EasyChair Preprint

№ 15580

Priv-IoT: Privacy-Preserving Machine Learning in IoT Utilizing TEE and Lightweight Ciphers

Arash Kariznovi and Kalikinkar Mandal

December 16, 2024

Priv-IoT: Privacy-preserving Machine Learning in IoT Utilizing TEE and Lightweight Ciphers

Arash Kariznovi and Kalikinkar Mandal[0000−0002−8228−5016]

University of New Brunswick
Faculty of Computer Science
Fredericton, New Brunswick, E3B 5A3, CANADA
{arash.k,kmandal}@unb.ca

Abstract. The need for lightweight cryptographic primitives is greater than ever due to the rapid advancements in the Internet of Things (IoT) and the increasing presence of resource-constrained devices. In response to this, the NIST has standardized the ASCON lightweight authenticated encryption with associated data (AEAD) and hash algorithm as a standard for lightweight cryptography (LWC). Besides protected IoT data communications, IoT data analytics is crucial for operational efficiency, data-driven innovation, improved decision-making, and predictive maintenance. We consider a real-world scenario of Cloud-IoT where an IoT application is connected to a (potentially untrusted) cloud. In this paper, we propose Priv-IoT, a privacy-preserving machine learning (PPML) system, using it an IoT application owner can securely transport IoT data to the cloud and enable secure machine learning (ML) on the IoT data. Our secure IoT data transport protocol is based on a lightweight AEAD scheme and a standard security protocol (e.g., TLS) to resist against various external and internal attacks. We enable secure ML analytics using a trusted execution environment (e.g., Intel-SGX) in the bring-your-own-encryption paradigm. We prototype and evaluate our proposed system using a list of LWC algorithms and fundamental regression algorithms in SGX, and present extensive experimental results on real-world datasets.

Keywords: IoT · Machine Learning · Trusted Execution Environment · Lightweight Cryptography.

1 Introduction

With technological advancements such as IoT and smart grids, the demand for resource-constrained devices like smart meters, smart cards, RFIDs, and low-power wireless devices is increasing rapidly. Due to the constrained nature of these devices, providing security and privacy is a significant concern and it is not possible to use the conventional cryptography standards that are used for high-speed powered processors like servers and desktops. The devices mentioned use hardware or software implementation to perform the cryptography computations. RFIDs are embedded with FPGAs and ASICs and smart meters use 16-bit, 8-bit, or sometimes 4-bit microcontrollers. When designing a cryptographic primitive, there is always a trade-off between resource and efficiency

for a constant security level. In lightweight cryptography, it is much harder since the researchers are limited in using resources and they have to meet a certain security level at the same time. The resources for software include RAM, ROM, and registers, while the resources for hardware implementation are gate equivalents and gate area. Latency, power and energy consumption, and throughput are the key metrics to evaluate the performance of these primitives. Over the past more than a decade, several cryptographic primitives have been proposed including block ciphers, stream ciphers, message authentication code, and hash functions. In the case of block ciphers, DESL [29] has been proposed by reducing the S-box rounds from eight to one round and, Present [10], was designed as a dedicated algorithm. Moreover, Simon and Speck [5], RC5 [36], TEA [39] and XTEA [32] are proposed for efficient performance on resource constrained devices. Lightweight hash functions including PHOTON [22], Quark [1], SPONGNET [9] and Lesamnta-LW [24] are developed that have smaller state size and lower energy consumption. Chaskey [31], TuLP [20], and LightMAC [30] are some of the examples of lightweight MAC algorithms developed on hardware applications. The growth in IoT and smart grid, has led to the initiation of many communications between many of previously unconnected nodes in the network and thereby introducing new security and privacy challenges. The data from smaller nodes are transmitted to the more powerful nodes to be processed, aggregated or stored but data owners are sensitive about the privacy of their data. For instance, there are many sensors in a smart home, these sensors collects the data from different appliances and transmit it to the utility providers server for further analysis. With accessing this data, adversaries can understand about the pattern of the residences and extract important information. In the smart grid, thousands of smart meters send the metering data to the data aggregator, or as another example, hospitals send their patient health information to the third-party cloud provider to be processed. The adversary can be an attacker far away in another continent or an insider that is working in the utility provider company. The data must be protected against all types of misuse. A solid solution to this challenge is using Trusted Execution Environments (TEEs). TEE uses the enclaves as a protected area in the memory suitable for maintaining the sensitive information, the data residing in this area is encrypted and is only decrypted when it is being processed by CPU. By using this technology, confidentiality and integrity of the private data is guaranteed and even the OS and system administrator do not have access to its data and code. Finally, if data is encrypted using the mentioned lightweight ciphers in the resource-constrained devices, they need to be decrypted using the same lightweight algorithm before being used in the cloud or data aggregator. This shows that the resource-constrained devices are not the only devices in which lightweight primitives are performed and assessing their performance in the powerful processors is required and important equally. Although TEE is a beneficial tool and many benchmarks performed on NIST finalists, to the best of our knowledge no research has evaluated the performance of those lightweight algorithms on Trusted Execution Environments.

