding the trained ML model. Using 2PC, the parties can compute the inference without revealing their inputs to each other. This is often impractical due to the protocol overhead imposed on the client device. Therefore, many recent approaches to PPML inference involve outsourcing the computation to multiple independent third parties. In this approach, both the client and the model owner split their inputs among the predetermined computing parties, who then compute the ML inference using MPC. This ensures secure computation without revealing any information about the private inputs.

There are various cryptographic primitives that can enable MPC. For details regarding generic MPC primitives, we refer the reader to Evans et al. [41]. In MOZAIK, the MPC protocols are based on a secret sharing technique. In secret sharing, each party with a private input splits its secret into multiple parts and distributes these among the computing parties.

3.3.1 Secret Sharing

In this work we will mainly employ replicated secret sharing (see Definition 1) either over the ring $\mathbb{Z}_{2^\ell}$, for $\ell \in \mathbb{N}$, or over the binary extension field $GF(2^n)$, for $n \in \mathbb{N}$.

Replicated Secret Sharing

We use replicated secret sharing (RSS) as a 2-out-of-3 secret-sharing technique described by Furukawa et al. [28]. The secret is split among three parties, such that any two parties can reconstruct the secret. Note that protocols employing RSS are thus secure in the honest majority setting tolerating up to one corrupted party. We use two variants of replicated secret sharing in this paper: arithmetic sharing over the ring $\mathbb{Z}_{2^\ell}$ denoted as $[\![\cdot]\!]^{\mathsf{A}}$ and boolean sharing over the field $GF(2^n)$ denoted as $[\![\cdot]\!]^{\mathsf{B}}$.

Definition 1 (Replicated Secret Sharing)**.** Let $x \in \mathbb{Z}_{2^\ell}$. We denote the arithmetic replicated secret sharing of x as $[\![x]\!]^{\mathsf{A}}$, where $x \equiv x_1 + x_2 + x_3 \mod 2^\ell$. Let $x \in GF(2^n)$. We denote the boolean replicated secret sharing of x as $[\![x]\!]^{\mathsf{B}}$, such that $x = x_1 \oplus x_2 \oplus x_3$. We define the share of party P_i to be $[\![x]\!]_i = \big(x_i, x_{(1+i) \mod 3}\big)$. We will use the notation $[\![x]\!]$ (without subscript) to denote operations on a RSS-shared x performed by every party i.

Multiplication

To obtain RSS shares of the product of two RSS-shared values x and y, we make use of Beaver's trick [42], which works as follows. The parties preprocess shares of a multiplication triple $[\![a]\!], [\![b]\!], [\![c]\!]$, such that $c = a \cdot b$. In the online phase, the parties hold $[\![x]\!], [\![y]\!]$ and wish to obtain $[\![z]\!]$, where $z = x \cdot y$. For this, the parties compute and reveal to all parties $d = x - a$ and $e = y - b$, and then set

$$[\![z]\!] = d[\![b]\!] + e[\![a]\!] + [\![c]\!] + de\,.$$

Conversion between $[\![x]\!]^{\mathsf{A}}$ ***and*** $[\![x]\!]^{\mathsf{B}}$

In the following, we present two protocols to convert a value x shared in arithmetic RSS, i.e., the parties hold $[\![x]\!]^{\mathsf{A}}$, into a boolean RSS, i.e., the parties obtain $[\![x]\!]^{\mathsf{B}}$. Afterwards, the reverse operation, converting from boolean to arithmetic, is presented. We denote with $\mathsf{A2B}$ the arithmetic to boolean, and with $\mathsf{B2A}$ the boolean to arithmetic share conversion. To efficiently convert between the two different types of secret sharing schemes, we employ a conversion protocol based on the ideas by Mohassel and Rindal [43]. The conversion protocols use a Ripple Carry Adder (RCA) circuit, defined in Definition 2 below.

Definition 2 (Ripple Carry Adder)**.** A Ripple Carry Adder is a circuit used for adding binary numbers. Let $x, y \in \mathsf{GF}(2^\ell)$, then the sum $x + y$ over $\mathbb{Z}_{2^\ell}$ can be computed as

$$z_i = x_i \oplus y_i \oplus c_{i-1} \;\; \forall i \in [1, \ell]\,,$$

where z_i denotes the i-th bit of the arithmetic sum and c_i denotes the carry bit, such that

$$c_i = ((x_i \oplus c_{i-1}) \wedge (y_i \oplus c_{i-1})) \oplus c_{i-1} \;\; \forall i \in [1, \ell]\,,$$

given $c_0 = 0$.

In the conversion from arithmetic secret sharing to boolean, detailed in Algorithm 1, the parties locally define the individual arithmetic shares as boolean, add these using the RCA circuit and obtain boolean shares of the secret.

The conversion from boolean secret shares to arithmetic shares is detailed in Algorithm 2. The parties first obtain pairwise uniformly random values, where r_1 is drawn between parties 1 and 2 and r_2 is drawn between parties 2 and 3. The parties then proceed to compute the RCA circuit computing shares of $r_3 = x + r_1 + r_2$. Note that the arithmetic replicated secret shares can then simply be derived by revealing r_3 to parties 1 and 3, and defining the shares to be $(r_3, -r_1, -r_2)$.

Algorithm 1 Arithmetic to Boolean share conversion Π_{A2B}

Input: Party index i, $[\![x]\!]^{\mathsf{A}}$
Output: $[\![x]\!]^{\mathsf{B}}$

1: P_1 sets $[\![x_1]\!]^{\mathsf{B}}_1 := (x_1, 0)$, $[\![x_2]\!]^{\mathsf{B}}_1 := (0, x_2)$ and $[\![x_3]\!]^{\mathsf{B}}_1 := (0, 0)$.
2: P_2 sets $[\![x_1]\!]^{\mathsf{B}}_2 := (0, 0)$, $[\![x_2]\!]^{\mathsf{B}}_2 := (x_2, 0)$ and $[\![x_3]\!]^{\mathsf{B}}_2 := (0, x_3)$.
3: P_3 sets $[\![x_1]\!]^{\mathsf{B}}_3 := (0, x_1)$, $[\![x_2]\!]^{\mathsf{B}}_3 := (0, 0)$ and $[\![x_3]\!]^{\mathsf{B}}_3 := (x_3, 0)$.
4: The parties compute $[\![x]\!]^{\mathsf{B}} := \mathsf{RCA}(\mathsf{RCA}([\![x_1]\!]^{\mathsf{B}}, [\![x_2]\!]^{\mathsf{B}}), [\![x_3]\!]^{\mathsf{B}})$.

Algorithm 2 Boolean to arithmetic share conversion Π_{B2A}

Input: Party index i, $[\![x]\!]^{\mathsf{B}}$
Output: $[\![x]\!]^{\mathsf{A}}$

1: If $i \neq 3$: $r_1 \xleftarrow{\$_{1,2}} \mathbb{Z}_{2^\ell}$ and P_1 sets $[\![r_1]\!]^{\mathsf{B}}_1 := (r_1, 0)$ and P_2 sets $[\![r_1]\!]^{\mathsf{B}}_2 := (0, r_1)$.
2: P_3 sets $[\![r_1]\!]^{\mathsf{B}}_3 := (0, 0)$.
3: If $i \neq 1$: $r_2 \xleftarrow{\$_{2,3}} \mathbb{Z}_{2^\ell}$ and P_2 sets $[\![r_2]\!]^{\mathsf{B}}_2 := (r_2, 0)$ and P_3 sets $[\![r_2]\!]^{\mathsf{B}}_3 := (0, r_2)$.
4: P_1 sets $[\![r_2]\!]^{\mathsf{B}}_1 := (0, 0)$.
5: Compute $[\![r_3]\!]^{\mathsf{B}} = \mathsf{RCA}(\mathsf{RCA}([\![x]\!]^{\mathsf{B}}, [\![r_1]\!]^{\mathsf{B}}), [\![r_2]\!]^{\mathsf{B}})$
6: Reveal r_3 to P_1 and P_3
7: P_1 sets $[\![x]\!]^{\mathsf{A}}_1 := (r_3, -r_1)$, P_2 sets $[\![x]\!]^{\mathsf{A}}_2 := (-r_1, -r_2)$ and P_3 sets $[\![x]\!]^{\mathsf{A}}_3 := (-r_2, r_3)$.

Encoding Fixed-Point Arithmetic

Due to the complexity of ML operations and the high multiplicative depth of the forward pass in a neural network, and to minimize the computation cost, secure ML protocols are designed over rings, more precisely rings of 32 or 64-bit integers leveraging the natural CPU types. For this reason, the state-of-the-art frameworks that focus on secure inference work with fixed-point arithmetic. To encode a decimal real number $\tilde{x} \in \mathbb{R}$ in a fixed-point integer representation $x \in \mathbb{Z}_{2^\ell}$, define $x = \lfloor \tilde{x} \cdot 2^f \rfloor$, where f denotes the fixed-point precision. After a multiplication, the precision is doubled and truncation is needed to restore the original precision [44].

3.4 Fully Homomorphic Encryption

Fully homomorphic encryption schemes are cryptosystems that allow arbitrary computation to be performed over encrypted data by any party, independently of whether such a party is able to decrypt ciphertexts. Historically, several schemes provided homomorphic operations for a single operation, such as RSA [45] with regards to multiplication or the Pailler cryptosystem [46] with regards to addition. In 2009, the first *fully* homomorphic encryption (FHE) scheme was proposed by Gentry based on ideal lattices [47]. The current state-of-the-art consists of two types of FHE schemes, accumulator-based and SIMD-type schemes.

Accumulator-based schemes such as TFHE [48] or FHEW [49] operate on ciphertexts encrypting small plaintext space ranging from 4 to 6 bits, but provide a built-in mechanism to efficiently evaluate arbitrary lookup tables. From a theoretical point of view, it is feasible to work with a larger plaintext space but at a significantly higher

computational cost. In the case of Single-Instruction Multiple-Data (SIMD) FHE schemes like BGV, BFV, or CKKS [50–53], ciphertexts encrypt entire vectors of values and enable efficient, coefficient-wise addition and multiplication over the slots of the vector. Furthermore, the magnitude of entries within any vector is significantly larger than those used in accumulator-based schemes. In general, the latency per operation in SIMD schemes is greater than in accumulator-based schemes, but by considering the cost amortized over the size of the encrypted vector, SIMD schemes may be superior.

In this work, we rely on a SIMD scheme, namely CKKS [53], which is introduced in the subsequent paragraph. Note that, by default, FHE schemes only serve to protect the underlying data of a computation but not the computation itself. More specifically, any user obtaining the (encrypted) result of the computation may be able to derive potentially confidential information about the nature of the computation. This may be detrimental to a model provider aiming to hide certain characteristics of a machine learning model.

3.4.1 The CKKS Scheme

The CKKS scheme was introduced by Cheon et al. [53] and, similarly to other SIMD schemes, encrypts entire vectors. Here, we give a brief overview of operations the scheme provides, but do not discuss the intricacies of the scheme as they are beyond the scope of this article. Let $(\mathsf{PK}, \mathsf{SK})$ denote a public and private key of the CKKS scheme respectively. The cryptosystem provides the following operations.

- $\mathsf{CT} \leftarrow \mathrm{Enc}_{\mathrm{CKKS}}(\vec{m}, \mathsf{PK})$: The procedure $\mathrm{Enc}_{\mathrm{CKKS}}$ takes as input a vector $\vec{m} \in \mathbb{R}^L$ and a public key PK, and produces a ciphertext CT.
- $\vec{m}' \leftarrow \mathrm{Dec}_{\mathrm{CKKS}}(\mathsf{CT}, \mathsf{SK})$: The decryption procedure $\mathrm{Dec}_{\mathrm{CKKS}}$, upon the input of a ciphertext CT and the secret key SK, outputs a vector $\vec{m}' \in \mathbb{R}^L$. We stress that in the case of the CKKS scheme it may be true that

  $$\mathrm{Dec}_{\mathrm{CKKS}}(\mathrm{Enc}_{\mathrm{CKKS}}(\vec{m}, \mathsf{PK}), \mathsf{SK}) = \vec{m} + \vec{\epsilon}.$$

  In other words, the decryption procedure may not output the exact value we encrypted. Often, the error $\vec{\epsilon}$ is compared to the error incurred due to floating point operations if we operated over the plaintext vector $\vec{m}$ without encryption. If managed properly, this error will not affect the correctness of the computation.
- $\mathsf{CT}_+ \leftarrow \mathrm{Add}(\mathsf{CT}_0, \mathsf{CT}_1)$: Given $\mathsf{CT}_i = \mathrm{Enc}_{\mathrm{CKKS}}(\vec{m}_i, \mathsf{PK}), i \in \{0, 1\}$, the procedure Add outputs a new ciphertext CT_+ such that

  $$\mathrm{Dec}_{\mathrm{CKKS}}(\mathsf{CT}_+) = \vec{m}_0 + \vec{m}_1 + \vec{\epsilon}.$$

- $\mathsf{CT}_\odot \leftarrow \mathrm{Mul}(\mathsf{CT}_0, \mathsf{CT}_1)$: Given $\mathsf{CT}_i = \mathrm{Enc}_{\mathrm{CKKS}}(\vec{m}_i, \mathsf{PK}), i \in \{0, 1\}$, the procedure Mul outputs a new ciphertext $\mathsf{CT}_\odot$ such that

  $$\mathrm{Dec}_{\mathrm{CKKS}}(\mathsf{CT}_\odot) = \vec{m}_0 \odot \vec{m}_1 + \vec{\epsilon},$$

  where $\odot$ denotes the Hadamard product.

