Fidelius: A Novel Secure Data Analysis Framework Leveraging Intel SGX and Blockchain

Chenmin Wang
The University of Aizu
Aizuwakamatsu, Japan
d8211101@u-aizu.ac.jp

Chunhua Su
The University of Aizu
Aizuwakamatsu, Japan
chsu@u-aizu.ac.jp

Zhenxing Hu
YeeZTech
Beijing, China
hzx@pku.edu.cn

Xuepeng Fan
YeeZTech
Beijing, China
xuepeng.fan@yeez.tech

Yulong Zeng
YeeZTech
Beijing, China
zengyulong@yeez.tech

Abstract

As the reliance on data analysis grows in the era of big data, concerns over data leakage and privacy breaches have become increasingly prevalent. While existing technologies such as Secure Multi-Party Computation (MPC), Homomorphic Encryption (HE), Federated Learning (FL), and Trusted Execution Environments (TEE) aim to address these concerns, they often exhibit limitations in addressing complex data analysis scenarios involving multiple roles. In this paper, we propose Fidelius, a novel system that leverages Intel SGX and blockchain to enhance data analysis security. Fidelius employs a static binary analysis approach and a privacy description language (PDL) to prevent data leakage in computation results. Furthermore, it introduces a cryptographic protocol to ensure the trustworthiness and verifiability of computation results, along with a combination of cryptographic protocol and local attestation to achieve consistent verification of analysis programs. Experimental results demonstrate that Fidelius incurs minimal overhead while surpassing existing solutions in performance. Thus, Fidelius presents a promising solution to enhance the security of data analysis in complex scenarios involving multiple roles.

CCS Concepts

• Security and privacy Trusted computing; Privacy-preserving protocols.

Keywords

Data Analysis, Data Privacy, Trusted Execution Environment

BSCI '25, Hanoi, Vietnam

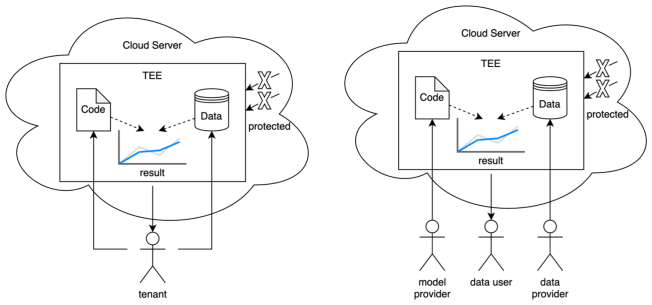
ACM ISBN 979-8-4007-1412-2/2025/08
https://doi.org/10.1145/3709016.3737801

1 Introduction

In the era of big data, people increasingly rely on data analysis for decision-making [?]. To perform data analysis, individuals without their own servers typically delegate data analysis tasks to cloud servers [?]. However, while using cloud servers for data analysis, there are concerns about the potential leakage of this critical data [?], as it often involves personal privacy or business secrets. To address the issue of data leakage, an increasing number of relevant technologies have been proposed. These technologies encompass: Secure Multi-Party Computation (MPC) [? ? ?], Homomorphic Encryption (HE) [? ? ?], Federated Learning (FL) [? ? ?], Trusted Execution Environments (TEE) [? ? ?]. Among these technologies, TEE is versatile, catering to a wide range of general computing scenarios and offers higher computational performance.

However, with the growing complexity of data analysis scenarios involving multiple roles, existing technologies are not readily applicable to effectively address data leakage in these intricate contexts. In existing data analysis scenarios, there are only two roles, as shown in Figure 1(a): tenants and cloud service providers. Existing technologies primarily focus on preventing cloud service providers from leaking tenant data. However, in today's data analysis scenarios, there are multiple roles involved, as shown in Figure 1(b), including data providers, data users, model providers, and cloud service providers. Data must be protected from leakage by all parties outside of the data provider. In addition to this, it is necessary to safeguard the interests of all parties involved in the data analysis scenario. For instance, data users should obtain fair and valid analysis results, while data providers, model providers, and cloud service providers should engage in equitable revenue sharing.