Our contributions. Motivated by the deployment of Cloud-IoT system in the real-world and the need for secure IoT data processing, we propose Priv-IoT, a privacy-preserving system for secure collection and processing of IoT in the cloud. Our contributions in this paper are summarized as follows:

- **Privacy-preserving ML system for Cloud-IoT:** We present a privacy-preserving machine learning system tailored for Cloud-IoT systems. The Priv-IoT system has three main entities namely an IoT application, the owner of the application and a cloud. We leverage well-analyzed/understood cryptographic schemes and security protocols to design our secure system. Our Priv-IoT system is built upon four fundamental building blocks namely a lightweight AEAD scheme, a TEE realized by SGX, the TLS security protocol, and a key-exchange protocol for enclave. Our choice of cryptograhic primitives and protocols for IoT devices is done so that it meets the lightweight requirement. We prove the security of our proposed system against semi-honest adversaries.
- **Evaluating NIST LWC finalists in SGX.** As an independent contribution, we evaluate and benchmark the execution of the encryption and decryption algorithms of ten (finalists) AEAD algorithms from the NIST LWC competition. Our experimental shows that the overhead of running encryption/decryption algorithms inside SGX, compared to the cleartext computation, lies in the range of 1.06 and 2.23. To the best of our knowledge, our work is the first to perform such analysis of lightweight ciphers.
- **Experimental evaluation.** To capture the real-world Cloud-IoT scenario, we develop a testbed using a `Raspberry Pi Pico` (IoT device), `Raspberry Pi 4B` (IoT gateway) and the Microsoft Azure cloud. We implement our privacy-preserving ML system on IoT devices and the Azure cloud. We consider linear and logistic regression algorithms and evaluate the performances on 11 different real-world datasets from the UCI ML repository. We present experimental results for different regression training.

2 Preliminaries

2.1 Security and Privacy in IoT

Security must be lightweight due to IoT's resource limitations. It involves securing devices across layers: the perception layer against attacks, the network layer for secure routing, and the application layer with authentication and access control. The fundamental security properties of IoT data and systems include authentication, confidentiality, integrity, and availability. Moreover, privacy is critical due to an autonomous data transmission of IoT devices. Even fragmented data can reveal sensitive information when combined. Users should control their data dissemination [12].

2.2 Lightweight Cryptography

Despite all available algorithms, the National Institute of Standards and Technology was planning to develop a family of algorithms that are more fast and

lightweight, and still secure in resource-constrained devices. In 2018, the NIST published a call for algorithms to describe the requirements, selection process, and evaluation criteria [34]. After two rounds of the NIST standardization process, the final rounds began by selecting ten algorithms namely, ASCON [16], Elephant [8], GIFT-COFB [3], Grain- 128AEAD [23], ISAP [15], PHOTON-Beetle [4], Romulus [25], SPARKLE [6], TinyJAMBU [40], and Xoodyak [14]. The final round ended with the selection of the ASCON family as the NIST lightweight cryptography standard in February 2023.

2.3 Trusted Execution Environments

A Trusted Execution Environment is a secure processing environment that isolates sensitive operations, data, and computations from the normal rich execution environment (REE) where typical operating systems and applications run. TEEs enhance security by ensuring confidentiality and integrity, allowing REE applications to securely partition sensitive tasks into the TEE.

Software Guard Extension. Intel Software Guard Extension (SGX) is a set of instructions added to some of Intel CPUs that ensures the confidentiality, integrity of the code and data by creating a trusted execution environment called an enclave. SGX reserves a memory region called Processor Reserved Memory (PRM). CPU protection keeps the PRM safe from non-enclave memory accesses, including kernel, hypervisor, and SMM accesses, and DMA accesses from peripherals. The PRM manages the Enclave Page Cache (EPC) with 4 KB pages housing enclave code and data. Untrusted system software assigns EPC pages to enclaves, tracked by the CPU in the Enclave Page Cache Metadata (EPCM). Initially, the system software loads code and data into enclaves from unprotected memory, establishing the initial enclave state. Once all pages are loaded, the enclave is marked as initialized, allowing application code execution inside. Enclave execution occurs in protected mode, with CPU-enforced isolation. Interruptions during enclave execution trigger an Asynchronous Enclave Exit (AEX) to handle them securely. OS or hypervisor manages EPC page allocation, using cryptographic protections for evictions to untrusted memory.

Attestation plays a vital role in SGX usage and comes in two forms: remote attestation and local attestation. Remote attestation is employed when an external client seeks to connect to the enclave. Before key exchange, the client must verify the SGX enclave. This involves requesting the enclave to generate a report, which is then passed to a platform service to produce a credential known as a quote, reflecting the enclave and platform state. The client receives the quote and employs the Intel Attestation Service to verify the enclave, enabling trust based on the IAS response for key exchange. On the other hand, local attestation is used when two enclaves within a platform wish to communicate. Each enclave sends a report to the other enclave, allowing both sides to verify each other using this report and a symmetric key (GETKEY).

2.4 Transport Layer Security

Transport Layer Security (TLS) is a cryptographic protocol ensuring secure communication over networks, most notably securing HTTPS. It provides privacy,

integrity, and authenticity using techniques like certificates. TLS evolved from SSL and is currently at version 1.3 (defined in 2018). Operating at the presentation layer, it has two main components:

- **TLS Handshake**: Establishes a secure communication channel by exchanging messages between a client and server. The process involves agreeing on cryptographic parameters, authenticating identities, and generating session keys. In TLS 1.3, the handshake includes a client hello message, a server hello message, and key exchange to finalize secure communication.
- **TLS Record Protocol**: Secures data transmission after the handshake by ensuring confidentiality, integrity, and authenticity. It segments, compresses (optional), authenticates, and encrypts outgoing data, while performing the reverse on incoming data. Encrypted data is then passed to the TCP layer for transport.

3 Related Work

Lightweight cryptography. While we could not find any studies to evaluate the NIST ten finalists performance on trusted execution environments, there are several benchmarks available for software and hardware implementations. These implementations evaluate the performance of NIST finalists on FPGAs, microcontrollers and desktops and servers bringing a wide range of insights about their efficiency on different devices from resource-constrained devices in the edge to powerful servers on the cloud. The NIST team compared AEAD finalists to AES-GCM and SHA-256 on 8-bit and 32-bit microcontrollers, with ASCON, TinyJAMBU, and GIFT-COFB performing best [33]. Weatherley [38] optimized finalists on AVR, ESP32, and Cortex-M3, with ASCON, GIFT-COFB, SPARKLE, and Xoodyak outperforming ChaChaPoly on AEAD benchmarks. SPARKLE was fastest overall. eBACS [7] showed ASCON and Xoodyak outperforming AES-GCM without AES-NI and on ARM Cortex-A7. Both were competitive with SHA-256. Renner et al. [35] benchmarked AEAD finalists on five microcontrollers, with ASCON leading in speed, code size, and RAM usage.