- $\mathsf{CT}' \leftarrow \mathrm{Rot}(\mathsf{CT}, r), r \in \mathbb{Z}$: Given a ciphertext $\mathsf{CT} = \mathrm{Enc}_{\mathrm{CKKS}}(\vec{m}, \mathsf{PK})$ and a scalar r, the procedure Rot outputs a ciphertext CT' such that $\mathrm{Dec}_{\mathrm{CKKS}}(\mathsf{CT}', \mathsf{SK}) = \vec{m}' + \vec{\epsilon}$ and $\vec{m}'_i = \vec{m}_{i+r \pmod L}$ that is, the procedure outputs a new vector containing the *cyclically shifted* coefficients of $\vec{m}$.

Note that the CKKS scheme relies on additional key material beyond a secret key SK and public key PK, which we suppressed in the notation above. This requirement is not specific to the CKKS scheme but any FHE scheme, though the nature of the key material may vary. Specifically, the operations Mul and Rot require so-called relinearization (multiplication) and automorphism (rotation) keys, respectively. Furthermore, the additional key material is intended to be made public and does not pose a security risk.

3.4.2 Arbitrary Function Evaluation in CKKS

The CKKS scheme is particularly suited to perform polynomial operations. However, many functions commonly used in machine learning cannot be exactly represented using polynomials. Hence, we must rely on polynomial approximations that we construct using polynomial interpolation.

Polynomial interpolation is a fundamental technique in numerical analysis and approximation theory, aimed at constructing a function that passes through a given set of data points. The goal is to approximate arbitrary functions using a set of basis functions that are easier to evaluate. Among the various methods of interpolation, polynomial interpolation is widely used due to its simplicity and efficiency and a common choice in FHE-based settings as polynomials can be easily evaluated. However, the choice of interpolation nodes, that is the set of points at which the function values are known, can significantly impact the accuracy and stability of the interpolation.

We give a general procedure to interpolate a function f using a polynomial. Given a set of $n+1$ data points $(x_i, f_i), i = 0, 1, ..., n$ such that $f_i = f(x_i)$, we aim to find a polynomial $P(x) = \sum_{i=0}^{n} a_i x^i$ such that:

$$P(x_i) = f_i \quad \text{for} \quad i = 0, 1, \ldots, n\,.$$

The coefficients $a_0, a_1, \ldots, a_n$ can be determined by solving the system of equations:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}\,.$$

The system matrix is commonly referred to as a Vandermonde Matrix. We stress that the procedure above is unsuitable for many settings as the matrix may be poorly conditioned.

In cases for which we may pick the set of nodes, it is possible to enforce certain properties of the interpolant. Chebyshev nodes are specific points used in interpolation to minimize the maximum interpolation error over an interval. They are defined as

the roots of the Chebyshev polynomials of the first kind, $T_{n+1}(x)$, which are defined on the interval $[-1, 1]$. The Chebyshev nodes x_k are given by:

$$x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right) \quad \text{for} \quad i = 0, 1, \ldots, n\,.$$

These nodes are particularly useful because they help reduce the Runge phenomenon, characterized by large oscillations at the edges of the interval when using equally spaced points. Furthermore, by leveraging Chebyshev nodes it is possible to bound the absolute interpolation error for any $x \in [-1, 1]$ as

$$|f(x) - P(x)| \leq \frac{1}{2^n(n+1)!} \max_{\bar{x}\in[-1,1]} |f^{(n+1)}(\bar{x})|\,,$$

where $f^{(n+1)}$ denotes the $n+1$-th derivative of f. Finally, we note that extending the procedure to arbitrary intervals instead of $[-1, 1]$ can be achieved through a change of variables, and that Chebyshev nodes often perform well even if f is not differentiable.

4 Platform Design and Implementation Details

In the following section, we detail the general design and implementation details of the MOZAIK platform. We start by outlining the design goals of the system, which serve as the foundation for our design and development process.

4.1 Design Goals

The MOZAIK project aims to design a novel system architecture that improves security, preserves privacy, and enables scalable, end-to-end secure computation and analytics on sensitive IoT data within a cloud-assisted environment. Such a secure, scalable, and privacy-preserving computation platform must achieve the following key design goals.

End-to-End Data Confidentiality. Ensure that IoT data is encrypted at the source before it leaves the user's domain, and that it remains encrypted throughout transmission, storage, and processing, without exposing sensitive information to any external or cloud-based entity.
User Control and Trust. Establish a user-centric trust model that places the user in full control of their data throughout its lifecycle. Users should be required to give their explicit consent to data processing on specific subsets of their data and only by data processors approved by the user.
Correctness. The accuracy of all data processing and analytics operations in the system should match the results that would be obtained if the same computations were performed directly on the plaintext data without any privacy-preserving mechanism in place.
Scalability. Support large-scale IoT deployments with a system that is capable of handling large volumes of data generated by numerous distributed devices, while maintaining performance and low latency suitable for applications that require real-time or near real-time processing.

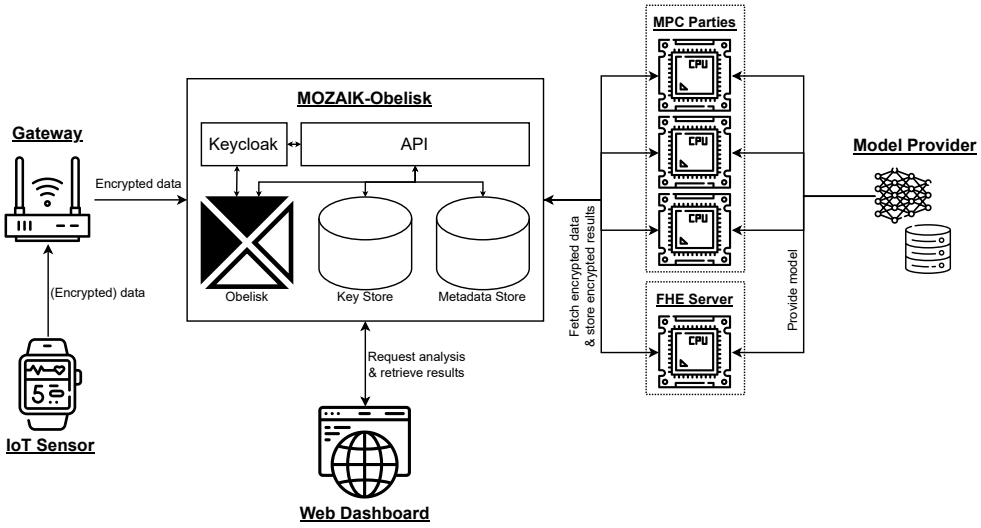


Fig. 2: Schematic overview of the proposed proof-of-concept system architecture of MOZAIK. The data generated by an IoT device is encrypted and securely stored in Obelisk. The FHE server or MPC parties perform computation on the encrypted data on request while ensuring user privacy and store the encrypted result back into Obelisk. The model provider supplies a machine learning model to the computation servers for inference on the user's data. This figure contains resources from Flaticon.com.

Interoperability. Provide a modular and extensible system that can easily integrate with various IoT devices, cloud providers, and data processing entities, promoting broad applicability across different use cases.
Resilience Against Threats. Ensure that the system is robust against a wide range of attack vectors, including unauthorized access, data leakage, malicious cloud service providers, and adversarial attacks on encrypted computation.

4.2 MOZAIK Platform Design Overview

Based on the aforementioned design goals, we propose a proof-of-concept (PoC) system architecture to realize secure and scalable privacy-preserving IoT data processing. The PoC implementation is composed of several core components, depicted in Figure 2. These components are IoT sensors, a gateway, the MOZAIK-Obelisk data storage platform, a web dashboard, data processing servers (MPC parties or an FHE server), and a model provider.

Users are in control of one or more IoT devices that produce or collect personal or sensitive data using sensors. Although not a requirement, a user can also be in control of a gateway, which acts as a single data collection point for multiple IoT devices from that user. The gateway can support other connectivity technologies like Bluetooth to directly communicate with and collect data generated by sensors, allowing support for a wider range of IoT devices or sensor types. Collected data from IoT devices is

encrypted before leaving the user's control. This can occur on the IoT device itself, or on the gateway, depending on the user's preference, the specific data collection scenario or location, or because of IoT device limitations. The encrypted data is then sent to the platform for storage.

All aspects of data storage within the MOZAIK system are unified under what we call MOZAIK-Obelisk. It contains a stateless, high-level application programming interface (API) that provides an abstraction layer to the underlying components of MOZAIK-Obelisk, enhancing modularity and interoperability. The encrypted IoT data itself is stored in Obelisk [32], an independently scalable data platform designed for IoT applications (as discussed in Section 3.2.1), waiting to be processed.

When a computation is triggered, either manually by a user via the web dashboard or automatically when streamed processing is enabled, the API calls a standardized endpoint on the chosen MPC parties or FHE server to request the computation. The computation servers then obtain a machine learning model from a model provider, fetch the computation-specific encrypted data from Obelisk, and perform the requested computation. The encrypted computation results are subsequently stored in Obelisk and made available to the data owner, who is the only one that has the correct keys to decrypt the result.

4.3 Threat Model

The threat model details our assumptions about the adversarial behavior of the different entities in the system. It is an important part to appropriately model security goals in the system and to validate our employed protection techniques.

We consider two types of adversarial behavior. *Semi-honest* (or passive) adversaries follow the protocol honestly, nevertheless try to gain information from the data available in the process. On the other hand, we say an adversary is *malicious* (or active) if it deviates arbitrarily from the protocol in order to violate security.

Our threat model is as follows. We assume that a semi-honest/malicious adversary $\mathcal{A}$ fully controls multiple users, the central storage platform (MOZAIK-Obelisk), and controls the processing entity to the degree described below in Sections 4.3.1 and 4.3.2 (depending on the COED technology used). The goal of $\mathcal{A}$ is to violate any of the following design goals (defined in Section 4.1) involving another user: user control, user trust, privacy, or correctness. We also allow $\mathcal{A}$ to control the model provider *semi-honestly* to support its goal. Specifically, we aim to model a setting where the model provider does not learn any information about the user's sensitive data. At the same time, we want to exclude a variety of possible attacks where the model provider could selectively change the model parameters or architecture to gain insights from user inputs. Since our work is about the processing part, we consider these attack vectors out of scope and therefore only allow semi-honest corruption of the model provider.

Note. The goal of our adversary is to attack the user of the platform, not the model (provider). While we allow the adversary to use other malicious users to gain an advantage, we do not want to claim full privacy for the model parameters. Attacks to (partially) learn the model parameters or training data, such as model inversion or membership inference attacks, are out of scope.

4.3.1 Threat Model for MPC-based Processing

When MPC is used to perform data processing, we assume that in addition to corrupting multiple users, the central storage platform, and the model provider, the adversary also corrupts at most t out of n MPC parties. Concretely, in our PoC architecture, we use three MPC parties and allow up to one semi-honest or malicious corruption.

4.3.2 Threat Model for FHE-based Processing

If FHE is used for data processing, we assume that the adversary additionally has semi-honest access to the FHE computation server.

By construction of an FHE scheme, any malicious adversary is capable of manipulating ciphertexts. Therefore, an adversary can target a specific user by strategically causing computations to fail and may incrementally obtain information about the encrypted data. Thwarting such attacks is possible by relying on zero-knowledge proofs, but may cause a significant overhead in practice. For additional details on this we refer the reader to [54, 55]. Finally, we stress that publishing intermediate or final results of an FHE computation may create security risks if the primitives are used improperly; see [56, 57].

4.4 MOZAIK Component Implementation Details

We now turn to describing each system component separately.

4.4.1 Data Processing Modes

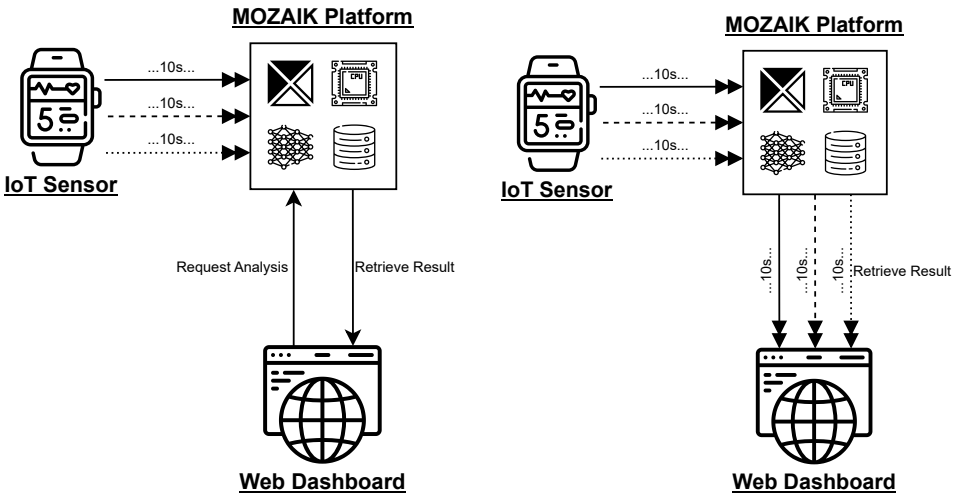
IoT sensor devices typically generate data or capture measurements at a certain frequency. This data is then ingested into the platform at the same rate, with an added timestamp to ensure it can be efficiently searched and indexed. As data is continuously ingested, two distinct data processing modes emerge: ad hoc analysis and streaming. Both modes are detailed in the following and conceptually depicted in Figure 3.