Therefore, when dealing with data analysis scenarios with multiple roles, we need to detect whether data users attempt to leak data by extracting sensitive information from the computed results, whether model providers engage in data leakage through malicious models, whether cloud service providers attempt data leakage by manipulating hypervisors, operating systems, memory, disks, and other means, and whether the computed results generated collectively by



(a) tenant and cloud service provider (b) multiple roles

Figure 1: Data Analysis Scenarios with Two Roles vs. Multiple Roles.

data providers, model providers, and cloud service providers are legitimate.

Many relevant studies have been proposed to address the challenges faced in data analysis scenarios with multiple roles, including SDTE [?], PrivacyGuard [?], SPDS [?], Amanuensis [?] and etc. SDTE [?] introduced an innovative model known as "data processing-as-a-service" in the data trading industry. Leveraging this model, it achieves secure data trading through a series of trading protocols. PrivacyGuard [?] utilizes smart contracts to define data usage policies and leverages TEE for efficient contract execution while safeguarding private data. SPDS [?], much like PrivacyGuard, employs smart contracts to define data usage policies and implements a two-phase delivery protocol to ensure secure release of computing results and payment. Amanuensis [?] explores the intersection of blockchain technology and TEE to address the challenges of data provenance, confidentiality, and user privacy in data sharing.

However, these relevant studies still exhibit certain limitations when addressing these data analysis scenarios. First, these related research efforts do not guarantee the prevention of data leakage within data analysis results. Although these works introduce data usage policies to describe who can access what kinds of data at what price, there is the possibility of malicious behavior in data analysis programs. The final output of the analysis program may include the leakage of sensitive information. An example is when the analysis program directly outputs the raw data, allowing the data analysis results to gain access to the complete, raw data. A straightforward solution is to require the analysis program to be open and accessible for data providers to audit, ensuring that it does not contain code that could lead to data leaks. However, for model providers, their models are valuable assets, and they may be reluctant to make them publicly available. On the other hand, auditing complex analysis programs represents a significant workload, and it is not feasible to require audits for every analysis task, as this would significantly reduce the efficiency of data analysis.

Second, the existing works cannot ensure the trustworthiness and verifiability of computation results. Specifically, when a data user receives computation results, they are unable to confirm that these results indeed stem from running the specified data through the specified model and are not simply randomly generated. Furthermore, data users lack any means to validate the correctness of these results. Without the assurance of both trustworthiness and verifiability of the computation results, there is the potential for deceptive results to be used to mislead data users, jeopardizing their interests.

Finally, in existing works, remote attestation is required for every data analysis task to ensure the consistency of the analysis program. However, related works [? ?] have shown that remote attestation has drawbacks such as inefficiency and dependence on trusted third parties. Taking Intel Software Guard Extensions (SGX) Enhanced Privacy ID (EPID) as an example, a single remote attestation process involves interactions between the Intel Provisioning Service (IPS), Intel Attestation Service (IAS), Intel-signed provisioning enclave (PvE), Intel-signed quoting enclave (QE), and the analysis program enclave. These interactions require data transmission over a wide area network, and the transmitted data must be encrypted to ensure its security. For frequent data analysis tasks, this can lead to significant performance overhead. On the other hand, IPS and IAS are centralized services provided by Intel, making remote attestation dependent on their stable working.

In this paper, we propose Fidelius, a system that leverages Intel SGX and blockchain to enhance data analysis security, addressing the previously mentioned limitations. Intel SGX ensures the security and integrity of the data analysis process, while the blockchain is utilized for trustworthy transmission, storage, and verification.

To address the issue of data leakage in computation results, Fidelius employs a static binary analysis approach [?] to examine whether the analysis program provided by the model adheres to a privacy description language (PDL). In this context, the introduced PDL characterizes the computational rules of data as a finite state machine (FSM), capturing the state transitions from input data to output data. Any state transitions not described in the PDL are considered as violations of the privacy rules.

To ensure the trustworthiness and verifiability of computation results, Fidelius offers a cryptographic protocol. In this protocol, the data user provides a private key, which is securely transmitted to the analysis program's Enclave and used to sign the computation results. Since this private key can only be decrypted within the specified analysis program's Enclave, it remains inaccessible to the data provider, model provider, and cloud service provider. Therefore, if the signed computation results can be successfully verified, it confirms that the results indeed originate from the specified analysis program and are verifiable.