Privacy-preserving ML using TEE/SGX. The IoT has various applications of regression analysis. In general, privacy-preserving regression analysis received considerable attention. There are various ways to realize a privacy-preserving regression system such as homomorphic encryption (e.g., [11,19,27]), secret sharing (e.g., [13,17,18]), and garbled circuit (e.g., [2]). However, we focus on designing private regression systems based on TEE. Several frameworks have been developed to perform secure computations using Intel SGX. PRIVADO [21] ensures secure DNN inference in SGX by eliminating input-dependent access patterns, reducing leakage while maintaining low performance overheads across various models. SecureLR [26] proposes a hybrid cryptographic framework that combines homomorphic encryption and Intel SGX to enable secure logistic regression on biomedical data in public clouds without compromising data security or efficiency. VC3 [37] secures distributed MapReduce by isolating memory regions using SGX, achieving minimal overhead while ensuring confidentiality and integrity even if large components like the OS are compromised . EnclaveDB

secures databases by placing sensitive data within SGX enclaves, ensuring confidentiality, integrity, and freshness with low overhead. Lastly, SGX-PySpark [28] integrates SGX with PySpark to secure distributed data analytics in cloud environments.

4 The Cloud-IoT System Model

In this section, we describe the Cloud-IoT system model, its adversarial model, and present a privacy-preserving machine learning system for IoT ML analytics.

4.1 Our System and Trust Model

We consider a real-world Cloud-IoT system consisting of three key entities, namely a set of IoT devices, a gateway, and a cloud server. The IoT devices are connected to the cloud via a gateway and transmit IoT data to the cloud. We assume that all the devices belong to a single owner who wishes to perform various tasks on IoT data such as data collection, processing, ML learning computations, and visualization. Figure 1 shows a high-level overview of the system.

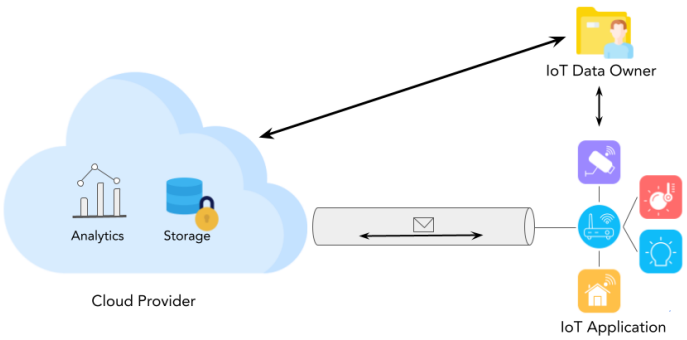


Fig. 1: A Cloud-IoT system connecting an IoT application to the cloud

Problem statement. Suppose the IoT devices streams high-dimensional data to the cloud. Let $\mathcal{D} = \{(\mathbf{x}, y)\}$ be a combined dataset of the same type generated by IoT devices, where $(\mathbf{x}, y)$ is a high-dimensional data point. Suppose the owner wishes to train a regression model $\boldsymbol{\theta}$, i.e., $\boldsymbol{\theta} \leftarrow \mathsf{Training}(\boldsymbol{\theta}, \mathcal{D}, f)$ where f is a linear or logistic regression algorithm. We consider the problem of *securely* transmitting data from the IoT devices to the cloud and *privately* performing training linear or logistic regression models $\boldsymbol{\theta}$ on $\mathcal{D}$ in the cloud. Our system is aimed to protect IoT data privacy and integrity when in-transit, at-rest and in-use, and ensure the fine-grain control that remains exclusively with its legitimate owner, while still allowing them to access and benefit for various purposes such as operational and automation.

Adversarial/Trust model. Our trust model is similar to the one in [37]. In our system, we consider semi-honest adversaries where an adversary may compromise some IoT devices or the cloud applications, and observes the execution of the protocol. The goal of the adversary is to learn any unintended information about other honest IoT devices' data or the trained model. We assume that the

adversary can intercept the IoT data communications including record, replay, and modify network data and can compromise the cloud software applications. We also assume that the adversary is unable to physically open and tamper the SGX-enabled processors located at the cloud. If such is possible, we claim no IoT data and ML model privacy.

5 Our Solution for Private Cloud-IoT Data Analytics

Overview. To design a secure Cloud-IoT analytics system, we take a practical approach and make use of well-understood cryptographic schemes and protocols, namely a trusted execution environment (e.g., SGX), a security protocol (e.g., TLS) and a lightweight AEAD algorithm (e.g., ASCON). In our system, fundamentally there are two tasks: 1) securely transport data from an IoT application to the cloud, and 2) privately perform an analytical algorithm in the cloud so that the cloud cannot infer any private data at-rest and in-use. Our solution takes the bring-your-own-encryption (BYOE) paradigm where the owner manages the keys of IoT devices. To achieve the IoT data security and privacy, we design a doubly-encryption mechanism by combining a lightweight AEAD with the standard TLS-like security protocol to achieve the privacy against external communication attacks and internal attacks in the cloud. We perform analytics securely in a trusted execution environment to achieve IoT data privacy against the cloud. We formalize our secure system in Definition 1. Figure 2 shows a secure version of the system.