Ad Hoc Analysis

In ad hoc analysis mode, the user has full control over when data is processed. No data point is analyzed without the user's explicit permission or involvement. The user selects which data to process by specifying a time range and grants explicit consent to perform the analysis. Multiple data points can be processed in parallel by grouping them in a batch. The analysis results of the selected data points are then delivered to the user within the expected analysis latency.

Streaming

In streaming mode, the user activates streaming for an IoT sensor device once for a specified amount of time, during which they provide consent for continuous automated data processing without further explicit approval for the specified period. The user can also opt to stop streaming at any point before the specified time ends. Once streaming is activated, the platform groups the ingested data points in micro-batches and analyzes each batch as soon as possible, providing the user with near real-time data



(a) Ad Hoc Analysis Mode (b) Streaming Mode

Fig. 3: Visualization of the ad hoc analysis (a) and streaming (b) mode provided by the MOZAIK platform. In ad hoc analysis mode, ingested data is only processed upon explicit request by the user. In streaming mode, the user explicitly grants the continuous processing of their data for a certain time period. This figure contains resources from Flaticon.com.

insights. It is essential to keep the streaming ingestion rate lower than the end-to-end computation latency, otherwise the system may become overloaded.

4.4.2 Sensor Data Encryption Details

We distinguish two locations where the collected IoT sensor data is encrypted: on the IoT device itself or on a gateway. The choice of location depends on the capabilities and resource constraints of the IoT device, the specific data collection scenario, the availability of a gateway, and the adversarial model.

Encryption on the IoT Sensor Device

If the IoT sensor device is capable of computing symmetric authenticated encryption, then encryption directly on the device is the simplest option.

Since nonce-based schemes require a fresh nonce for every encryption, we require the device to hold a small state st across sensing operations, from which we derive a fresh nonce N[1]. Let d be the sensor data point, the encryption is computed as

$$\mathsf{ct}_d^k := \mathsf{Enc}_k(d, N, \mathsf{ID}_{\mathsf{user}} \| N)\,, \tag{1}$$

[1]For example, this can be a counter which is incremented every time a new nonce is generated. Once the counter overflows the nonce domain, we need to rotate the key k.

where $\mathsf{ID}_{\mathsf{user}}$ is the user ID of the owner of the IoT sensor device. Then, N and ct_d^k are sent to MOZAIK-Obelisk.

Encryption on the Gateway

In case the IoT sensor device is not capable of performing the encryption itself, or if data from multiple sensor devices is aggregated through a gateway, the encryption process can be delegated to the gateway. We do not specify by which method data is transmitted from the sensor device(s) to the gateway, but we assume this transmission occurs securely within the user's control domain. At the gateway, a nonce N is sampled, the corresponding device key k is retrieved or a gateway-specific key is used, and the data point is encrypted following the same procedure as in Equation 1.

4.4.3 Central Data Storage: MOZAIK-Obelisk

One of MOZAIK's goals is to securely store sensitive (encrypted) data, even if only temporarily, in order to perform privacy-preserving computation and provide users with insights into their data. As mentioned above, we utilize Obelisk HFS (see Section 3.2.1) to store the encrypted IoT sensor data. Completing the entire end-to-end data flow, from the IoT sensor device to the computation server(s) and ultimately to reporting the analysis result to the user, involves numerous components that must interact reliably and efficiently with each other. To address this, an overarching API is designed that facilitates seamless communication between all components and entities within the system. It serves as the single external access point for communicating with any component within MOZAIK-Obelisk.

The overarching API is designed to be entirely stateless, meaning that it does not retain any information in memory between requests. Each request is treated completely independently and self-contained, with no reliance on previous interactions. This stateless design eliminates the need for the API to track client state, facilitating horizontal scaling. As a result, performance and reliability can be improved through load balancing and failover across multiple API instances.

The API provides a high-level abstraction of the underlying components of MOZAIK-Obelisk, creating a modular and interoperable system that reduces the dependency on any particular software vendor. For example, if a specific use case demands a different data store, Obelisk can be replaced by another data store without affecting the interface with IoT devices, users, and computation servers. In such case, only the abstraction layer would need to be modified to support the replacement data store.

To enhance security, the API performs additional authentication and authorization checks using Keycloak and JSON Web Token (JWT) [58, 59]. It acts as the initial safeguard against unauthorized data access. For instance, when a user seeks to perform a computation on a specific data point within a dataset, the API verifies whether the user has the necessary permissions to access that dataset. The API will delegate the computation request to the designated computation server(s) to initiate the analysis process if and only if the authorization was successful.

The key store and metadata store are two separate Redis [60] instances. The key store temporarily stores encrypted key shares (described below in Section 4.4.4), while

the metadata store maintains information about each requested analysis, including a unique analysis ID, the Obelisk dataset ID for sensor data, the target dataset ID for storing results, the participating computation servers, and other related metadata. Obelisk, the key store, and the metadata store interact solely with the overarching API and not with each other directly. Additionally, Kubernetes network policies [61] enforce this isolation, permitting communication only with the API (and Keycloak in Obelisk's case), to mitigate lateral movement attacks.

4.4.4 Encryption Key (Share) Management Details

Key Management for FHE

We start by describing key management for FHE, as it is conceptually simpler than that of MPC. When an FHE-based analysis is initiated, the request is sent to a compute engine, which then retrieves the associated ciphertext and the necessary auxiliary key material linked to the user ID from Obelisk. Since the ciphertexts are encrypted under the FHE scheme, data processing can begin immediately, without requiring any additional information apart from the ciphertext and auxiliary key material.

Key (Share) Management for MPC

For MPC-based processing, IoT sensor data is encrypted using a symmetric encryption scheme with the user's or device's secret key k. Before data analysis can begin, the MPC processing parties must receive sufficient information about k in the form of additive secret shares of k.

Since only the user knows this secret key, the key shares k_i are created locally by the user using a $\mathsf{SecretShare}$ function such that $k_1 \oplus k_2 \oplus k_3 = k$. These shares should then be included in the analysis request to the overarching API. However, in the MPC setting, it is crucial that the central data storage does not gain direct access to these key shares, as possessing all of them would allow an adversary who compromises the central data storage to reconstruct the key and breach data privacy.

Instead, we employ the key management protocol of Marquet et al. [12]. Here, the user selects the MPC parties that will perform the computation beforehand and encrypts each key share with the corresponding party's public key, i.e., $\mathsf{ct}_{k_i}^{\mathsf{PK}_i} := \mathsf{Enc}_{\mathsf{PK}_i}(k_i, ad_i)$, where PK_i is the public key of party i and ad_i denotes associated data for this encryption. So, instead of sending the key shares directly to the overarching API, only $\mathsf{ct}_{k_1}^{\mathsf{PK}_1}, \mathsf{ct}_{k_2}^{\mathsf{PK}_2}$ and $\mathsf{ct}_{k_3}^{\mathsf{PK}_3}$ are sent to the API during the analysis request, and these encrypted key shares are temporarily stored in MOZAIK-Obelisk's key store (see Figure 2) until the computation request completes or expires. We require that Enc is an IND-CCA(2) secure public-key encryption scheme, e.g., RSA-OAEP [62] or ECIES [63].

The associated data needed to encrypt each key share is constructed as follows. Let $\mathsf{ID}_{\mathsf{user}}$ be a unique user ID, $\mathsf{ID}_{\mathsf{data}}$ be a set of data point identifiers, e.g., timestamps, type be the analysis type (i.e., the requested analysis algorithm or inference model), and alg be the identifier for the symmetric encryption algorithm that the IoT sensor device used to encrypt the sensor data (e.g., AES-128-GCM), we define the associated

context data as

$$ad_i := 0\text{x}01 \| \mathsf{ID}_{\mathsf{user}} \| \mathsf{PK}_1 \| \mathsf{PK}_2 \| \mathsf{PK}_3 \| \mathsf{ID}_{\mathsf{data}} \| \mathsf{type} \| \mathsf{alg} \| \mathsf{PK}_i \qquad \text{for ad hoc analysis},$$
$$ad_i := 0\text{x}02 \| \mathsf{ID}_{\mathsf{user}} \| \mathsf{PK}_1 \| \mathsf{PK}_2 \| \mathsf{PK}_3 \| t_{\mathsf{begin}} \| t_{\mathsf{end}} \| \mathsf{type} \| \mathsf{alg} \| \mathsf{PK}_i \qquad \text{for streaming},$$

where t_{begin} and t_{end} denote the user-defined start and expiration times of the streaming request.

When an MPC party receives a computation request, it also receives additional metadata, such as the user ID, the data indices that require analysis, and the group of MPC parties that will perform the processing. The MPC party derives ad_i from the received metadata and requests $\mathsf{ct}_{k_i}^{\mathsf{PK}_i}$ from the API, after which it attempts to decrypt the ciphertext key share with SK_i to obtain a key share k_i that will be the input for the data processing step (described in Section 6.3.1). Note that in the case of streaming, the MPC party will also check if the current time is in the user-specified time window for which streaming was allowed, i.e., $t_{\mathsf{begin}} \leq t \leq t_{\mathsf{end}}$. If not, the MPC party refuses the operation. This prevents that a malicious central storage entity attempts to process data with expired key material.

This approach has a central purpose in our design. The associated data binds the encrypted key shares to a processing context. An honest MPC party cannot be coerced into performing a different analysis, using different data points, involving different MPC parties, or switching between streaming and ad hoc computation, other than specified in the processing parameters selected by the user when the analysis request was created. Therefore, the created ciphertexts $\mathsf{ct}_{k_i}^{\mathsf{PK}_i}$ can be viewed as a technical part of the consent the user gives for data processing. The processing context increases the obstacles for abuse and works towards user trust. Additionally, authenticating the encrypted key share detects malicious modification of the ciphertext by an intermediary party, such as the central data storage, thereby preventing decryption oracle attacks.

4.4.5 Data Processing

The MOZAIK platform supports two methods of processing encrypted data: MPC and FHE. The following overview outlines the general interaction and data flow between MOZAIK-Obelisk and the processing servers for both MPC- and FHE-based processing. Specific implementation details, which are highly use-case dependent, are provided in Section 6.3.

MPC

MPC programs interact with MOZAIK-Obelisk through a REST API. An overview of the flow of requests, data, and computation for an ad hoc analysis is illustrated in Figure 4.

Processing. Algorithm 3 describes MOZAIK's MPC-based processing at a high level. To initiate a computation, the MOZAIK-Obelisk's API sends a request to each MPC party containing a unique analysis ID ($\mathsf{ID}_{\mathsf{analysis}}$), the user's unique ID ($\mathsf{ID}_{\mathsf{user}}$), a set of data point identifiers ($\mathsf{ID}_{\mathsf{data}}$) and analysis type ($\mathsf{type}$), specifying the analysis to

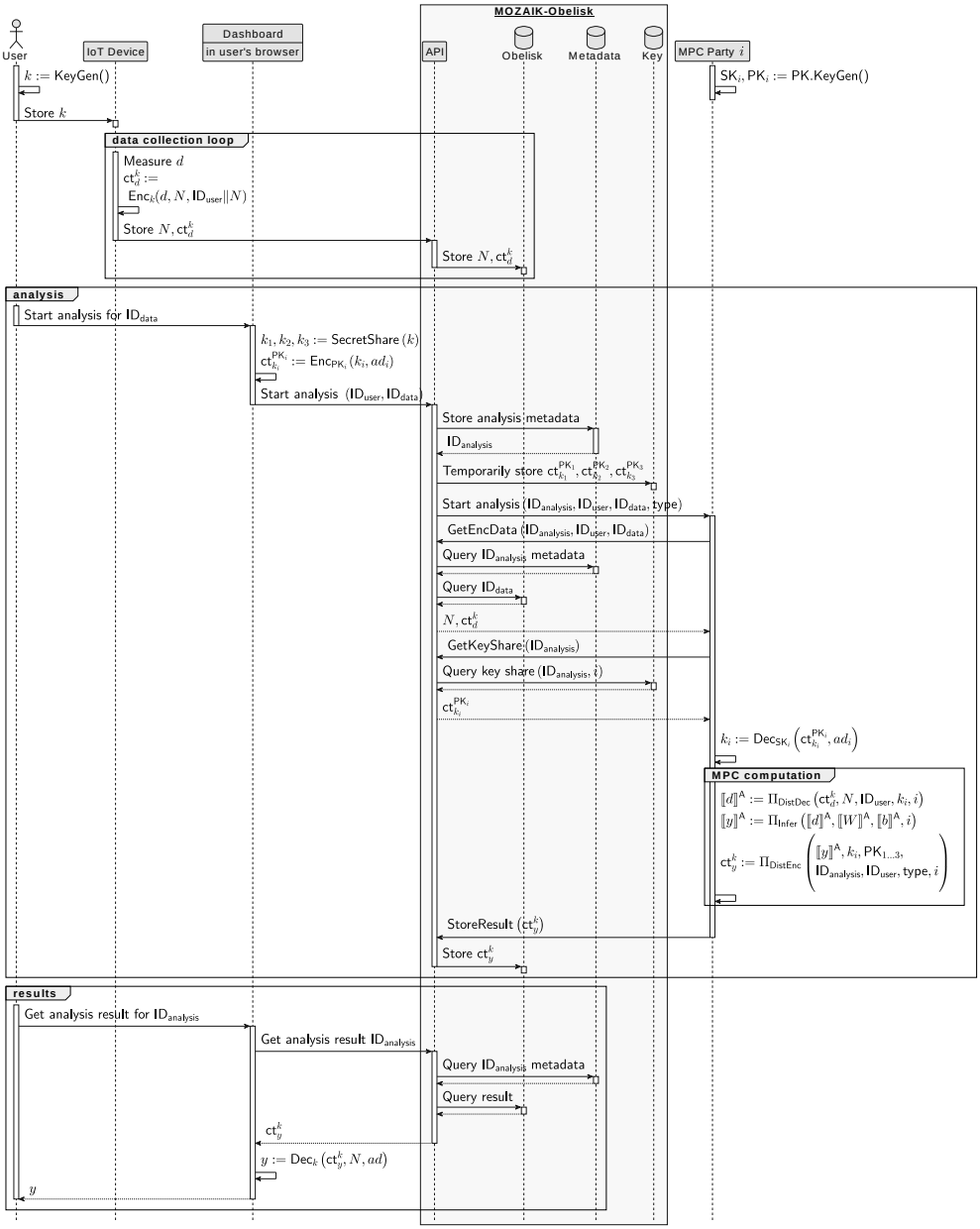


Fig. 4: Sequence diagram illustrating the complete end-to-end flow of an ad hoc MPC analysis. An IoT sensor device collects and encrypts data points, after which they are stored within the user's dataset in Obelisk. The user later initiates an ad hoc analysis on selected data points, triggering the computation flow. Finally, the user obtains the analysis result.