To address the issues of the inefficiency and continued reliance on centralized services in remote attestation, Fidelius

combines a designed cryptographic protocol with local attestation to achieve the consistency verification of analysis programs. Fidelius introduces a Key Management Enclave that obtains authorized keys during initialization. Subsequently, in all data analysis tasks, this authorized key is used to perform local attestation, ensuring the consistency of the analysis program.

The main contributions of this paper are summarized as follows.

First, we introduce the Privacy Description Language (PDL) combined with static binary analysis to rigorously enforce data confidentiality and prevent sensitive data leakage in computation outcomes.
Second, we design a cryptographic protocol to ensure the trustworthiness and verifiability of computation results.
we integrate the cryptographic protocol with a local attestation mechanism to ensure the integrity and correctness of analysis programs executed within protected environments.
Finally, we evaluate the performance of Fidelius, and the experimental results demonstrate that it incurs minimal overhead, contributing less than 2% to the data analysis system, while surpassing existing solutions by more than 30 times.

2 Background

2.1 Intel SGX

Intel Software Guard Extensions (SGX) is a hardware-level trusted execution environment that safeguards program execution. It establishes a secure, private memory region, called Enclave Page Cache (EPC), safeguarding its content from unauthorized access or modification by external processes. This designated memory space is considered trusted, while any region beyond it is deemed untrusted, strictly restricting access from untrusted environments.

A program deployed within the EPC is referred to as an enclave program. Enclaves are designed to operate in isolation from untrusted environments and can only interact through explicitly declared methods, known as ECALL or OCALL methods.

We will introduce the key methods and operations that are encapsulated within an enclave.

sgx_get_key(): This function retrieves a symmetric key that is generated based on the hash of the current enclave, CPU ID, and other configuration files. Each enclave has a unique symmetric key generated in this manner. The symmetric key remains inaccessible from outside the enclave unless specific exporting (OCALL) methods are declared.
*sgx_ecc*256*_create_key_pair*(): This function generates an ECC-256 asymmetric key pair by utilizing SGX's self-contained random number generator.

To securely store private messages from an enclave in local storage, the "seal" operation employs the enclave's symmetric key to encrypt the message and then stores it locally. This approach ensures that the plaintext of the sealed message remains inaccessible to users, as the symmetric key remains undisclosed. To unseal the message, users transmit the message to the enclave using a designated (ECALL) method, and the enclave decrypts the message using its own symmetric key.

2.2 Remote Attestation

Remote attestation is a technology provided by Intel SGX to prove the integrity of enclave programs running on an unknown platform. Currently, Intel SGX provides two types of remote attestation: Intel Enhanced Privacy ID (Intel EPID) Attestation and Elliptic Curve Digital Signature Algorithm (ECDSA) Attestation.

Remote attestation based on EPID typically involves the following steps. The application enclave initiates a request to the local quoting enclave (QE) provided by Intel SGX, generating a quote for the application enclave. These two enclaves use local attestation for request initiation and quote transmission, wherein local attestation establishes a trusted channel and enables data transfer between two enclaves on the same platform. After receiving the quote, the application enclave sends it to the remote Intel Attestation Service (IAS) for verification and obtains the verification result. EPID has several limitations, such as requiring that the network where the application enclave is located can connect to the Intel Attestation Service. It also demands that the Intel Attestation Service is highly available at all times, meaning there are no single points of failure or downtime.

ECDSA-based Attestation was introduced to address the limitations associated with EPID. With Intel SGX Data Center Attestation Primitives (DCAP), ECDSA-based attestation enables providers to establish and offer a third-party attestation service, eliminating the need to rely on Intel's provided remote attestation service. Intel SGX DCAP allows third parties to provide a quoting enclave and generate quotes for application enclaves. The attestation key used by the third-party quoting enclave can be generated internally within the data center. The public key of the attestation key is sent to Intel's Provisioning Certification Enclave (PCE) and signed/authorized. After generating the quote, it is sent to the data center's internal attestation service for verification.