Definition 1. *Our secure system consists of a set of four algorithms/sub-protocols involving the IoT devices, the owner, the cloud server and the TEE:*

- $\{\mathsf{K}_{\mathsf{ID}}\}_{\mathsf{ID}\in[n]} \leftarrow \mathsf{KeySetup}(1^\lambda)$*: On a security parameter* λ*, the owner runs this algorithm to generate a set of keys* $\{\mathsf{K}_{\mathsf{ID}}\}_{\mathsf{ID}\in[n]}$ *where the unique key* K_{ID} *is for each IoT device with a unique* ID *and* n *is the number of IoT devices in the system. The owner securely stores and distributes the keys to the IoT devices.*
- $(c, tag, \mathsf{ID}) \leftarrow \mathsf{SecureDataTransport}(\mathsf{K}_{\mathsf{ID}}, (\boldsymbol{x}, y), \mathsf{ID})$ *: The secure data transport protocol is run between an IoT device with identity* ID *and the cloud server where the IoT device gives* $\mathsf{K}_{\mathsf{ID}}, (\boldsymbol{x}, y)$ *as inputs, and the cloud receives* (c, tag, ID) *as output.*
- $(\mathsf{K}_{\mathsf{ID}}, \mathsf{ID}) \leftarrow \mathsf{SecureKeyTransport}(\mathsf{K}_{\mathsf{ID}}, \mathsf{ID})$ *: The secure key transport protocol is run between the owner with identity* ID *and the TEE where the owner gives* K_{ID} *as inputs, and the TEE receives* $(\mathsf{K}_{\mathsf{ID}}, \mathsf{ID})$ *as output.*
- $\boldsymbol{\theta} \leftarrow \mathsf{SecureAnalytics}(\mathcal{D}, \{\mathsf{K}_{\mathsf{ID}}\}, f)$*: After receiving a training request from the owner for an ML training algorithm* f *and the dataset* $\mathcal{D}$*, the TEE runs this algorithm to train a model* $\boldsymbol{\theta} \leftarrow \mathsf{Training}(\boldsymbol{\theta}, \mathcal{D}, f)$ *within the trusted environment and return either an encrypted model or the cleartext model to the request party.*

5.1 Description of Components in Our Secure System

We now provide the details of the security algorithms in Definition 1.

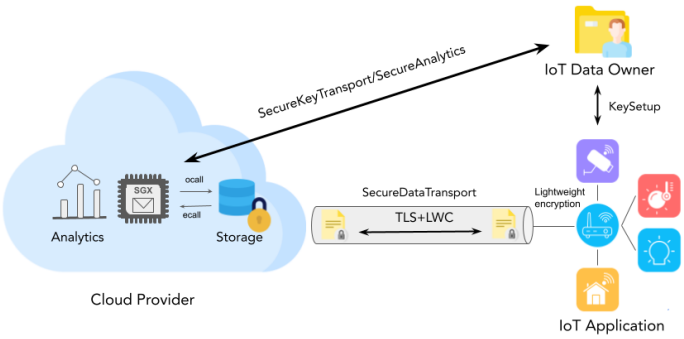


Fig. 2: Overview of Our Privacy-preserving ML System for Cloud-IoT

KeySetup. Suppose there are n IoT devices in the system and each device has a unique identity (ID). Based on a security parameter λ (e.g., $\lambda = 128$-bit security), the owner randomly samples a set of symmetric-keys $\{\mathsf{K}_{\mathsf{ID}}\}_{\mathsf{ID}\in[n]}$, one key K_{ID} is for each IoT device with identity ID. These symmetric-keys will be used in an AEAD algorithm to encrypt and authenticate IoT data.

SecureDataTransport. When an IoT device with identity ID wishes to send a high-dimensional data $(\mathbf{x}, y)$, it first encrypts the data using a lightweight AEAD algorithm to produce $(c, tag) \leftarrow \mathsf{AEAD.Enc}(\mathsf{K}_{\mathsf{ID}}, (\mathbf{x}, y))$ where c is a ciphertext and tag is an authentication tag. The encrypted data (c, tag) is put in the payload of the corresponding application layer protocol. Our rationale for choosing a (symmetric-key) lightweight AEAD is to have a reduced computational complexity of IoT devices and less ciphertext expansion which will grately save space in the cloud.

Next, the gateway/IoT device establishes a TLS connection with the cloud and transmit the message $M := (c, tag, \mathsf{ID})$ over TLS as $(C, T) \leftarrow \mathsf{TLS.Enc}(M)$ where the TLS connection establishment includes the handshake, key establishment and record protocols. We omit these details. When the cloud receives (C, T) from the TLS, it removes the TLS layer encryption as $M \leftarrow \mathsf{TLS.Dec}(C, T)$ to obtain (c, t, ID) and stored in the database in a structured way.

SecureKeyTransport. As the owner manages the keys for the IoT devices, to decrypt an AEAD ciphertext inside the TEE/enclave, the corresponding symmetric-key has to be transported to the TEE using a security protocol and the decryption algorithm is run inside the enclave. For this purpose, the owner and the enclave first execute a key-exchange protocol, called enclave key-exchange protocol (EKEP), to establish a shared secret-key, which the owner uses to securely transport the key of the AEAD algorithm for a particular IoT device or an AEAD encrypted datasets.

SecureAnalytics. Before performing an analytics inside TEE, a necessary first step is to execute the $\mathsf{SecureKeyTransport}$ protocol to securely transport the key(s) to the enclave. As IoT data are stored in the encrypted form, data is transmitted to a remote attested enclave, which has received AEAD key(s) via

the remote attestation. To start an analytical computation (e.g., training) within the enclave, encrypted data is decrypted first using the received key(s) and then the decrypted data/plaintext is fed into the analytics algorithms such as regression training. That is, $\boldsymbol{\theta} \leftarrow \mathsf{Training}(\boldsymbol{\theta}, \mathcal{D}, f)$ is run in the enclave. Finally, the resulting output/model $\boldsymbol{\theta}$ is encrypted or sent to the cloud without any knowledge of the fine-grained data.