be performed. Each MPC server then adds this request to a queue for sequential processing. For each request in the queue, the parties proceed with the computation as follows. First, the MPC parties fetch the requested encrypted data points from Obelisk and the corresponding encrypted symmetric key shares from the key store. Note that the data points are encrypted under a symmetric key k, whereas each corresponding secret share of k for party i is encrypted with their corresponding public key PK_i. The MPC party locally decrypts the secret share k_i of the symmetric key k using SK_i and then jointly runs the MPC protocol for distributed decryption among the other MPC parties indicated in the request. The obtained shares of each data point are used in the subsequent MPC protocol to compute the desired inference task. Note that we assume the machine learning model to be evaluated had been pre-shared by the model provider prior to this service. As a result, parties store their shares of the model locally and retrieve them for computing the inference. Finally, the resulting share is encrypted via a distributed encryption protocol, and the encrypted result is sent back to MOZAIK-Obelisk where it is stored in Obelisk in the user's result dataset.

Algorithm 3 MPC-based data processing

Input: Party index i, private-public key pair $(\mathsf{SK}_i, \mathsf{PK}_i)$, shares of a machine learning model $\left(\llbracket W \rrbracket^{\mathsf{A}}, \llbracket b \rrbracket^{\mathsf{A}}\right)$

1: queue $+= (\mathsf{ID}_{\mathsf{analysis}}, \mathsf{ID}_{\mathsf{user}}, \mathsf{ID}_{\mathsf{data}}, \mathsf{type}) \xleftarrow[\mathsf{HTTPS}]{} \mathsf{MozaikObelisk}$
2: **for** request **in** queue **do**
3: $\quad N, \mathsf{ct}_d^k \xleftarrow[\mathsf{HTTPS}]{\mathsf{GetEncData}(\mathsf{ID}_{\mathsf{analysis}}, \mathsf{ID}_{\mathsf{user}}, \mathsf{ID}_{\mathsf{data}})} \mathsf{MozaikObelisk}$
4: $\quad \mathsf{ct}_{k_i}^{\mathsf{PK}_i} \xleftarrow[\mathsf{HTTPS}]{\mathsf{GetKeyShare}(\mathsf{ID}_{\mathsf{analysis}})} \mathsf{MozaikObelisk}$
5: $\quad k_i := \mathsf{Dec}_{\mathsf{SK}_i}\left(\mathsf{ct}_{k_i}^{\mathsf{PK}_i}, ad_i\right)$
6: $\quad \llbracket d \rrbracket^{\mathsf{A}} := \Pi_{\mathsf{DistDec}}\left(\mathsf{ct}_d^k, N, \mathsf{ID}_{\mathsf{user}}, k_i, i\right)$ (see Algorithm 5)
7: $\quad \llbracket y \rrbracket^{\mathsf{A}} := \Pi_{\mathsf{Infer}}\left(\llbracket d \rrbracket^{\mathsf{A}}, \llbracket W \rrbracket^{\mathsf{A}}, \llbracket b \rrbracket^{\mathsf{A}}, i\right)$ (see Algorithm 6)
8: $\quad \mathsf{ct}_y^k := \Pi_{\mathsf{DistEnc}}\left(\llbracket y \rrbracket^{\mathsf{A}}, k_i, \mathsf{PK}_{1\ldots3}, \mathsf{ID}_{\mathsf{analysis}}, \mathsf{ID}_{\mathsf{user}}, \mathsf{type}, i\right)$ (see Algorithm 7)
9: $\quad \mathsf{ct}_y^k \xrightarrow[\mathsf{HTTPS}]{\mathsf{StoreResult}} \mathsf{MozaikObelisk}$
10: **end for**

Batched Processing. Batched processing is similar to the processing of a single analysis, except that it operates on all data points from multiple analysis requests simultaneously, improving processing throughput. Note that, for simplicity, Algorithm 3 does not inherently support batched requests with multiple users. In order to support batched computation for our architecture, MOZAIK-Obelisk aggregates computation requests from multiple users in a batch and sends them collectively to the MPC parties. As a result, the MPC parties receive a vectorized version of the previously described variables (i.e., a vector of $\mathsf{ID}_{\mathsf{analysis}}$, $\mathsf{ID}_{\mathsf{user}}$, and $\mathsf{ID}_{\mathsf{data}}$), which is naturally supported by the underlying MPC protocols. The distributed decryption protocol is run on a vector of encrypted data, with a list of symmetric key shares $[k_i]$

as input, such that the i-th key share corresponds to the i-th requested analysis. The output of the distributed decryption is in the form of a vector of matrices, where the rows of each i-th matrix correspond to the selected data points ($\mathsf{ID}_{\mathsf{data}_i}$) from the i-th analysis. For inference, this structure is flattened into a single large matrix, with each row corresponding to an individual data point. The parties then perform ML inference on all the provided data at once. The inference output is then unflattened into a vector of matrices, where the rows of the i-th matrix represent the prediction confidence levels for data points from the i-th analysis. The parties then run the distributed encryption protocol, encrypting each i-th matrix with the corresponding i-th key. Finally, the parties store the ciphertexts in Obelisk.

FHE

At a high level, the FHE-based processing is designed in a way similar to the MPC-based processing and also relies on the REST API for communication with MOZAIK-Obelisk.

FHE-based processing is described in Algorithm 4. Each time a user wishes to perform an analysis, a request is sent to an FHE-Engine by the API which is then added to a task queue. For every request in the queue, the relevant ciphertext is fetched from Obelisk. Then, the required key material needs to be loaded onto the local storage of the FHE-Engine. In practice, the size of the material may be considerable (40 MiB–200 GiB), but is associated with a specific user rather than a particular computation. Hence, we cache the key material locally on the FHE-Engine and only load the material if it is not present yet. Finally, the inference is performed using Algorithm 8 and the result is sent back to MOZAIK-Obelisk.

Algorithm 4 FHE-based data processing

1: $\mathsf{queue} \mathrel{+}= (\mathsf{ID}_{\mathsf{analysis}}, \mathsf{ID}_{\mathsf{user}}, \mathsf{ID}_{\mathsf{data}}) \xleftarrow[\mathsf{HTTPS}]{} \mathsf{MozaikObelisk}$
2: **for** request **in** queue **do**
3: $\quad \mathsf{CT} \xleftarrow[\mathsf{HTTPS}]{\mathsf{GetEncData}(\mathsf{ID}_{\mathsf{analysis}}, \mathsf{ID}_{\mathsf{user}}, \mathsf{ID}_{\mathsf{data}})} \mathsf{MozaikObelisk}$
4: $\quad$ **if** $\mathsf{key_cache}[\mathsf{ID}_{\mathsf{user}}] = \varnothing$ **then**
5: $\quad\quad \mathsf{KM} \xleftarrow[\mathsf{HTTPS}]{\mathsf{GetKeyMaterial}(\mathsf{ID}_{\mathsf{user}})} \mathsf{MozaikObelisk}$
6: $\quad\quad \mathsf{key_cache}[\mathsf{ID}_{\mathsf{user}}] := \mathsf{KM}$
7: $\quad$ **else**
8: $\quad\quad \mathsf{KM} := \mathsf{key_cache}[\mathsf{ID}_{\mathsf{user}}]$
9: $\quad$ **end if**
10: $\quad \mathsf{CT}' := \mathsf{FHEInfer}(\mathsf{CT}, \mathsf{KM})$ (see Algorithm 8)
11: $\quad \mathsf{CT}' \xrightarrow[\mathsf{HTTPS}]{\mathsf{StoreResult}} \mathsf{MozaikObelisk}$
12: **end for**

5 Use Case Description and Requirements

The MOZAIK platform enables the collection and processing of various sensor and IoT data, without making any assumptions about the structure, format, or source of the data. To evaluate the capabilities of our proposed PoC system architecture and demonstrate its practical feasibility, we implement a realistic use case on top of MOZAIK.

5.1 The Heartbeat Analysis Use Case

In our selected use case, users wear heartbeat sensors that continuously measure electrocardiogram (ECG) data. These sensors can be built into a wearable, for example, a smartwatch or other fitness trackers. In this context, the service offered by the platform is continuous monitoring and diagnosis of common heartbeat anomalies that can be extracted from a single lead signal of an ECG. Thus the user's device continuously sends the data points to the platform, which stores them and returns diagnosis predictions to the user. The predictions are computed using a machine learning model with pre-trained parameters. The model is provided by a third party, the so-called model provider.

During a 12-hour measurement period, approximately 21.5 MiB of heartbeat data is collected, while the model parameters for prediction require about 70 KiB. This storage demand may already be substantial for resource-constrained IoT devices. Moreover, such devices often have limited CPU processing capabilities. To address this, we outsource both heartbeat data storage and prediction computation to the cloud. Additionally, the model provider may be reluctant to deploy the prediction model on a user's device to protect their intellectual property and not to reveal business-related model parameters.

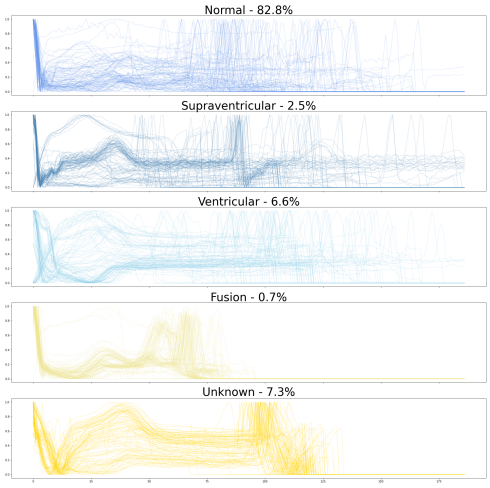
This ECG data is clearly sensitive and personal, requiring the need for a privacy-preserving storage and processing solution. Implementing this use case on top of the MOZAIK platform will keep the user's heartbeat data inaccessible from both the platform and model provider, while at the same time keeping the model parameters private from both the platform and the user. The only information leaked to the platform is the model architecture.

5.2 Dataset and Model

In this work, we do not aim to improve data analysis techniques to enable diagnosis of common heartbeat anomalies but instead use an existing annotated dataset to showcase the processing part of the platform. We utilize the ECG heartbeat classification dataset by Kachuee et al. [64], which unifies the ECG lead II signal of the MIT-BIH Arrhytmia dataset [65] with the PTB Diagnostic ECG database [66]. The raw, single-lead ECG signal is preprocessed to extract all single heartbeats with surrounding signal of 1.2× the length of the median heartbeat duration of a ten-second window. The signal is further normalized. Each heartbeat is classified into one of five classes according to the Association for the Advancement of Medical Instrumentation (AAMI) EC57 standard [67]. Figure 5 shows (a) the five classes and (b) samples

Class	ECG annotation
N	• Normal • Left/Right bundle branch block • Atrial escape • Nodal escape
S	• Atrial premature • Aberrant atrial premature • Nodal premature • Supra-ventricular premature
V	• Premature ventricular contraction • Ventricular escape
F	• Fusion of ventricular and normal
Q	• Paced • Fusion of paced and normal • Unclassifiable

(a) Mapping of AAMI EC57 classes and heartbeat annotations (from [64]).



(b) Visualization of the dataset samples for each class.

Fig. 5: Summary of annotated classes of the heartbeat classification dataset.

of each class. Each sample consists of 187 data points and represents a duration of approximately 1.5 seconds.

In our use-case implementation, the model provider employs a conceptually simpler machine learning model than the model of Kachuee et al. [64]. More specifically, our model consists of four dense layers with 50 intermediary nodes and a dense output layer with five nodes, as shown in Figure 6. The accuracy of the plaintext ECG heartbeat classification is 96%.

5.3 Use Case Requirements

We identify the following requirements, derived from the platform's design goals (see Section 4.1), to guide our implementation of the use case.

Privacy. The sensor data collected from a user remains private throughout all interactions with the platform and its entire lifecycle, including during processing. No sensitive information about the data can be learned by other users, the storage platform, or the model provider.
User Control. The sensor devices, such as smartwatches or fitness trackers, remain under the user's control.
User Trust. The user remains in control even during data processing. The user has to explicitly authorize processing on a selected set of sensor data points by a specific set of processors. Processing of data outside the selected set or involving an unauthorized processor is not possible. Additionally, the user determines the duration of data storage and can retrieve their data at any time.
Correctness of the Diagnosis. The diagnosis prediction delivered to the user is as accurate as if the user had direct access to the same model using their plaintext data.