2.3 Blockchain

2.3.1 Permissionless Chain and Permissioned Chain. Blockchain can be categorized into permissionless chains and permissioned chains. In permissionless chains, accessibility is open to everyone, and nodes can freely join or leave without any permission control. Due to the limited restrictions in permissionless chains, achieving consensus among all nodes can be relatively challenging, resulting in lower system throughput.

Permissionless chains initially found application in cross-border payments and later evolved into the foundational technology framework for decentralized finance.

Permissioned chains consist of multiple members, which are typically comprised of enterprises, institutions, government entities, and similar organizations. Permissioned chains serve as trusted third parties, offering characteristics of openness and transparency. They are primarily used for public data notarization, traceability, verification. Due to the smaller number of members in permissioned chains, efficient consensus algorithms can be employed to facilitate consensus among them, thereby endowing permissioned chains with higher throughput performance.

In Fidelius, the utilization of permissioned blockchain serves as a reliable and fault-tolerant third-party entity for tasks like data transmission, data storage. In contrast, traditional mediators are susceptible to single point failures, data tampering risks, and the high costs associated with establishing P2P private networks.

2.3.2 Smart Contract. Blockchain's smart contracts are on-chain programs known for their transparency in code and execution processes. In Fidelius, smart contracts play a pivotal role in verifying the signature of the analysis results provided by cloud service providers. The success of a data analysis is assured once the verification is successfully completed, instilling trust in all parties involved.

3 Threat Model and Design Choice

In this section, we first introduce the roles within the system, then detail the threat model, highlighting potential risks and discussing various types of attacks. Additionally, we establish key assumptions that form the foundation for the design and implementation of Fidelius. Subsequently, we delve into the design choices made for Fidelius, aimed at mitigating the identified risks and countering potential attacks on the system.

3.1 Roles

Data Provider (DP). Data provider, as the sole owner of the raw data, initially publishes metadata on the blockchain, which includes the hash and a necessary description of the raw data. The accuracy of this metadata is validated by the data provider's credibility, exemplified by a history of successful data analyses.
Model Provider (MP). Model provider supplies analysis programs for specific types of data.
Cloud Service Provider (CSP). Cloud computing provider supplies the computational resources required for data analysis and additionally provides a Trusted Execution Environment to ensure secure data analysis. After receiving a request for a data analysis task, the cloud computing provider is obligated to execute the analysis program and return the results to the blockchain.
Data User (DU). Data user selects the desired raw data by examining the metadata on the blockchain and initiates a data analysis to obtain the results from the specified analysis program.
Blockchain. Blockchain serves as a reliable, failure-resistant third party for data transmission and storage. Specifically, a smart contract verifies the correctness of the signature on the analysis results.

3.2 Threat Model

The system comprises four distinct roles: data provider, model provider, cloud service provider, and data user, which are inherently distrustful of one another. During the data analysis process, the following attacks are prevalent:

Data Theft Attacks: Cloud service provider may steal data through the hardware or software resources they provide; data user may log into the server to steal data. The program provided by the model provider may contain malicious code aimed at stealing raw or intermediate data.
Data Fabrication Attacks: The data provided by the data provider is inconsistent with what is claimed.
Data Misuse Attacks: Data from the data provider is used across different models and cloud servers without proper authorization.
Result Fabrication Attacks: The results of the data analysis do not accurately reflect the true execution of the model.
Result Theft Attacks: The results of data analysis are illicitly accessed by the data provider, model provider, cloud service provider, or an unidentified attacker.

We do not address side-channel attacks on Intel SGX in this paper. We assume Intel SGX's hardware functionality is as advertised, ensuring that code within an enclave remains unaltered and the values of internal variables are protected from direct memory access. Notably, our reliance on Intel is limited to a one-time interaction during the setup process; no further contact with Intel servers or DCAP is necessary. In contrast, previous approaches rely on remote attestation, requiring ongoing integrity and availability of Intel servers or DCAP for each data analysis task.

We assume the presence of a collision-resistant hash function and secure signature and encryption schemes. Specifically, our signatures incorporate a nonce to ensure that a valid signature pair $(msg, signed_{msg})$ cannot be generated by an adversary without the private key, rendering historical signatures invalid.

3.3 Design Choice

To counter the identified attacks, we have made the following design choices.