Memory optimized training. Although our proposed system works for any ML training, we restrict ourselves to linear and logistic regression in this work. We propose two implementations of training algorithms that can be implemented as follows: 1) full enclave memory implementation – loading the entire dataset $\mathcal{D}$ in the enclave memory; and 2) low enclave memory implementation – loading a subset of $\mathcal{D}$ as needed by the training. For instance, to train a model for a linear regression model, one can build the model by solving a system of equations which will need the entire matrix associated with the dataset to be loaded in the enclave. On the other hand, training a linear model using a stochastic gradient descent (SGD) or mini-batch gradient descent requires a single or subset of data point loaded in the enclave by exploiting the tradeoff between the (secondary) storage and computation (performing the decryption of data stored in the secondary storage).

5.2 Security Discussion

In this section, we present a (semi-)formal security proof of our system. Recall that our system is built upon four fundamental secure building blocks namely a lightweight AEAD scheme, the TLS protocol, an enclave key-exchange protocol (EKEP) and a TEE realized by SGX.

Theorem 1. *Assume that the lightweight* AEAD *scheme is secure under indistinguishability under chosen plaintext attack (IND-CPA), the TLS and EKPE protocols are secure, and the TEE realized by SGX is trusted. Our proposed system is secure against semi-honest adversaries.*

Proof (Sketch). In our system, there are two types of adversaries. We use $\mathcal{A}_i$ to denote internal adversaries and $\mathcal{B}_i$ to denote external adversaries. Both adversaries may collude that can be considered as a single adversary. As the AEAD scheme is IND-CPA secure, the advantage of a (polynomial-time) adversary $\mathcal{A}_0$, denoted by $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}_0}$, is negligible. We consider an IND-CPA adversary $\mathcal{B}_0$ for TLS where the adversary $\mathcal{B}_0$'s goal is to distinguish encrypted messages transmitted over a TLS-secured communication. The security of TLS assures that the advantage of $\mathcal{B}_0$, denoted by $\mathsf{Adv}^{\mathsf{TLS}}_{\mathcal{B}_0}$, is negligible. The security of the EKEP, denoted by $\mathcal{B}_1$, is defined using the indistinguishability of the shared secrey-key, where there is no polynomial-time adversary that can distinguish the shared-key/session key from a random key. The security of the EKEP assures that the advantage of $\mathcal{B}_1$, denoted by $\mathsf{Adv}^{\mathsf{EKEP}}_{\mathcal{B}_1}$, is negligible. The SGX security guarantees that no adversary $\mathcal{A}_1$ can distinguish between real and simulated enclave data (analytical algorithm's code and metering data) and can break the

integrity or tampering the computation with a valid output of f. The advantage of $\mathcal{A}_1$, denoted by $\mathsf{Adv}^{\mathsf{TEE}}_{\mathcal{A}_1}$ is negligible. We argue that the advantage of a (semi-honest) adversary $\mathcal{A}$ (internal or external) to break IoT data and analytical computation confidentiality with respect to IND-CPA in all phases is bounded by $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} \leq 2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}_0} + \mathsf{Adv}^{\mathsf{TLS}}_{\mathcal{B}_0} + \mathsf{Adv}^{\mathsf{EKEP}}_{\mathcal{B}_1} + \mathsf{Adv}^{\mathsf{TEE}}_{\mathcal{A}_1}$. We define the real system's security with the IND-CPA game where the adversary $\mathcal{A}$ interacts with the real system and aims to distinguish an IoT data at any phase. The initial game is the standard IND-CPA game between the adversary $\mathcal{A}$ and the challenger. The advantage of $\mathcal{A}$ is $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}$.

- *Reduction to the AEAD security:* In this game G_1, we replace an original ciphertext $(c, tag) \leftarrow \mathsf{AEAD.Enc}(\mathsf{K_{ID}}, (\mathbf{x}, y))$ with an encryption of a random plaintext $(\mathbf{r}_x, r_y)$, i.e., $(r_c, r_{tag}) \leftarrow \mathsf{AEAD.Enc}(\mathsf{K_{ID}}, (\mathbf{r}_x, r_y))$. Since AEAD is IND-CPA secure, the ciphertexts c and r_c are indistinguishable. $\mathcal{A}$'s advantage in distinguishing the original game and this game is negligible. The difference between the advantages is $|\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} - \mathsf{Adv}^{G_1}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}_0}$.
- *Reduction to the TLS security:* In this game G_2, we replace actual TLS encryption with a TLS encryption of a random value $M' := (r_c, r_{tag}, r_{\mathsf{ID}})$, i.e., $(C', T') \leftarrow \mathsf{TLS.Enc}(M')$ By the security of TLS, $\mathcal{A}$'s advantage in distinguishing between G_1 and G_2 is negligible, and we have $|\mathsf{Adv}^{G_1}_{\mathcal{A}} - \mathsf{Adv}^{G_2}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{TLS}}_{\mathcal{B}_0}$.
- *Reduction to the EKEP security:* In this game G_3, we simulate the EKEP protocol and obtain a shared-key r_K and then use the shared-key r_K in the AEAD to encrypt $\mathsf{K_{ID}}$, i.e., $\mathsf{AEAD.Enc}(r_K, \mathsf{K_{ID}})$. The security of EKEP and AEAD assures that $\mathsf{AEAD.Enc}(r_K, \mathsf{K_{ID}})$ and $\mathsf{AEAD.Enc}(K_s, \mathsf{K_{ID}})$ are indistinguishable and the adversary's advantage is negligible. So, we have $|\mathsf{Adv}^{G_2}_{\mathcal{A}} - \mathsf{Adv}^{G_3}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{EKEP}}_{\mathcal{B}_1} + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}_0}$.
- *Reduction to the TEE security:* In this game G_4, we replace the TEE with a simulated enclave that guarantees confidentiality and integrity of the data and computation. This game is indistinguishable from the previous one due to the security of the TEE, and A's advantage is negligible. Therefore, we have $|\mathsf{Adv}^{G_4}_{\mathcal{A}} - \mathsf{Adv}^{G_4}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{TEE}}_{\mathcal{A}_1}$.