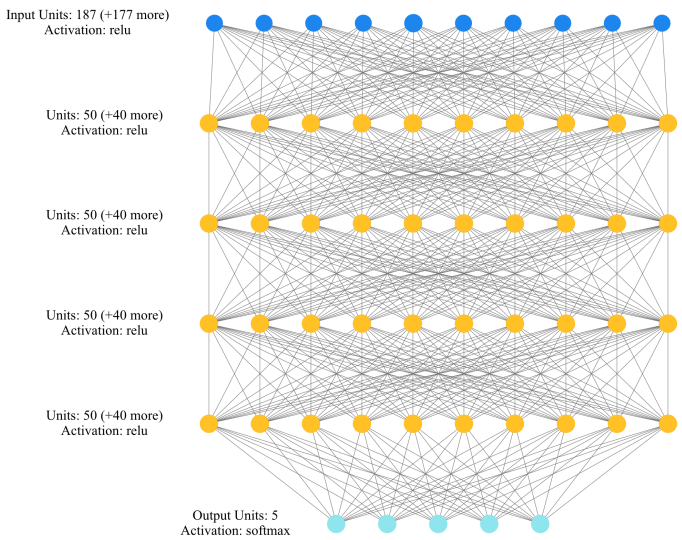


Fig. 6: Visualization of the ECG heartbeat classification model.

From these fundamental requirements, we define the following reasonable performance parameters that the use-case implementation should adhere to.

Data Ingestion Frequency. The user's device sends one sample every 10 seconds.
Analysis Latency. The platform makes an analysis result available to the user within 10 seconds.

As described in Section 5.2, preprocessing the raw ECG signal yields a variable number of 'single' heartbeats depending on the user's heart rate. For example, at a resting heart rate of 80 beats per minute (bpm), approximately 13–14 beats are extracted from a ten-second window, whereas a rate of 150 bpm during exercise yields around 25 beats. However, since arrhythmias are typically characterized by patterns present in multiple atypical heartbeats, sending all beats within the ten-second window offers limited additional value compared to sending a fixed number. We therefore fix the number of beats in the ten-second window to one, allowing the sensor to arbitrarily select one representative beat from each window.

6 Use Case Implementation Details

The following section builds upon the general design and implementation details described in Section 4 and provides use-case-specific implementation details for the use case described in Section 5.

6.1 Data Collection and Encryption at the Source

Our prototype implementation emulates an IoT sensor device by reading a random sample from the ECG dataset described in Section 5.2. This sample is then

encrypted using AES-GCM with AES-128 as the underlying block cipher, as outlined in Section 4.4.2, and then transmitted to MOZAIK-Obelisk via the REST API for storage in Obelisk.

6.2 Key Share Creation and Result Decryption

Creation and encryption of key shares and decryption of analysis results are performed in the user's browser through the web dashboard. To leverage standard implementations of cryptographic primitives provided by the browser, we chose to use RSA-OAEP as public-key encryption scheme and SHA-256 as the hash function, based on the limited options available via the Web Crypto API [68].

6.3 Data Processing

The following extends the general data processing implementation of Section 4.4.5 with the use-case-specific implementation details for both supported processing methods, MPC and FHE.

6.3.1 Data Processing in MPC

Our data processing implementation in MPC comprises two main components, namely distributed decryption and encryption protocols, and a secure inference protocol. These MPC protocols operate on three servers in the honest majority setting, ensuring security against a malicious adversary with the possibility of abort. In the following, we detail each individual MPC component.

Distributed Decryption

This step transforms the public ciphertext ct_d^k into $[\![d]\!]^{\mathsf{A}}$ using shares of the encryption key k_i. The MPC parties receive N, ct_d^k, $\mathsf{ID}_{\mathsf{user}}$, and obtain k_i as described in Sections 4.4.4 and 4.4.5. The parties compute Π_{DistDec} in Algorithm 5 where ct_d^k, N and $\mathsf{ID}_{\mathsf{user}}$ are public inputs, i.e., known to all parties, and the key share k_i is the private input of party P_i. We view $\mathsf{Dec}_{[\![k]\!]^{\mathsf{B}}}(\mathsf{ct}_d^k, N, \mathsf{ID}_{\mathsf{user}} \| N)$ as a circuit where ct_d^k, N, and $\mathsf{ID}_{\mathsf{user}}$ are constants and $[\![k]\!]^{\mathsf{B}} = [\![k_1]\!]^{\mathsf{B}} + [\![k_2]\!]^{\mathsf{B}} + [\![k_3]\!]^{\mathsf{B}}$ is the encryption key that is reconstructed from the input k_i by party P_i but is still secret-shared.

If the concrete symmetric scheme is fixed, then more optimizations are possible to compute Dec. In our use case, we use AES-GCM with AES-128 as block cipher. We implement AES-128-GCM encryption and decryption in three-party replicated secret sharing by extending the MAESTRO framework [69]. The framework implements oblivious AES-128 enciphering which we extend to the encrypt/decrypt operation of GCM [70]. This entails adding support for the GCM field $GF(2^{128})$ during the computation and verification phase. The batched multiplication verification in MAESTRO is implemented for $GF(2^{64})$. We show how to reduce the verification of a multiplication $u \cdot v = w$ with $u, v, w \in GF(2^{128})$ to two inner product checks of values in $GF(2^{64})$. Let $P(X) = X^2 + \beta X + \gamma$ where $\beta, \gamma \in GF(2^{64})$ be an irreducible polynomial. We make use of the isomorphism $\phi : GF(2^{128}) \to (GF(2^{64}))^2$ to represent each element in $GF(2^{128})$ as an element of the degree-2 extension of $GF(2^{64})$ modulo $P(X)$. So, let

$\phi(u) = aX + b$, $\phi(v) = cX + d$, and $\phi(w) = eX + f$, we write the multiplication as

$$(aX + b) \cdot (cX + d) \mod (X^2 + \beta X + \gamma) = (ad + bc + \beta ac)X + (bd + \gamma ac)\,.$$

We can therefore verify the inner product checking two constraints: (1) $[\![a]\!] \cdot [\![d]\!] + [\![b]\!] \cdot [\![c]\!] + \beta \cdot [\![a]\!] \cdot [\![c]\!] = [\![e]\!]$ and (2) $[\![b]\!] \cdot [\![d]\!] + \gamma \cdot [\![a]\!] \cdot [\![c]\!] = [\![f]\!]$. We chose $\beta = 1, \gamma = 2^{61}$.

Note that the output shares $[\![d]\!]^{\mathsf{B}}$ of the distributed decryption are elements in $GF(2^8)$ (the AES field). Since the input to the MPC inference protocol is expected to be in arithmetic replicated secret sharing, the parties run the conversion protocol Π_{B2A} (Algorithm 2) to switch from boolean to arithmetic secret sharing.

Algorithm 5 Distributed decryption Π_{DistDec}

Input: Party index i, ciphertext ct_d^k, nonce N, user ID $\mathsf{ID}_{\mathsf{user}}$, key share k_i
Output: $[\![d]\!]^{\mathsf{A}}$

1: $[\![k]\!]^{\mathsf{B}} := [\![k_1]\!]^{\mathsf{B}} + [\![k_2]\!]^{\mathsf{B}} + [\![k_3]\!]^{\mathsf{B}}$
2: $[\![d]\!]^{\mathsf{B}} := \mathsf{Dec}_{[\![k]\!]^{\mathsf{B}}}(\mathsf{ct}_d^k, N, \mathsf{ID}_{\mathsf{user}} \| N)$
3: $[\![d]\!]^{\mathsf{A}} := \Pi_{\mathsf{B2A}}([\![d]\!]^{\mathsf{B}})$ (see Algorithm 2)
4: **return** $[\![d]\!]^{\mathsf{A}}$

Secure Inference

Our MPC protocol for secure inference is based on arithmetic replicated secret sharing (see Definition 1) and uses fixed-point precision with the parameters $\ell = 64$ and $f = 8$. The precision of 8 bits has been shown to be sufficient for maintaining the model's accuracy, achieving the same 96% as in the plaintext case (see Section 5.2). Furthermore, by setting the precision to 8 bits, we reduce the probability of large errors in a truncation protocol as detailed below.

The protocol for secure inference is described in Algorithm 6. It takes the ML model and the replicated secret shared input sample as input, and evaluates the forward pass of an L-layer deep neural network. Note that addition of secret shares can be done locally. On the contrary, to multiply two secret shares, the parties need to communicate to compute the cross-terms of the product, as described in Section 3.3.1. We implement the ML inference on top of the MP-SPDZ framework [71]. The protocol for generating Beaver triples in the offline phase is based on the preprocessing of Cramer et al. [72]. Activation layers involving non-linear operations are evaluated using binary circuits following the approach of ABY3 [43].

To accommodate the requirements outlined in Section 5.3, we use the MaSTer [44] truncation protocol to minimize the total inference time and maximize the throughput. While MaSTer is currently the most efficient truncation protocol for RSS, it exhibits probabilistic behavior, which may cause the honest parties to abort the computation unnecessarily. However, the probability of abort can be made small by setting the parameters appropriately; for example, choosing $\ell = 64$ and $f = 8$ would make this probability $< 2^{-46}$, see MaSTer [44]. Note that MaSTer achieves its efficiency by

operating under a new adversarial model. Recently, Zbudila et al. [73] demonstrated that the MaSTer protocol is robust in realistic adversarial settings.

Algorithm 6 Secure inference Π_{Infer}

Input: Party index i, input $[\![d]\!]^{\mathsf{A}}$, model parameters $[\![W]\!]^{\mathsf{A}}_j, [\![b]\!]^{\mathsf{A}}_j \ \forall j \in [L]$ where L is the number of layers
Output: $[\![y]\!]^{\mathsf{A}}$
1: $[\![y]\!]^{\mathsf{A}}_{\mathsf{input}} := [\![d]\!]^{\mathsf{A}}$
2: **for** $j \in [L]$ **do**
3: $\quad [\![y]\!]^{\mathsf{A}}_j := \Pi_{\mathsf{Actf}_j}([\![y]\!]^{\mathsf{A}}_{j-1} \cdot [\![W]\!]^{\mathsf{A}}_j + [\![b]\!]^{\mathsf{A}})$
4: **end for**
5: **return** $[\![y]\!]^{\mathsf{A}} := [\![y]\!]^{\mathsf{A}}_L$

Distributed Encryption

In the final step, the parties convert the prediction result $[\![y]\!]^{\mathsf{A}}$ into a ciphertext ct^k_y, y encrypted under k. The parties locally create the associated data ad and nonce N as

$$ad := \mathsf{ID}_{\mathsf{user}} \| \mathsf{PK}_1 \| \mathsf{PK}_2 \| \mathsf{PK}_3 \| \mathsf{ID}_{\mathsf{analysis}} \| \mathsf{type}\,,$$
$$N := \lfloor H(ad) \rfloor_{|N|}\,,$$

where H denotes a collision-resistant hash function and $\lfloor \cdot \rfloor_{|N|}$ is truncation to the nonce length, if necessary. The $\mathsf{ID}_{\mathsf{analysis}}$ is generated by the metadata store and type identifies the type of prediction. By defining ad in this way, we bind the prediction result to a particular request of the user and to the assigned MPC party pool. This excludes mix-and-match types of attacks by MOZAIK-Obelisk targeting the user.

Since the distributed encryption is based on the same techniques as distributed decryption described earlier, we first convert the prediction result $[\![y]\!]^{\mathsf{A}}$ from arithmetic into boolean sharing $[\![y]\!]^{\mathsf{B}}$ using the Π_{A2B} protocol (Algorithm 1) before running the distributed encryption. Protocol Π_{DistEnc} in Algorithm 7 details this.

Algorithm 7 Distributed encryption Π_{DistEnc}

Input: Party index i, inference output $[\![y]\!]^{\mathsf{A}}$, key share k_i, MPC party public keys $\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{PK}_3$, analysis ID $\mathsf{ID}_{\mathsf{analysis}}$, user ID $\mathsf{ID}_{\mathsf{user}}$, prediction type type
Output: ct^k_y
1: $[\![k]\!]^{\mathsf{B}} := [\![k_1]\!]^{\mathsf{B}} + [\![k_2]\!]^{\mathsf{B}} + [\![k_3]\!]^{\mathsf{B}}$
2: $ad := \mathsf{ID}_{\mathsf{user}} \| \mathsf{PK}_1 \| \mathsf{PK}_2 \| \mathsf{PK}_3 \| \mathsf{ID}_{\mathsf{analysis}} \| \mathsf{type}$
3: $N := \lfloor H(ad) \rfloor_{|N|}$
4: $[\![y]\!]^{\mathsf{B}} := \Pi_{\mathsf{A2B}}([\![y]\!]^{\mathsf{A}})$ (see Algorithm 1)
5: $\mathsf{ct}^k_y := \mathsf{Enc}_{[\![k]\!]^{\mathsf{B}}}([\![y]\!]^{\mathsf{B}}, N, ad)$
6: **return** ct^k_y

The MPC parties all obtain ct_y^k after the distributed encryption protocol, which can be safely stored in Obelisk. Note that it might be tempting to have only one MPC party return the result to the database, however, this shortcut comes with a caveat. Since MOZAIK-Obelisk, by design, does not know k, it cannot verify the validity of the ciphertext ct_y^k. Therefore, the single MPC party can arbitrarily modify the ciphertext causing decryption failures only at the moment the user accesses the results. To address this denial of service attack vector, there are two options. To *detect* modification, a single MPC party sends ct_y^k while the remaining only send a cryptographic checksum, e.g., a hash, of ct_y^k. MOZAIK-Obelisk can compare the received value with the received checksums and report the modification to the user and/or the remaining MPC parties. Alternatively, to *prevent* modification, all three MPC parties send ct_y^k. In that case, MOZAIK-Obelisk keeps the version of ct_y^k where at least two parties agree. We choose the second option to ensure a robust processing operation.