Defending Against Data Theft Attacks: All data located in non-trusted environments, such as network transmissions or the non-TEE components of cloud servers, are encrypted to prevent direct data theft by data users or cloud service providers. Additionally, the

analysis programs provided by model providers undergo static code analysis prior to data analysis to prevent data theft through the analysis results.
Defending Against Data Fabrication Attacks: Prior to conducting data analysis, the data hashes are verified to ensure they match the claimed hashes.
Defending Against Data Misuse Attacks: Digital signature technology is implemented, enabling the data provider to sign the platform, model, and model parameters using a private key. The validity of this signature is verified before executing data analysis.
Defending Against Result Fabrication Attacks: The data user's private key is securely transmitted to the Trusted Execution Environment (TEE) on the cloud server and used to sign the results. When the data user receives the results, they also receive the signature made by the private key. If the signature is validated, it confirms that the results were indeed obtained from the TEE, ensuring the results have not been fabricated. This approach is analogous to zero-knowledge proof.
Defending Against Result Theft Attacks: The results of data analysis are encrypted with the data user's public key, ensuring that only the data user can access them.

4 Design

4.1 Overview

This section provides a brief overview of the enclaves and notations used in Fidelius, along with its architecture and workflow.

4.1.1 Enclaves and Notations. Throughout this paper, we define the following enclaves and notations:

EKeyMgr. Enclave Key Manager manages asymmetric keys, handling creation, deletion and provides fundamental cryptographic functions, including message encryption, decryption, signing, and signature verification.
EAnalyzer. Enclave Analyzer, an analysis program in Intel SGX Enclave format, ensures the program remains unaltered during execution.
(PK_{DP}, SK_{DP}), an asymmetric key pair of data provider.
(PK_{DU}, SK_{DU}), an asymmetric key pair of data user.
(PK_{CSP}, SK_{CSP}) represents the asymmetric key pair of cloud service provider, generated by EKeyMgr. The private key, SK_{CSP}, is securely stored within the enclave, ensuring that it cannot be extracted by the cloud service provider.
$H()$ denotes the hash function.
$Enc(PK, msg)$ denotes the encryption of msg using PK.
$Dec(SK, cipher)$ denotes the decryption of $cipher$ using SK.
$Sign(SK, msg)$ denotes the signing of msg with SK.
$Verf(PK, msg, sig)$ denotes the verification of sig using PK and msg.
(SK, PK_{CSP}, H_{EA}) represents the process of forwarding SK to EKeyMgr using PK_{CSP}. Within this context, SK is utilized in EAnalyzer, identified by H_{EA}. The function () involves $Sign(SK, concat(PK_{CSP}, H_{EA}))$ and $Enc(PK_{CSP}, SK)$.

4.1.2 Architecture and Workflow. The Fidelius architecture and data analysis workflow are illustrated in Figure ??. The process begins with a setup phase where the EKeyMgr in CSP generates an asymmetric key pair. The public key, PK_{CSP}, undergoes verification via Intel's remote attestation service, while the private key, SK_{CSP}, remains securely stored within EKeyMgr. Concurrently, the Data Provider (DP) and Data User (DU) each generate their own asymmetric key pairs, which they retain. Additionally, the DP encrypts data using her public key, PK_{DP}, and prepares this encrypted data for the DU's use.

The DP uploads encrypted data to the CSP and forwards her private key, SK_{DP}, to EKeyMgr using $(SK_{DP}, PK_{CSP}, H_{EA})$. Similarly, the DU uploads the analysis program to the CSP and forwards her private key, SK_{DU}, to EKeyMgr using the same method. It is important to note that the plaintext private keys (SK_{DP} and SK_{DU}) can only be decrypted within EKeyMgr.

Once the analysis program, encrypted data, and private key are prepared, the data analysis task begins. The analysis program is first checked against the rules defined by the Privacy Description Language (PDL). Any program that fails this check results in an immediate termination of the analysis task. Subsequently, EAnalyzer loads and decrypts the encrypted data. It then verifies the hash of the decrypted data; if it does not match the claimed hash, EAnalyzer halts immediately to prevent processing altered or fraudulent data. If the data is verified, EAnalyzer proceeds with the main analysis, ultimately producing an encrypted result using PK_{DU}. Additionally, the result is signed using SK_{DU}. Since SK_{DU} resides within EKeyMgr, EAnalyzer establishes a secure channel via local attestation to request SK_{DU}.