Combining all reductions, we can conclude the claimed advantage bound of the adversary $\mathcal{A}$, which is negligible. Hence, the overall system's IoT data privacy is preserved. We emphasize that the integrity against the semi-honest adversary is assured. Due to the space limit, we skip the security arguments.

6 Experimental Evaluation

In this section, we evaluate our system when instantiate by lightweight AEAD schemes from the NIST LWC competition and the TEE by Intel-SGX. We present the benchmarking results on lightweight AEAD schemes in SGX, followed by training linear and logistic regression models in SGX.

6.1 Setup

Testbed setup. We implement our Cloud-IoT data analytics framework, ensuring privacy and security of users' sensitive data, controlled by a legitimate

data owner. We simulate our system model, shown in Figure 2, an IoT device that is a `Raspberry Pi Pico` and a gateway using `Raspberry Pi 4B`. We use a Microsoft Azure virtual machine with an `Intel Xeon (Ice Lake)` processor, 8 GB of RAM, and 4 GB of memory EPC. The Raspberry Pico is a resource-constrained device with an `ARM Cortex-M0+` processor, while the Raspberry Pi 4B, equipped with a `64-bit quad-core ARM Cortex-A7` processor, functions as a more powerful gateway. The Raspberry Pi 4B is connected to both the Microsoft Azure cloud and the Raspberry Pico. We program the Pico device so that data is set at a regular time interval and transmitted via the gateway to the Microsoft Azure cloud. Figure 3 shows a snapshot of our Cloud-IoT system implementation in our lab environment.

We prototype our system in `C/C++`. We realize a secure channel between the Raspberry Pi 4B and the cloud using the `OpenSSL` library, while the Raspberry Pi Pico uses `mbedTLS`, a lighter version of TLS, to connect to the Raspberry Pi 4B. Before performing the TLS, we use the NIST lightweight AEAD to encrypt the IoT data in the application layer where we put the AEAD ciphertext and tag in the payload. The second layer of security is enable by the standard TLS protocol before it is transmitted to the Azure cloud using the TCP/IP protocol.

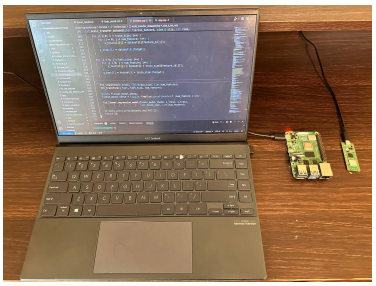


Fig. 3: An Overview of Our Testbed Setup using the Azure cloud, Raspberry Pi Pico and Raspberry Pi 4B Devices

6.2 Results on Lightweight Ciphers in SGX

AEAD implementation details in SGX. We use a lightweight AEAD as a symmetric-key encryption and integrity protection algorithm as it is suitable for the IoT devices. We instantiate the AEAD algorithm by ten finalists, mentioned in Section 2.2, from the NIST LWC competition. We use the source code implementations of these ciphers from the NIST LWC competition website [34].

We evaluate the performance of each AEAD encryption/decryption inside the SGX by counting the number of clock cycles. We also obtain the execution time outside of SGX to compare the overhead introduced by the security operations of SGX. In our experiment, we use 128-bit plaintext data and an empty associated data for both encryption and decryption operations.

In our experiment, encryption and decryption functions are performed five times for each AEAD scheme, and the average clock cycles are computed for

both inside and outside the enclave. To implement an SGX version of each AEAD scheme, we create an `E-call` that triggers the encryption/decryption function and add this `E-call` to the `Enclave.edl` file. The AEAD scheme code and headers are included in the trusted part of the program, and the headers are added in the `Enclave.cpp` file. The corresponding function is run inside the `E-call` function in `Enclave.cpp`. To compare performance with the untrusted version, we measure the `E-call` clock cycles for all algorithms five times and compute the average time similarly.

Benchmark results. Table 1 shows the number of clock cycles for the encryption operation for both trusted and untrusted execution of ten NIST finalists. Despite ASCON being in the NIST LWC standard and demonstrating a robust performance, TinyJambu exhibits a better performance, achieving the shortest encryption and decryption times in both environments. In contrast, Grain128 and Photon-Beetle exhibit the longest processing time, particularly in SGX. Table 1 also shows the overhead introduced by the SGX operations for the encryption of each AEAD algorithm. The cipher Romulus has the highest overhead that is 2.23×, and ISAP has the lowest overhead which is 1.06×.