6.3.2 Data Processing in FHE

In case of fully homomorphic encryption, we assume that ciphertexts are encrypted on the IoT device using the CKKS encryption scheme (see Section 3.4.1). All cryptographic operations are performed by using the OpenFHE [74] library and, when requested, the decryption is performed in-browser using a WebAssembly port of OpenFHE. Furthermore, to interface with MOZAIK-Obelisk, we set up an HTTP server on the FHE engine that requests input ciphertexts and keys, or sends the inference results back.

In Algorithm 8 we describe the steps necessary to perform the FHE-based inference, leveraging the algorithms described in [75] to compute the matrix-vector product. The inference algorithm makes use of two other subroutines, ChebyshevApproximation and EvalPolynomial. Recall that SIMD-based FHE schemes are tuned towards efficient evaluation of polynomial operations over the encrypted vectors. However, the activation functions typically used in the evaluation of deep learning models are rarely polynomials. It is theoretically possible to compute the activation functions exactly, but in practice it will be more efficient to compute a polynomial approximation and evaluate the latter. The required polynomial is computed in the procedure ChebyshevApproximation and evaluated in EvalPolynomial. Note that the latter simply relies on the Add and Mul functions, and is therefore not described in detail. Due to the use of approximations, the model's accuracy drops marginally from the plaintext baseline to 95% (see Section 5.2).

Algorithm 8 FHE-based inference

Input: CT: the ciphertext encoding the input vector of the inference.
Input: KM: the auxiliary key material of the scheme.
1: Load $(W_i, b_i, F_i)_{i \in [0..k) \cap \mathbb{Z}}$, the weight matrices, biases and activation functions for all k layers.
2: $\mathsf{CT}^0 := \mathsf{CT}$
3: **for** $i = 0$ **to** $i = k - 1$ **do**
4: $\quad \mathsf{CT}^i_{\mathsf{mm}} := \mathsf{FHEMatMul}(W_i, \mathsf{CT}^i)$ (see [75])
5: $\quad \mathsf{CT}^i_{\mathsf{lin}} := \mathrm{Add}(\mathsf{CT}^i_{\mathsf{mm}}, b_i)$
6: $\quad P_i(X) := \mathsf{ChebyshevApproximation}(F_i)$
7: $\quad \mathsf{CT}^{i+1} := \mathsf{EvalPolynomial}(\mathsf{CT}^i, P_i(X))$
8: **end for**
9: **return** CT^k

7 Evaluation Results

This section evaluates the end-to-end privacy-preserving data storage and processing platform introduced in Section 4, focusing on the overhead it introduces compared to processing plaintext data without privacy protections. These evaluations are performed on the implemented use case described in Sections 5 and 6. All evaluation results and scripts are available on GitHub [13].

7.1 Experimental Setup

Our experimental evaluation uses the following hardware.

IoT Sensor and Gateway. The IoT sensor device simulator that ingests samples into the system runs on a virtual machine server, and is supplied with 4 cores of a 2.40 GHz Intel(R) Xeon(R) E5645 CPU and 8 GB of RAM. The gateway runs on a separate virtual machine server with the same CPU type, but supplied with 8 cores and 16 GB of RAM.
MOZAIK-Obelisk. All components encompassed within MOZAIK-Obelisk are deployed on a Kubernetes v1.29.1 cluster, consisting of one controller node and three worker nodes, all with 6 cores of a 2.40 GHz Intel(R) Xeon(R) Silver 4314 CPU and 32 GB of RAM.
MPC Parties. We run the MPC protocol in a three-party setting on three virtual machine servers supplied with 4 cores of a 2.40 GHz Intel(R) Xeon(R) Silver 4314 CPU and 16 GB of RAM.
FHE Server. We run the FHE inference on a virtual machine server supplied with 16 cores of a 2.10 GHz Intel(R) Xeon(R) Silver 4208 CPU and 64 GB of RAM.

The MOZAIK-Obelisk nodes and the MPC parties are interconnected via a 10 Gbit/s network link, achieving realistic data transfer rates of approximately 9.35 Gbit/s as measured with iPerf [76]. The FHE server, on the other hand, is connected to MOZAIK-Obelisk using a 1 Gbit/s network link, achieving approximately 940 Mbit/s.

Experimental Setup for Micro-Benchmarks

We provide micro-benchmark results for the data processing components using the same hardware setup described above. For comparison, we also include computation times for plaintext processing, that is, without applying privacy-preserving COED techniques. These plaintext processing experiments were conducted on a laptop equipped with a 24-core Intel(R) Core(TM) i7-12850HX CPU, 32 GB of RAM, and an RTX A2000 GPU with 8 GB of memory.

7.2 Micro-Benchmarks of ML Inference

In this section, we compare the inference runtimes of our deep neural network (DNN) in MPC and FHE to the corresponding plaintext runtimes when using the TensorFlow library [77, 78]. We present the results in Table 1. The plaintext experiments are run in C++ on GPU using the cppflow [79] library utilizing the TensorFlow C API [80]. The depicted times are averaged over ten experiments, measuring the total inference time, including setup and wrap-up procedures. Note that in the plaintext experiments, this entails data transfers from the CPU to the GPU and back, while in MPC, this involves the networking setup between the computing parties. We further compare the MPC protocol that is secure against a malicious (M) adversary to its semi-honest (SH) counterpart, highlighting the noticeable overhead introduced by the protocol that has stronger security guarantees. Finally, for the MPC experiments, we split the total running time into an offline (preprocessing) phase and an online (input-dependent) phase (see Section 3.3). Since the offline (preprocessing) phase does not depend on the input, the MPC parties could, in theory, perform it during idle times. This would speed up computation when a request arrives, as only the online phase would need to be executed.

The plaintext running times scale sublinearly with the batch size, with running times ranging from the order of microseconds to milliseconds. In contrast, the MPC overhead scales linearly with the number of required multiplications. For smaller batch sizes up to 16, the overhead introduced by MPC is minimal and has little practical impact compared to plaintext inference. However, as the batch size increases, the difference in performance becomes more pronounced. For a testing dataset of 10000 samples, running the inference using a maliciously secure MPC protocol can be up to five orders of magnitude slower than the plaintext version.

In the case of FHE-based inference, we implement single-sample and batched inference for our use case. In SIMD-based schemes, the maximal vector size that can be encrypted usually exceeds the input dimension of the model by several orders of magnitude. Hence, it is possible to perform the inference over batches by storing the concatenation of multiple vectors within the same ciphertext. Then, for a range of batch sizes, the runtime remains constant provided the amount of encoded vectors fits into the ciphertext. However, unlike in MPC, it becomes a requirement that every vector in a batch is associated with the same user, as the keys must match.

Table 1: ML inference times for different batch sizes. The reported times compare the plaintext ML inference time with that of MPC in the semi-honest (SH) setting, MPC in the malicious (M) setting, and of FHE.

Batch size		Time (s)		
		Preprocessing	Input-dependent	Total
1	Plaintext	-	-	0.0006
	MPC (SH)	0.004	0.078	0.082
	MPC (M)	0.066	0.085	0.153
	FHE	-	-	40
16	Plaintext	-	-	0.0009
	MPC (SH)	0.015	0.127	0.145
	MPC (M)	0.395	0.186	0.583
	FHE	-	-	179.2
32	Plaintext	-	-	0.0009
	MPC (SH)	0.031	0.171	0.204
	MPC (M)	0.795	0.301	1.099
	FHE	-	-	179.2
64	Plaintext	-	-	0.0009
	MPC (SH)	0.049	0.292	0.343
	MPC (M)	1.626	0.551	2.180
	FHE	-	-	179.2
128	Plaintext	-	-	0.001
	MPC (SH)	0.099	0.529	0.627
	MPC (M)	3.183	0.965	4.151
	FHE	-	-	179.2
256	Plaintext	-	-	0.0015
	MPC (SH)	0.200	0.990	1.194
	MPC (M)	6.477	1.884	8.366
	FHE	-	-	179.2
512	Plaintext	-	-	0.002
	MPC (SH)	0.368	1.849	2.222
	MPC (M)	13.086	3.901	16.993
	FHE	-	-	358.7
10000	Plaintext	-	-	0.005
	MPC (SH)	6.866	34.463	41.362
	MPC (M)	249.587	71.196	320.858
	FHE	-	-	-

7.3 Communication Overhead

In the following, the sizes of the relevant key material and ciphertexts required to achieve MOZAIK's end-to-end encryption design goal are detailed. These illustrate the communication overhead compared to plaintext data transfers.

For MPC, the symmetric key k that is held by the user's IoT device is a standard AES-128-GCM key with a size of 16 bytes. Each encrypted sample consists of 1496 bytes and has a final size of 1524 bytes due to the added 12 bytes nonce and 16 bytes authentication tag. Each MPC party's RSA public key has a size of 256 bytes. Instead

of secret sharing k, we secret share the key schedule of AES-128 for k, resulting in key shares of 176 bytes that need to be encrypted with RSA-OAEP. These three ciphertext key shares have a size of 256 bytes each, and each of these is intended for one of the three MPC parties. They are temporarily stored in MOZAIK-Obelisk's key store. The ciphertext of the analysis result, produced by performing the distributed encryption protocol, has a size of 56 bytes.

For FHE, there is a larger set of keys. Beyond the secret and public key, measuring 29 MiB and 71 MiB, respectively, the CKKS scheme requires additional keys to enable the homomorphic operations, totaling 26 GiB. We stress again that these additional keys can be publicly shared similarly to the public key. Finally, the input and output sample ciphertext have an identical size of 57 MiB.

7.4 Performance of Ad Hoc Analysis

To evaluate the performance of the proposed platform, we begin by assessing the computation latency of the ad hoc analysis processing mode by measuring the duration from the moment an analysis request is submitted by the user to the point at which the computation result is made available on the web dashboard. We define this as the end-to-end processing latency.

For each batch size, we construct an analysis request containing the corresponding number of data points for that batch size. Once the request is submitted, we wait until the computation results are stored in Obelisk, at which point they also become accessible to the user via the web dashboard. To calculate the end-to-end processing latency for each batch size, we use the timestamps of request submission and result storage that are recorded for each analysis in the metadata store. We repeat this process ten times for each batch size to compute the mean end-to-end processing latencies, which are detailed in Table 2 for both MPC-based and FHE-based processing methods.

Note that FHE-based computation shows constant latency across batch sizes 1 through 8, and a higher constant latency for batch sizes 16 through 256. This is due to certain optimizations we include that only apply for smaller batch-sizes. Briefly, for smaller batch sizes, we can include multiple copies of the same sample in a single ciphertext. By exploiting this redundancy, we can decrease the required number of rotations and multiplications.

These results indicate that in ad hoc analysis mode, the MPC-based processing method in a malicious setting supports batch sizes of up to 240 while maintaining an end-to-end processing latency just below the ten-second requirement specified in Section 5.3. This highlights the cost of an end-to-end confidential data storage and processing solution, since the plaintext counterpart of a computation with batch size 240 would require less than 0.0015 seconds (see Table 1). The FHE-based processing method, on the other hand, does not meet the specified use-case requirements.

7.5 Performance of Streaming

In streaming mode, on the other hand, data is continuously ingested at a certain ingest rate (in samples/s) and grouped into batches, with processing starting as soon

Table 2: Mean end-to-end (E2E) processing latency for varying batch sizes and the two evaluated COED methods in ad hoc analysis.