4.2 Privacy Description Language

To prevent data leakage from analysis results, we have developed a Privacy Description Language (PDL) that defines rules for data usage. Static binary analysis is used to ensure that the analysis program adheres to these PDL-defined rules. As illustrated in Figure ??, the code snippet uses PDL to specify the rules for applying the KMeans algorithm on the Iris dataset. This code is then translated into LLVM's intermediate representation (IR), followed by symbolic execution [? ?] to derive the state S as described by PDL. Concurrently, we utilize GTIRB [?] for intermediate representation in our analysis. After converting the analysis program to this format, symbolic execution is applied to obtain the state S of each output variable, ensuring that S is a subset of S, thereby confirming compliance.

4.3 Trustworthy and Verifiable Results

Algorithm ?? outlines the primary operations of EKeyMgr and EAnalyzer. The encrypted analysis result and its signature, generated within EAnalyzer, are transmitted to the blockchain. There, anyone can verify the signature's validity. A valid signature confirms that the DP successfully executed the analysis program and achieved the correct result, since the signature originates from within EAnalyzer. This mechanism prevents attackers from forging a valid signature without executing the analysis program, thereby ensuring the integrity of the data analysis process. The DU can download the encrypted result from the blockchain and decrypt it using SK_{DU} to obtain the plaintext analysis result.

4.4 One-time Remote Attestation

We incorporate a one-time remote attestation process, as detailed in Algorithmă??, to establish a secure and trusted environment from the outset. During the setup phase, the Cloud Service Provider (CSP) validates the authorization of the public key PK_{CSP} generated by EKeyMgr by performing a remote attestation with Intels attestation service. This thorough verification confirms both the integrity and legitimacy of the CSPs credentials. Once verified, subsequent analysis tasks on the CSP no longer require remote attestation. The private keys (SK) forwarded by the DP and DU are securely transmitted from EKeyMgr to EAnalyzer via local attestation.

5 Evaluation

In this section, we assess Fidelius's performance through comprehensive experiments. Initially, we gauge its time consumption by executing CPU-intensive and I/O-intensive tasks separately. Subsequently, we conduct a performance comparison between Fidelius and alternative solutions that execute analysis programs in the Ethereum Virtual Machine (EVM). Our experiments are conducted on a machine equipped with an Intel(R) Core(TM) i5-10400F CPU boasting 12 cores at 2.9 GHz, 16 GBytes of RAM, and a 12 MByte cache. Unless otherwise stated, each experiment is repeated 1,000 times to calculate the average value.

5.1 CPU-Intensive Task

To evaluate Fidelius's performance on CPU-intensive tasks, we implement a neural network algorithm with three layers and 128 hidden units, designed to recognize handwritten digits using the MNIST dataset [?]. We report both the time consumption and accuracy of the algorithm. Additionally, we deploy the algorithm on raw CPU (without SGX) for comparison. The key differences between the Fidelius-based algorithm and the raw CPU-based algorithm include:

> Random library. The Fidelius-based algorithm utilizes a random library based on Intel SGX, whereas the raw CPU-based algorithm relies on the C++ random library.
>
> Data decryption. Fidelius requires decryption of sealed data before initiating the algorithm, incurring additional processing time. Conversely, the raw CPU-based algorithm can directly access plaintext data.

Both for Fidelius and raw CPU, we maintain a fixed number of test sets at 1,000 while incrementally increasing the number of training sets from 10,000 to 50,000. For each training set, we execute the algorithm with epochs set to 100 and 200, respectively.

Figures 2(a) and 2(b) illustrate the time consumption of the algorithms running on Fidelius and raw CPU with varying epochs. As the number of training sets increases, the time consumption of the algorithms on both Fidelius and raw CPU shows a linear growth trend. However, the time consumption difference between these two algorithms is minimal. Specifically, it is observed that the execution time on Fidelius is only 2% longer than that of raw CPU, primarily due to data decryption overhead.