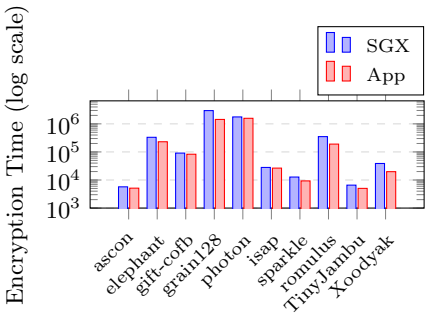


Fig. 4: Encryption Time in SGX and App

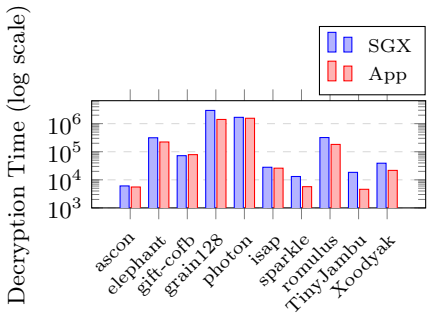


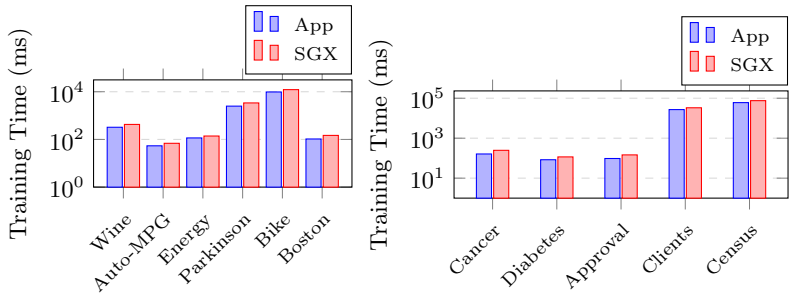
Fig. 5: Decryption Time in SGX and App

Table 1: Encryption Performance Comparison: SGX vs App

AEAD Cipher	**SGX Encryption** (clock cycle)	**App Encryption** (clock cycle)	**Overhead** (×)
Ascon	5,708	5,122	1.11
Elephant	331,888	212,682	1.56
GIFT-COFB	90,864	72,918	1.25
Grain128	2,972,530	1,373,994	2.16
ISAP	28,020	26,520	1.06
Photon-Beetle	1,764,714	1,554,082	1.14
Romulus	350,980	157,296	2.23
TinyJambu	6,486	5,544	1.17
Sparkle	12,748	5,656	2.25
Xoodyak	38,764	19,818	1.95

6.3 Results on Regression Analysis in SGX

Training implementation details in SGX. We assume that the IoT devices already streamed their data to the cloud. To protect data privacy at-rest against the cloud provider, we store ML data to be trained in the encrypted form using a lightweight AEAD algorithm and fetch within an enclave when the training is performed. Before we perform training, we first decrypt the ciphertext inside the enclave. Our implementation design of training linear and logistic models in SGX are similar to the one in Section 6.2. For an SGX-memory optimized implementation of training, we perform a stochastic gradient descent (SGD)-based training where we do not require to load all training data inside the enclave simultaneously.



In our experiment, we implemented two types of regression models, namely linear regression and logistic regression and train them on 11 different datasets, mentioned in Section 6.1, in both the SGX environment and an untrusted environment. We measured the training duration for all models 5 times.

Timing results. As the training time inside the SGX environment includes the timings of the decryption of AEAD ciphertexts and the linear/logistic training time. However, to better understand the only training time, we do not include the time taken the decryption and report the training time. First section in Table 2 shows the execution time in millisecond (*ms*) to train a logistic regression model on the Diabetes, Credit Approval, Breast Cancer, Credit Card Clients, and US Census Income datasets. Compared to training in an untrusted environment, the overhead of training the model in SGX varies from 32 milliseconds to 15,319 milliseconds. In Table 2, n and d denote the dimension and the size of a dataset. The second section of Table 2 shows the execution time to train a linear regression model on the Auto MPG, Boston Housing, Energy Efficiency, Wine Quality, Parkinson's Telemonitoring, and Bike Sharing datasets. Compared to training in an untrusted environment, the overhead of training a linear model in SGX varies from 15 milliseconds to 2,471 milliseconds. Larger datasets and those with more features generally experience a greater increase in training time in the secure environment.

7 Conclusion and Future Work

In this paper, we presented a privacy-preserving ML system for Cloud-IoT that enables secure collection and training of ML models on fine-grained IoT data. By

Table 2: Processing times and dataset details for regression models.

Model Type	Name	n	d	Untrusted (ms)	SGX (ms)	Overhead (×)
Logistic Regression	Diabetes	9	768	83	115	1.387
	Credit Approval	14	652	96	145	1.515
	Breast Cancer	32	454	162	247	1.523
	Credit Card Clients	24	30,000	26,972	33,245	1.231
	US Census Income	20	48,842	60,276	75,595	1.255
Linear Regression	Auto MPG	8	392	53	68	1.283
	Boston Housing	14	506	104	147	1.415
	Energy Efficiency	9	768	115	138	1.217
	Wine Quality	12	1,599	323	425	1.316
	Parkinson	22	5,875	2,489	3,392	1.362
	Bike Sharing	12	17,379	9,777	12,248	1.251

employing a lightweight AEAD scheme for encryption and leveraging TEE for in-use data protection, our system ensures data confidentiality and integrity. We validated its practicality by encrypting IoT testbed data, securely transmitting it to the cloud, and training logistic and linear regression models on real-world datasets. As a future work, we have been continuing to implement application-specific deep learning algorithms in our testbed.

Acknowledgment. This work was supported by an NBIF RAI 2024 award and the NSERC Discovery grant.

References

1. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. Journal of cryptology **26**, 313–339 (2013)
2. Ball, M., Carmer, B., Malkin, T., Rosulek, M., Schimanski, N.: Garbled neural networks are practical. Cryptology ePrint Archive (2019)
3. Banik, S.: Gift-cofb v1.0. Gaithersburg, MD, USA: NIST (2024)
4. Bao, Z.: Photon-beetle authenticated encryption and hash family. NIST (Mar 2021), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/photon-beetle-spec-round2.pdf, accessed: July 3, 2024
5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck lightweight block ciphers. In: Proceedings of the 52nd annual design automation conference. pp. 1–6 (2015)
6. Beierle, C.: Schwaemm and esch: Lightweight authenticated encryption and hashing using the sparkle permutation family. NIST (Mar 2021), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/sparkle-spec-round2.pdf, accessed: July 3, 2024
7. Bernstein, D., Lange, T.: ebacs: Ecrypt benchmarking of cryptographic systems. https://bench.cr.yp.to/ (2024), accessed: September 3, 2024
8. Beyne, T., Chen, Y.L., Dobraunig, C., Mennink, B.: Elephant v1.1. NIST (Mar 2021), accessed: July 3, 2024
9. Bogdanov, A., Knežević, M., Leander: Spongent: A lightweight hash function. In: Cryptographic Hardware and Embedded Systems–CHES 2011: 13th International Workshop, Nara, Japan. pp. 312–325. Springer (2011)

10. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: Present: An ultra-lightweight block cipher. In: Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9. pp. 450–466. Springer (2007)
11. Bonte, C., Vercauteren, F.: Privacy-preserving logistic regression training. BMC medical genomics **11**, 13–21 (2018)
12. Chanal, P.M., Kakkasageri, M.S.: Security and privacy in iot: a survey. Wireless Personal Communications **115**(2), 1667–1693 (2020)
13. Chen, C., Zhou, J., Wang, L., Wu: When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 2652–2662 (2021)
14. Daemen, J., Hoffert, S., Peeters, M., Assche, G.V., Keer, R.V.: Xoodyak, a lightweight cryptographic scheme. NIST (Mar 2021), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/Xoodyak-spec-round2.pdf, accessed: July 3, 2024
15. Dobraunig, C.: Isap v2.0. NIST (Mar 2021), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/isap-spec-round2.pdf, accessed: July 3, 2024
16. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1. 2: Lightweight authenticated encryption and hashing. Journal of Cryptology **34**, 1–42 (2021)
17. Dong, Y., Chen, X., Shen, L., Wang, D.: Privacy-preserving distributed machine learning based on secret sharing. In: Information and Communications Security: 21st International Conference. pp. 684–702. Springer (2020)
18. Ghavamipour, A.R., Turkmen, F., Jiang, X.: Privacy-preserving logistic regression with secret sharing. BMC medical informatics and decision making **22**(1), 89 (2022)
19. Giacomelli, I., Jha, S., Joye, M., Page, C.D., Yoon, K.: Privacy-preserving ridge regression with only linearly-homomorphic encryption. In: Applied Cryptography and Network Security: 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings 16. pp. 243–261. Springer (2018)
20. Gong, Z., Hartel, P., Nikova, S., Tang, S.H., Zhu, B.: Tulp: A family of lightweight message authentication codes for body sensor networks. Journal of computer science and technology **29**, 53–68 (2014)
21. Grover, K., Tople, S., Shinde, S., Bhagwan, R., Ramjee, R.: Privado: Practical and secure dnn inference with enclaves. arXiv preprint arXiv:1810.00602 (2018)
22. Guo, J., Peyrin, T., Poschmann, A.: The photon family of lightweight hash functions. In: Advances in Cryptology–CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings 31. pp. 222–239. Springer (2011)
23. Hell, M., Johansson, T., Meier, W., Sönnerup, J., Yoshida, H.: Grain-128aead—a lightweight aead stream cipher. NIST (Mar 2021), accessed: July 3, 2024
24. Hirose, S., Ideguchi, K., Kuwakado, H., Owada, T., Preneel, B., Yoshida, H.: A lightweight 256-bit hash function for hardware and low-end devices: Lesamnta-lw. In: Information Security and Cryptology-ICISC 2010: 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers 13. pp. 151–168. Springer (2011)

25. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Romulus v1.2. NIST (Mar 2021), https://csrc.nist.gov/CSRC/media/Projects/lightweightcryptography/documents/round-2/spec-doc-rnd2/Romulus-specround2.pdf, accessed: July 3, 2024
26. Jiang, Y., Hamer, J., Wang, C., Jiang, X., Kim, M., Song, Y., Xia, Y., Mohammed, N., Sadat, M.N., Wang, S.: Securelr: Secure logistic regression model via a hybrid cryptographic protocol. IEEE/ACM transactions on computational biology and bioinformatics **16**(1), 113–123 (2018)
27. Kim, M., Song, Y., Wang, S., Xia, Y., Jiang, X., et al.: Secure logistic regression based on homomorphic encryption: Design and evaluation. JMIR medical informatics **6**(2), e8805 (2018)
28. Le Quoc, D., Gregor, F., Singh, J., Fetzer, C.: Sgx-pyspark: Secure distributed data analytics. In: The World Wide Web Conference. pp. 3564–3563 (2019)
29. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New lightweight des variants. In: Fast Software Encryption: 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers 14. pp. 196–210. Springer (2007)
30. Luykx, A., Preneel, B., Tischhauser, E., Yasuda, K.: A mac mode for lightweight block ciphers. In: Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers 23. pp. 43–59. Springer (2016)
31. Mouha, N., Mennink, B., Van Herrewege, A., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: an efficient mac algorithm for 32-bit microcontrollers. In: Selected Areas in Cryptography–SAC 2014: 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers 21. pp. 306–323. Springer (2014)
32. Needham, R.M., Wheeler, D.J.: Tea extensions. Report (Cambridge University, Cambridge, UK, 1997) (1997)
33. NIST: Nist lightweight cryptography. https://github.com/usnistgov/Lightweight-Cryptography-Benchmarking (January 3 2017), accessed: 2024-09-02
34. NIST: Nist lightweight cryptography process timeline. https://csrc.nist.gov/projects/lightweight-cryptography/timeline (January 3 2017), accessed: 2024-09-02
35. Renner, S., Pozzobon, E., Mottok, J.: Lwc benchmark. https://lab.las3.de/gitlab/lwc/compare (2024), gitHub repository. Accessed: September 3, 2024
36. Rivest, R.L.: The rc5 encryption algorithm. In: International Workshop on Fast Software Encryption. pp. 86–96. Springer (1994)
37. Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., Russinovich, M.: Vc3: Trustworthy data analytics in the cloud using sgx. In: 2015 IEEE symposium on security and privacy. pp. 38–54. IEEE (2015)
38. Weatherley, R.: Lightweight cryptography primitives. https://rweather.github.io/lwc-finalists/index.html (2024), accessed: September 3, 2024
39. Wheeler, D.J., Needham, R.M.: Tea, a tiny encryption algorithm. In: Fast Software Encryption: Second International Workshop Leuven, Belgium, December 14–16, 1994 Proceedings 2. pp. 363–366. Springer (1995)
40. Wu, H., Huang, T.: Tinyjambu: A family of lightweight authenticated encryption algorithms. NIST (Mar 2021), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/TinyJAMBU-specround2.pdf, accessed: July 3, 2024