Batch Size	E2E Processing Latency (s) MPC-based	FHE-based
1	1.059	41.017
2	1.075	
4	1.106	
8	1.354	
16	1.931	180.217
32	2.784	
48	3.459	
64	4.020	
80	4.555	
96	4.962	
112	5.598	
128	6.245	
144	6.651	
160	7.149	
176	7.953	
192	8.479	
208	8.881	
224	9.280	
240	9.921	
256	10.507	

as a batch is filled. Since ingesting data at a certain ingest rate requires time as well, we distinguish between two end-to-end processing latencies: the maximum end-to-end processing latency and the minimum end-to-end processing latency. The maximum end-to-end processing latency includes both the time required to fill a batch and the end-to-end processing latency itself, representing the longest possible delay an ingested data point may experience until it has been processed. The minimum end-to-end processing latency does not include the time required to fill a batch and reflects only the end-to-end processing latency. The data point that arrives first in a batch will experience the maximum end-to-end processing latency, whereas a data point that arrives last in a batch experiences the minimum end-to-end processing latency. Under normal load conditions, we expect that the minimum end-to-end processing latency for a batch of a certain size remains relatively constant. To evaluate the performance, we gradually increase the ingest rate for each batch size until we observe a significant rise in minimum end-to-end processing latency. The maximum ingest rate for a given batch size is the highest ingest rate at which the latency remains consistent with the latency observed at lower ingest rates. This method is visualized in Figure 7 for a batch size of 192, which has a minimal end-to-end processing latency below ten seconds under normal load, as an example. Here, the minimum end-to-end processing latency remains relatively stable up until an ingest rate of 18.8 samples/s. Ingesting more data beyond the maximum ingest rate would build up a queue with an ever-increasing length, eventually resulting in a failing system. Note that we only evaluate the streaming

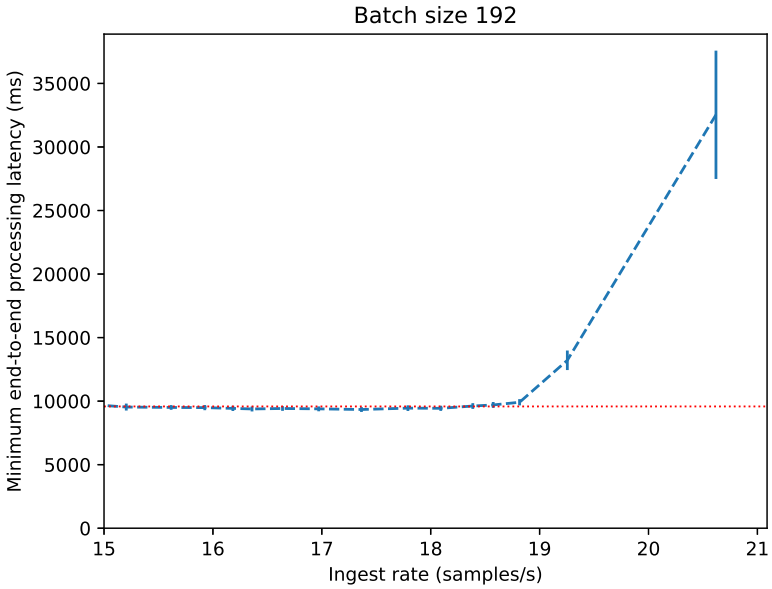


Fig. 7: Minimum end-to-end processing latency for batch size 192 under a gradually increasing ingest rate. It is evident that the latency begins to rise significantly beyond an ingest rate of 18.8 samples/s, identifying this as the maximum sustainable ingest rate for this batch size. The red dotted horizontal line represents the mean latency of all processing runs up to and including this threshold.

mode for the MPC-based processing method, since the FHE-based processing method, with an end-to-end processing latency of already 41.017 seconds for batch sizes one through eight, does not meet the ten-second analysis result requirement specified in Section 5.3.

Table 3 presents the evaluation results for MPC-based processing in streaming mode across varying batch sizes. Each row reports the maximum achieved ingest rate in samples/s for a certain batch size, the maximum and minimum end-to-end processing latencies at that ingest rate, and the number of supported data streams within a ten-second window at that ingest rate. The number of supported streams is derived from the ingest rate and reflects how many IoT sensor devices can be connected to the platform, assuming that each device ingests one data point every ten seconds, and that all connected devices ingest data points uniformly in the ten-second window. To meet the requirements outlined in Section 5.3, the maximum end-to-end processing latency must remain below ten seconds. Otherwise, some ingested data points will need to wait longer than ten seconds to retrieve their analysis result. The platform can therefore support an ingest rate of up to 13.841 samples/s at a batch size of 64, or 138 IoT sensor devices uniformly sending one data point each in a ten-second window. Here, the first data point in each batch will experience an average maximum wait time of 8.757 seconds, while the final data point in the batch will wait at least 4.205 seconds on average to retrieve their analysis result.

Table 3: Maximum ingest rate in samples/s achieved for streamed processing at certain batch sizes. The maximum and minimum end-to-end (E2E) processing latency at a given batch size respectively represents the longest and shortest time a data point might need to wait before its analysis result becomes available. The supported data streams reflect the maximum amount of IoT sensor devices that can be connected to the platform when performing streamed processing at a certain batch size.

	Streamed processing			
Batch size	Samples/s	Max. E2E latency (s)	Min. E2E latency (s)	Data streams
1	0.974	1.120	1.120	9
2	1.948	1.623	1.110	19
4	3.124	2.181	1.221	31
8	4.146	3.086	1.398	41
16	7.075	4.076	1.956	70
32	9.885	6.278	3.142	98
48	12.139	7.431	3.559	121
64	13.841	8.757	4.205	138
80	14.799	10.152	4.814	147
96	16.109	11.099	5.202	161
112	17.341	12.534	6.133	173
128	17.690	14.158	6.979	176
144	17.618	15.742	7.625	176
160	17.947	16.727	7.868	179
176	18.215	18.457	8.849	182
192	18.816	20.062	9.911	188
208	19.051	21.591	10.725	190
224	19.095	23.534	11.856	190

8 Conclusion

As the volume of sensitive IoT data that require storage and analysis continues to grow, the need for an end-to-end secure and privacy-preserving storage and analytics system arose. This paper presents MOZAIK, an end-to-end confidential data storage and distributed processing solution for IoT-to-cloud scenarios. This work covers a detailed examination of the proposed architecture's design and implementation details. To validate MOZAIK's practical feasibility and better evaluate its capabilities, a realistic heartbeat anomaly detection use case was implemented. The platform ensures end-to-end data confidentiality by encrypting the data at the source before it leaves the user's control using a symmetric encryption key held by the user. The encrypted data is stored in Obelisk, a scalable, cloud-based time series database tailored for IoT data ingestion and querying. In order to provide secure data analytics and insights, two distinct COED techniques, MPC and FHE, are explored. These techniques allow privacy-preserving data processing, including ML inference, without revealing sensitive information about the data to the processors. The resulting encrypted analytics are stored back in Obelisk.

Our evaluation results demonstrate the feasibility of MOZAIK while also highlighting the substantial overhead associated with delivering end-to-end privacy-preserving data storage and analytics compared to conventional plaintext systems that offer

no security or privacy. Despite the higher computational and performance costs, the strong privacy and security guarantees provided by MOZAIK make it a worthwhile trade-off in environments where data confidentiality is critical.

Trade-Offs Between MPC and FHE

This work explores and implements both MPC and FHE COED techniques. Based on our experience, we briefly discuss the advantages and shortcomings of both technologies. On the one hand, MPC-based techniques offer sizable advantages in terms of performance. Even in the malicious setting, MPC-based inference outperforms FHE by a factor of 20 to 260 depending on the batch size. Furthermore, FHE-based techniques require additional key material on a per-user basis, which are of considerable size. The latter also impacts the computational efficiency: for any batch size, approximately 20 seconds are spent loading keys into RAM from disk. On the other hand, the deployment and management of the FHE pipeline is much simpler. Network traffic is only required at the very beginning and end of any computation, and no coordination between FHE servers is ever required. Due to the same independence, adding new FHE servers for horizontal scaling is straightforward, whereas scaling an MPC setup often involves complex coordination among parties, who must agree on legal frameworks and trust relationships. Furthermore, while the performance of MPC is bounded by the network properties, the performance of FHE is largely bound to the hardware and therefore can be improved by upgrading the latter. Considering the growing body of dedicated hardware acceleration for FHE [81, 82], it follows that the performance gap may be reduced or even closed in the future.

In general, our experiments suggest that employing FHE is preferable in scenarios where the network between MPC parties has low throughput or when the assumptions underlying the MPC security model are not believed to hold. However, when performance, particularly in terms of time or memory efficiency, is critical, MPC-based approaches will typically be more suitable. Moreover, MPC protocols offering malicious security are especially well suited for critical applications that require verifiable computation, an area where FHE currently falls short, unless combined with additional costly cryptographic mechanisms such as zero-knowledge proofs. Finally, a hybrid approach is also worth considering, where MPC parties simultaneously serve as FHE servers, allowing low-priority tasks to be delegated to the FHE components.

Declarations

Funding. This work was supported by the Flemish Government through the *Fonds Wetenschappelijk Onderzoek* (FWO), also known as the Research Foundation Flanders, under the SBO project MOZAIK with grant number S003321N. The project began on 1 May 2021 and concluded on 30 April 2025.

Competing Interests. Filip De Turck, a listed author, is a member of the Editorial Advisory Board (EAB) of this Journal. The authors declare that they have no other competing interests.

Data, Materials, and Code Availability. The complete MOZAIK architecture, design, implementation, and evaluation scripts and results are released as open-source software, bundled in the https://github.com/MOZAIK-SBO/ GitHub organization.

Author Contributions. *Michiel Van Kenhove*, *Erik Pohle*, *Leonard Schild*, and *Martin Zbudila* contributed equally to this work. Together, they designed, developed, and implemented the MOZAIK architecture over the past four years, and have all contributed substantially to the writing of this paper. *Merlijn Sebrechts*, *Filip De Turck*, *Bruno Volckaert*, and *Aysajan Abidin* supervised the project, providing valuable feedback and critical revisions to both the work and the manuscript. In addition, *Aysajan Abidin* served as the project lead for MOZAIK, offering essential guidance that was instrumental to the project's successful completion. All authors reviewed and approved the final manuscript.

References

[1] Selvaraj, S., Sundaravaradhan, S.: Challenges and opportunities in IoT healthcare systems: a systematic review. SN Applied Sciences **2**(1), 139 (2019) https://doi.org/10.1007/s42452-019-1925-y

[2] Latre, S., Leroux, P., Coenen, T., Braem, B., Ballon, P., Demeester, P.: City of things: An integrated and multi-technology testbed for IoT smart city experiments. In: 2016 IEEE International Smart Cities Conference (ISC2), pp. 1–8 (2016). https://doi.org/10.1109/ISC2.2016.7580875

[3] Santos, J., Vanhove, T., Sebrechts, M., Dupont, T., Kerckhove, W., Braem, B., Seghbroeck, G.V., Wauters, T., Leroux, P., Latre, S., Volckaert, B., Turck, F.D.: City of Things: Enabling Resource Provisioning in Smart Cities. IEEE Communications Magazine **56**(7), 177–183 (2018) https://doi.org/10.1109/MCOM.2018.1701322

[4] Ansari, A.M., Nazir, M., Mustafa, K.: Smart Homes App Vulnerabilities, Threats, and Solutions: A Systematic Literature Review. Journal of Network and Systems Management **32**(2), 29 (2024) https://doi.org/10.1007/s10922-024-09803-1

[5] Bautista, O.G., Akkaya, K.: MPC-as-a-Service: A Customizable Management Protocol for Running Multi-party Computation on IoT Devices. In: NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, pp. 1–6 (2023). https://doi.org/10.1109/NOMS56928.2023.10154349

[6] Mosenia, A., Jha, N.K.: A Comprehensive Study of Security of Internet-of-Things. IEEE Transactions on Emerging Topics in Computing **5**(4), 586–602 (2017) https://doi.org/10.1109/TETC.2016.2606384

[7] Najib, W., Sulistyo, S., Widyawan: Trust Based Security Model in IoT Ecosystem. In: 2022 6th International Conference on Information Technology, Information

Systems and Electrical Engineering (ICITISEE), pp. 195–199 (2022). https://doi.org/10.1109/ICITISEE57756.2022.10057930

[8] Hosseini Shirvani, M., Masdari, M.: A survey study on trust-based security in Internet of Things: Challenges and issues. Internet of Things **21**, 100640 (2023) https://doi.org/10.1016/j.iot.2022.100640

[9] Wang, W., Xu, P., Yang, L.T.: Secure Data Collection, Storage and Access in Cloud-Assisted IoT. IEEE Cloud Computing **5**(4), 77–88 (2018) https://doi.org/10.1109/MCC.2018.111122026

[10] MOZAIK: Scalable and Secure Data Sharing. https://www.esat.kuleuven.be/cosic/projects/mozaik/ Accessed 2025-05-10

[11] Abidin, A., Marquet, E., Moeyersons, J., Limani, X., Pohle, E., Van Kenhove, M., Marquez-Barja, J.M., Slamnik-Kriještorac, N., Volckaert, B.: MOZAIK: An End-to-End Secure Data Sharing Platform. In: Proceedings of the Second ACM Data Economy Workshop. DEC '23, pp. 34–40. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3600046.3600052

[12] Marquet, E., Moeyersons, J., Pohle, E., Van Kenhove, M., Abidin, A., Volckaert, B.: Secure Key Management for Multi-Party Computation in MOZAIK. In: 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 133–140 (2023). https://doi.org/10.1109/EuroSPW59978.2023.00020

[13] Van Kenhove, M., Pohle, E., Schild, L., Zbudila, M., Sebrechts, M., De Turck, F., Volckaert, B., Abidin, A.: GitHub MOZAIK. https://github.com/MOZAIK-SBO

[14] Morley-Fletcher, E.: MHMD: My Health, My data. In: Ioannidis, Y.E., Stoyanovich, J., Orsi, G. (eds.) Proceedings of the Workshops of the EDBT/ICDT 2017 Joint Conference (EDBT/ICDT 2017), Venice, Italy, March 21-24, 2017. CEUR Workshop Proceedings, vol. 1810 (2017). https://ceur-ws.org/Vol-1810/EuroPro_paper_04.pdf

[15] Shafagh, H., Burkhalter, L., Hithnawi, A., Duquennoy, S.: Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In: Thuraisingham, B., Karame, G., Stavrou, A. (eds.) Proceedings of the 9th Cloud Computing Security Workshop, CCSW@CCS 2017, Dallas, TX, USA, November 3, 2017, pp. 45–50. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3140649.3140656

[16] Koutsos, V., Papadopoulos, D., Chatzopoulos, D., Tarkoma, S., Hui, P.: Agora: A Privacy-Aware Data Marketplace. IEEE Transactions on Dependable and Secure Computing **19**(6), 3728–3740 (2022) https://doi.org/10.1109/TDSC.2021.3105099

[17] Niu, C., Zheng, Z., Wu, F., Gao, X., Chen, G.: Achieving Data Truthfulness and Privacy Preservation in Data Markets. IEEE Transactions on Knowledge and