Figures 2(c) and 2(d) show the accuracy of algorithms on Fidelius and raw CPU over different epochs, with both achieving over 91% accuracy. At 100 epochs, accuracy fluctuates slightly, but with no marked difference between the platforms. At 200 epochs, both algorithms converge with nearly identical accuracies. Fidelius shows a marginal 2% increase in time consumption compared to raw CPU but maintains similar accuracy levels, demonstrating comparable performance in CPU-intensive tasks.

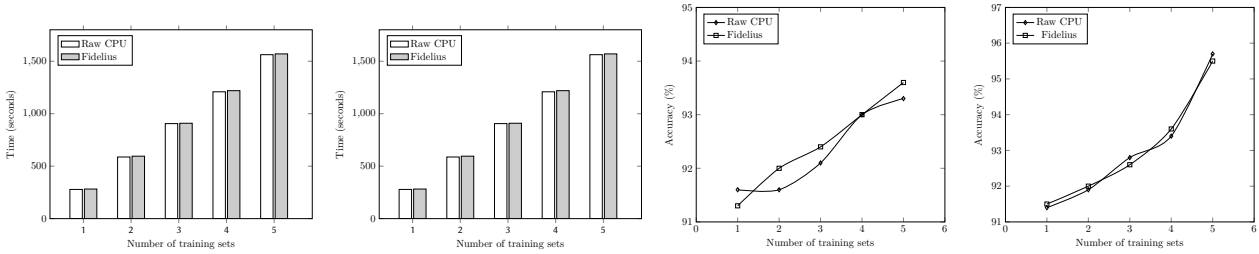
5.2 I/O-Intensive Task

To evaluate the time efficiency of Fidelius in handling I/O-intensive tasks, we tested an algorithm that searches for a target string within a file over 1 GByte on both Fidelius and raw CPU platforms. Time consumption depends on data loading and string searching, with additional time required on Fidelius due to data decryption. Given Intel SGX's constraints in Fidelius, we set data loading block sizes to 64 KBytes and 256 KBytes and varied the number of data lines from 1 to 128 and 1 to 256 for each block size to measure time efficiency.

Figure 3 shows the time efficiency of Fidelius compared to raw CPU across different data block sizes. Fidelius shows a significant reduction in time consumption as the number of lines read increases. For instance, with a block size of 64 KBytes, the time taken by Fidelius drops from 81 seconds (for 1 line) to 11 seconds (for 128 lines). Similarly, with a block size of 256 KBytes, it reduces from 213 seconds (for 1 line) to 10 seconds (for 256 lines). Increasing the block size to 256 KBytes further optimizes Fidelius's performance. Despite improvements, a gap remains in time consumption between Fidelius and raw CPU, largely due to the decryption process in I/O-intensive tasks.

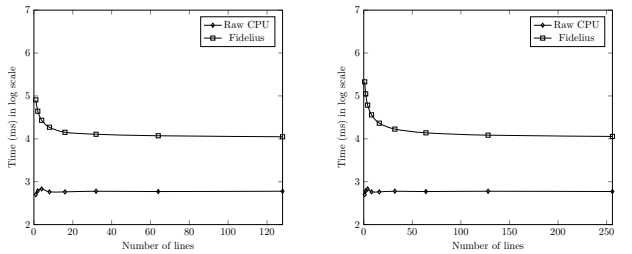
5.3 Performance Comparison

In this section, we compare the performance of Fidelius with SDTE [?], which uses a k-nearest neighbors (k-NN) algorithm as an Ethereum smart contract to execute machine



(a) epoch = 100 (b) epoch = 200 (c) epoch = 100 (d) epoch = 200

Figure 2: Time Consumption and Accuracy of CPU-Intensive Task.



(a) Data block size = 64 KBytes (b) Data block size = 256 KBytes

Figure 3: Time Consumption of I/O-Intensive Task.

learning tasks on the Ethereum Virtual Machine (EVM). We implement the k-NN algorithm on both Fidelius and raw CPU to evaluate each solution's time consumption, using the Titanic [?] dataset from Kaggle as input.

To evaluate the k-NN algorithm's execution time on the Ethereum Virtual Machine (EVM), we deployed the k-NN smart contract on Ganache, a personal Ethereum blockchain. The reported time consumption for k-NN on the EVM focuses solely on the execution time, excluding any time spent on transaction broadcasting or committing. We also exclude the time required to upload the Titanic dataset to the blockchain, as the k-NN contract uses on-chain data. Due to EVM's memory limits, the k-NN algorithm cannot handle more than 60 test sets, hence we limit test sets to 10-60 while keeping training sets at 100.