Data Engineering **31**(1), 105–119 (2019) https://doi.org/10.1109/TKDE.2018.2822727

[18] Gabrielli, S., Krenn, S., Pellegrino, D., Baún, J.C.P., Berganza, P.P., Ramacher, S., Vandevelde, W.: KRAKEN: A Secure, Trusted, Regulatory-Compliant, and Privacy-Preserving Data Sharing Platform. In: Curry, E., Scerri, S., Tuikka, T. (eds.) Data Spaces - Design, Deployment and Future Directions, pp. 107–130. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-98636-0_6

[19] Koch, K., Krenn, S., Pellegrino, D., Ramacher, S.: Privacy-Preserving Analytics for Data Markets Using MPC. In: Friedewald, M., Schiffner, S., Krenn, S. (eds.) Privacy and Identity Management: 15th IFIP WG 9.2, 9.6/11.7, 11.6/SIG 9.2.2 International Summer School, Maribor, Slovenia, September 21-23, 2020, Revised Selected Papers. IFIP Advances in Information and Communication Technology, vol. 619, pp. 226–246. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-72465-8_13

[20] Veeningen, M., Chatterjea, S., Horváth, A.Z., Spindler, G., Boersma, E., Spek, P.J., Galiën, O., Gutteling, J., Kraaij, W., Veugen, T.: Enabling Analytics on Sensitive Medical Data with Secure Multi-Party Computation. In: Ugon, A., Karlsson, D., Klein, G.O., Moen, A. (eds.) Building Continents of Knowledge in Oceans of Data: The Future of Co-Created eHealth - Proceedings of MIE 2018, Medical Informatics Europe, Gothenburg, Sweden, April 24-26, 2018. Studies in Health Technology and Informatics, vol. 247, pp. 76–80 (2018). https://doi.org/10.3233/978-1-61499-852-5-76

[21] Januszewicz, A., Gutierrez, D.M., Koirala, N., Zhao, J., Takeshita, J., Lee, J., Jung, T.: PPSA: polynomial private stream aggregation for time-series data analysis. IACR Cryptol. ePrint Arch., 1460 (2024)

[22] Golob, S., Pentyala, S., Dowsley, R., David, B., Larangeira, M., Cock, M.D., Nascimento, A.C.A.: A Decentralized Information Marketplace Preserving Input and Output Privacy. In: Proceedings of the Second ACM Data Economy Workshop, Seattle, WA, USA, 18 June 2023. DEC '23, pp. 1–6. ACM, New York, NY, USA (2023). https://doi.org/10.1145/3600046.3600047

[23] Zbudila, M., Suresh, A., Yalame, H., Mirzamohammadi, O., Abidin, A., Preneel, B.: SoK: Connecting the Dots in Privacy-Preserving ML - Systematization of MPC Protocols and Conversions Between Secret Sharing Schemes (2025). https://eprint.iacr.org/2025/1679

[24] Ng, L.K.L., Chow, S.S.M.: SoK: Cryptographic Neural-Network Computation. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 497–514 (2023). https://doi.org/10.1109/SP46215.2023.10179483

[25] Li, D., Chattopadhyay, A., Li, Q., Lü, Q., Wu, J., Xiang, T., Liao, X.: CryptDNN: A Fast Privacy-Preserving Deep Neural Network Inference Architecture Based

on Cloud-Edge-Client Collaboration. IEEE Transactions on Network Science and Engineering, 1–15 (2025) https://doi.org/10.1109/TNSE.2025.3620373

[26] Yang, W., Wang, H., Li, Z., Niu, Z., Wu, L., Wei, X., Su, Y., Susilo, W.: Privacy-Preserving Machine Learning in Cloud–Edge–End Collaborative Environments. IEEE Internet of Things Journal **12**(1), 419–434 (2025) https://doi.org/10.1109/JIOT.2024.3461410

[27] Wang, C., Xu, J., Chen, J., Varadharajan, V., Lewis, C.: PrivESD: A Privacy-Preserving Cloud-Edge Collaborative Logistic Regression Model over Encrypted Streaming Data. IEEE Transactions on Dependable and Secure Computing, 1–16 (2025) https://doi.org/10.1109/TDSC.2025.3617797

[28] Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority. In: Coron, J.-S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017, pp. 225–255. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_8

[29] Zha, D., Bhat, Z.P., Lai, K.-H., Yang, F., Jiang, Z., Zhong, S., Hu, X.: Data-centric Artificial Intelligence: A Survey **57**(5), 129–112942 https://doi.org/10.1145/3711118

[30] Singh, D., Reddy, C.K.: A survey on platforms for big data analytics. Journal of Big Data **2**(1), 8 (2014) https://doi.org/10.1186/s40537-014-0008-6

[31] The Cloud Native Computing Foundation: Kubernetes website. https://kubernetes.io/ Accessed 2025-04-29

[32] Bracke, V., Sebrechts, M., Moons, B., Hoebeke, J., De Turck, F., Volckaert, B.: Design and evaluation of a scalable Internet of Things backend for smart ports. Software: Practice and Experience **51**(7), 1557–1579 (2021) https://doi.org/10.1002/spe.2973

[33] Sharma, R., Mathur, A.: Traefik API Gateway for Microservices. Apress, Berkeley, CA (2021). https://doi.org/10.1007/978-1-4842-6376-1

[34] Kubernetes Ingress Controllers. https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/ Accessed 2025-04-29

[35] Kubernetes Ingress. https://kubernetes.io/docs/concepts/services-networking/ingress/ Accessed 2025-04-29

[36] Apache Kafka. https://kafka.apache.org/ Accessed 2025-04-29

[37] ClickHouse: ClickHouse website. https://clickhouse.com/clickhouse Accessed 2025-04-29

[38] HTML Standard: Server-sent events. https://html.spec.whatwg.org/multipage/

server-sent-events.html#server-sent-events Accessed 2025-04-29

[39] Team Keycloak: Keycloak. https://www.keycloak.org/ Accessed 2025-04-29

[40] OpenFGA: Fine-Grained Authorization. https://openfga.dev/ Accessed 2025-04-29

[41] Evans, D., Kolesnikov, V., Rosulek, M.: A Pragmatic Introduction to Secure Multi-Party Computation. Foundations and Trends in Privacy and Security **2**(2-3), 70–246 (2018) https://doi.org/10.1561/3300000019

[42] Beaver, D.: Efficient Multiparty Protocols Using Circuit Randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91, pp. 420–432. Springer, Berlin, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34

[43] Mohassel, P., Rindal, P.: ABY3: A Mixed Protocol Framework for Machine Learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS '18, pp. 35–52. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243760

[44] Zbudila, M., Pohle, E., Abidin, A., Preneel, B.: MaSTer: Maliciously Secure Truncation for Replicated Secret Sharing Without Pre-processing. In: Kohlweiss, M., Di Pietro, R., Beresford, A. (eds.) Cryptology and Network Security, pp. 49–73. Springer, Singapore (2025). https://doi.org/10.1007/978-981-97-8013-6_3

[45] Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM **21**(2), 120–126 (1978) https://doi.org/10.1145/359340.359342

[46] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Advances in Cryptology - EUROCRYPT '99. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer, Berlin, Heidelberg, Germany (1999). https://doi.org/10.1007/3-540-48910-X_16

[47] Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing. STOC '09, pp. 169–178. ACM, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536440

[48] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast Fully Homomorphic Encryption Over the Torus. Journal of Cryptology **33**(1), 34–91 (2020) https://doi.org/10.1007/s00145-019-09319-x

[49] Ducas, L., Micciancio, D.: FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, pp. 617–640. Springer, Berlin, Heidelberg (2015). https:

//doi.org/10.1007/978-3-662-46800-5_24

[50] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully Homomorphic Encryption without Bootstrapping (2011). https://eprint.iacr.org/2011/277

[51] Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012, pp. 868–886. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_50

[52] Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption (2012). https://eprint.iacr.org/2012/144

[53] Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic Encryption for Arithmetic of Approximate Numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15

[54] Aranha, D.F., Costache, A., Guimarães, A., Soria-Vazquez, E.: HELIOPOLIS: Verifiable Computation over Homomorphically Encrypted Data from Interactive Oracle Proofs is Practical. In: Chung, K.-M., Sasaki, Y. (eds.) Advances in Cryptology – ASIACRYPT 2024, pp. 302–334. Springer, Singapore (2025). https://doi.org/10.1007/978-981-96-0935-2_10

[55] Gama, M., Beni, E.H., Kang, J., Spiessens, J., Vercauteren, F.: Blind zkSNARKs for Private Proof Delegation and Verifiable Computation over Encrypted Data. Cryptology ePrint Archive, Paper 2024/1684 (2024). https://eprint.iacr.org/2024/1684

[56] Checri, M., Sirdey, R., Boudguiga, A., Bultel, J.-P.: On the Practical CPA D Security of "exact" and Threshold FHE Schemes and Libraries. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, pp. 3–33. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-68382-4_1

[57] Cheon, J.H., Choe, H., Passelègue, A., Stehlé, D., Suvanto, E.: Attacks Against the IND-CPAD Security of Exact FHE Schemes. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. CCS '24, pp. 2505–2519. Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3658644.3690341

[58] Jones, M.B., Bradley, J., Sakimura, N.: JSON Web Token (JWT). RFC Editor (2015). https://doi.org/10.17487/RFC7519

[59] Sheffer, Y., Hardt, D., Jones, M.B.: JSON Web Token Best Current Practices. RFC Editor (2020). https://doi.org/10.17487/RFC8725

[60] Redis: Redis - The Real-time Data Platform. https://redis.io/ Accessed 2025-05-02

[61] Network Policies. https://kubernetes.io/docs/concepts/services-networking/network-policies/ Accessed 2025-05-02

[62] Fujisaki, E., Okamoto, T., Pointcheval, D., Stern, J.: RSA-OAEP Is Secure under the RSA Assumption. In: Kilian, J. (ed.) Advances in Cryptology – CRYPTO 2001. Lecture Notes in Computer Science, vol. 2139, pp. 260–274. Springer, Santa Barbara, CA, USA (2001). https://doi.org/10.1007/3-540-44647-8_16

[63] Abdalla, M., Bellare, M., Rogaway, P.: DHIES: An Encryption Scheme Based On The Diffie-Hellman Problem. Contributions to IEEE P1363a (1998)

[64] Kachuee, M., Fazeli, S., Sarrafzadeh, M.: ECG Heartbeat Classification: A Deep Transferable Representation. In: 2018 IEEE International Conference on Healthcare Informatics (ICHI), pp. 443–444 (2018). https://doi.org/10.1109/ICHI.2018.00092

[65] Moody, G.B., Mark, R.G.: The impact of the MIT-BIH Arrhythmia Database. IEEE Engineering in Medicine and Biology Magazine **20**(3), 45–50 (2001) https://doi.org/10.1109/51.932724

[66] Bousseljot, R., Kreiseler, D., Schnabel, A.: Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet **40**(s1), 317–318 (1995) https://doi.org/10.1515/bmte.1995.40.s1.317

[67] Medical Instrumentation, A.: Testing and reporting performance results of cardiac rhythm and ST segment measurement algorithms. In: ANSI/AAMI EC57:2012/(R)2020; Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement algorithms, (2022). https://doi.org/10.2345/9781570204784.ch1

[68] Web Crypto API - MDN (2024). https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API Accessed 2025-06-02

[69] Morita, H., Pohle, E., Sadakane, K., Scholl, P., Tozawa, K., Tschudi, D.: MAESTRO: Multi-party AES using Lookup Tables (2024). https://eprint.iacr.org/2024/1317

[70] Dworkin, M.: Recommendation for block cipher modes of operation: Galois/-counter mode (GCM) and GMAC. Technical report, National Institute of Standards and Technology (2007). https://doi.org/10.6028/NIST.SP.800-38D . NIST Special Publication 800-38D

[71] Keller, M.: MP-SPDZ: A Versatile Framework for Multi-Party Computation. In:

Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. CCS '20, pp. 1575–1590. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3417872

[72] Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPDZ2k: Efficient MPC mod 2k for Dishonest Majority. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, pp. 769–798. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_26

[73] Zbudila, M., Abidin, A., Preneel, B.: Exploring Adversarial Attacks on the MaSTer Truncation Protocol. Cryptology ePrint Archive, Paper 2025/773 (2025). https://eprint.iacr.org/2025/773

[74] Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: OpenFHE: Open-Source Fully Homomorphic Encryption Library. In: Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. WAHC '22, pp. 53–63. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3560827.3563379

[75] Halevi, S., Shoup, V.: Algorithms in HElib. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, pp. 554–571. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_31

[76] iPerf: The TCP, UDP and SCTP network bandwidth measurement tool. https://iperf.fr/ Accessed 2025-05-10

[77] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org (2015). https://www.tensorflow.org/

[78] TensorFlow. https://www.tensorflow.org/ Accessed 2025-05-10

[79] Izquierdo, S.: cppflow: Run TensorFlow models in C++ without installation and without Bazel (2019). https://doi.org/10.5281/zenodo.7107621

[80] TensorFlow C API. https://github.com/tensorflow/tensorflow/blob/781baea39c7c8eb63aa357dd10cc34e37344a3d5/tensorflow/c/c_api.h Accessed 2025-05-10

[81] Defense Advanced Research Projects Agency (DARPA): DPRIVE: Data Protection in Virtual Environments (2025). https://www.darpa.mil/research/programs/data-protection-in-virtual-environments Accessed 2025-05-23

[82] Niobium: A FHE Hardware Accelerator (2025). https://niobiummicrosystems.com/products/fhe-solutions/ Accessed 2025-05-23