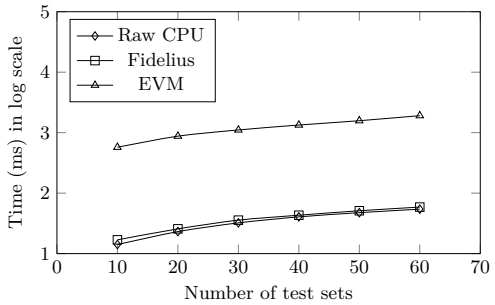


Figure 4: Time Consumption on Raw CPU, Fidelius and EVM.

Figure 4 shows the time consumption of the k-NN algorithm on raw CPU, Fidelius, and the Ethereum Virtual Machine (EVM), with the latter taking roughly 30 times longer than the others. Additionally, EVM executions often fail when testing more than 60 sets due to memory constraints, highlighting the difficulties of running complex machine learning algorithms with large datasets on the EVM, as also noted in previous studies about Ethereum's memory limits [?]. In contrast, Fidelius provides a more reliable and stable environment for sophisticated machine learning tasks.

6 Related Work

GXS [?] operates as a blockchain-based data trading platform that records and facilitates transactions between buyers and sellers, who exchange data via private channels upon mutual agreement. AccountTrade [?] uses blockchain to pair analyzers with data providers and ensures ecosystem security through enforceable protocols and data indices that detect and penalize dishonest behaviors. Zhao et al. [?] developed a data trading system focused on provider privacy, employing ring signatures and double authentication to secure transactions and prevent unauthorized access.

In data-sharing systems [? ? ?], Shen et al. [?] develop a scheme that ensures data integrity and confidentiality. Zuo et al. [?] introduce a system that efficiently safeguards and revokes sellers' secret keys using proxy re-encryption and key separation, enhancing data protection with attribute-based encryption. To mitigate malicious proxy involvement in data leaks, Guo et al. [?] present accountable proxy re-encryption (APRE), a framework that detects and addresses misuse of re-encryption keys, further validating its CPA security and accountability under the DBDH assumption. Deng et al. [?] formalize an identity-based encryption transformation (IBET) model for sharing encrypted data with additional recipients beyond the initial designation by the data owner.

SDTE [?] introduces a blockchain-based data analysis platform where data providers encrypt and upload data to the blockchain. Data users select this data, form analysis contracts, and request services. Upon approval, providers send decryption keys to a trusted node via Intel SGX remote

attestation, which then decrypts the data and runs the analysis in the Ethereum Virtual Machine (EVM) protected by Intel SGX, before uploading the results to a settlement contract. However, SDTE's encrypted data uploads exert significant storage demands on the blockchain, and running analysis as Ethereum smart contracts introduces performance challenges. Running complex algorithms like k-NN on the EVM, as shown in Section 5.3, is often impractical due to the scale of data and complexity of tasks.

PrivacyGuard [?] mitigates the performance limitations of the Ethereum Virtual Machine (EVM) by transitioning on-chain analysis to off-chain Trusted Execution Environments (TEEs), allowing data providers to set policies that limit unauthorized data usage. However, it still encounters challenges with EVM performance. Similarly, Sterling [?] introduces a decentralized data market utilizing machine learning for data analysis but also places significant storage demands on the blockchain, like SDTE and PrivacyGuard. Moreover, privacy concerns or regulations may prevent the upload of sensitive data.

7 Conclusion

Fidelius is a novel system that enhances data analysis security in complex, multi-role scenarios. Leveraging Intel SGX and blockchain technology, it tackles data leakage and guarantees trustworthy, verifiable computation outcomes. By employing static binary analysis and a privacy description language (PDL), Fidelius secures results while its cryptographic protocol ensures integrity. Local attestation consistently verifies analysis programs, reducing reliance on centralized services and boosting efficiency. Experiments show that Fidelius incurs minimal overhead while outperforming existing solutions, offering a robust approach to protecting sensitive data in diverse environments.