On the Tracking of Sensitive Data and Confidential Executions

José Roberto Nascimento Jr, José B. S. Nunes, Eduardo Lucena Falcão, Lilia Sampaio, Andrey Brito
{roberto,benardi,eduardolfalcao,liliars,andrey}@lsd.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba, Brazil

ABSTRACT

The production of large amounts of sensitive data raises growing concerns on confidentiality guarantees. Considering this, it is natural that data owners have an interest in how their data are being used. In this work, we propose Data aNd Application Tracking (DNAT), a trustworthy platform for tracking the executions of applications over sensitive data in untrusted environments. For traceability purposes, we use blockchain and smart contracts, and to guarantee execution confidentiality and, especially, enforce that operations are appropriately logged in the blockchain, we use Intel SGX. Experiments show that tracking costs on Ethereum varies from 1 to 61 cents of a US dollar, depending on the operation and urgency for consolidation. The time cost of confidential execution is associated with the SGX overhead. It increases non-linearly initially but has a linear growth rate when data and application size gets much higher than the available enclave page cache (≈ 93 MB).

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; **Privacy-preserving protocols**.

KEYWORDS

data tracking, sensitive data, confidential execution, blockchain, trusted execution technologies

ACM Reference Format:
José Roberto Nascimento Jr, José B. S. Nunes, Eduardo Lucena Falcão, Lilia Sampaio, Andrey Brito. 2020. On the Tracking of Sensitive Data and Confidential Executions. In *The 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20), July 13–17, 2020, Virtual Event, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3401025.3404097

1 INTRODUCTION

The high-tech world we live in is transforming conventional people into data producers. Mobile applications, IoT devices such as smartwatches and other wearables are continuously collecting and processing data to ease our daily tasks. The data we produce is also valuable for big data and advertisement companies, which may use our preferences and opinions to influence individuals of a given profile into taking some action that may draw them some advantage [12]. In addition to its value, some of these pieces of information are sensitive and should be kept confidential. However, throughout the last decade, there were several major cases of data leakage.

Among the most famous data leakage scandals are Facebook's [22] and Uber's [21], where personal data from 57 million Uber's accounts and from 87 million Facebook's users were breached. The forces of market impose a natural punishment to these companies: Uber's valuation dropped from $68B billion to $48 billion, and $100 billion was knocked off from Facebook's valuation. However, as data breaches are not new or a temporary issue, the European community led a movement towards the legal regulation of these cases.

Among the goals of the General Data Protection Regulation (GDPR) is removing ambiguity from common laws that were not primarily thought to address data leakage cases. It clarifies its scope, quantifies the penalties, specifies that terms of consent must be intelligible and easily accessible, defines that data can be erased anytime, and clears out that data subjects may have access to which purposes their information is being used and by whom [5]. In other words, the access right to personal data means tracking which application (and consequently which entity) is consuming specific data. Furthermore, personal data is not the only type of sensitive data; from another perspective, applications considered strategic for some companies could also be regarded as sensitive data.

With these issues in mind, we propose the **Data aNd Application Tracking** (or **DNAT**, for short), a reliable platform for traceable executions of applications over data. The main difference from DNAT's strategy to standard data policies practiced by some companies is that the actions performed on the former can be audited in a transparent and tamper-proof fashion. Such a feature is an outcome of the combined utilization of the blockchain with trusted executions environments, technologies that can attest the integrity of log records, input data, and application.

Assets that retain intellectual property (applications) or sensitive information (data) typically need to be confidential due to their inherent value. When required, the integrity and confidentiality of applications can be guaranteed by *Trusted Execution Environments* (TEE) — an environment that provides guarantees that application execution will not deviate from the expected behavior. Nevertheless, it is not trivial to ensure that an application for which the code is unknown will not leak data during its processing.

If data is confidential, it becomes necessary to ensure that the application is not malicious. If application confidentiality is not a critical issue, such verification could be carried out by the data owner or mediated by some trusted third party with access to the source code. This third party then checks if the application complies with privacy and data management standards. Then, DNAT ensures that application integrity was not compromised before usage.

DEBS '20, July 13–17, 2020, Virtual Event, QC, Canada

ACM ISBN 978-1-4503-8028-7/20/07…$15.00
https://doi.org/10.1145/3401025.3404097

When data is not sensitive, and thus, privacy is not a concern, confidential applications (i.e., applications whose source code can not be publicized) could be used without integrity verification but still require logging. Finally, if the application is confidential, and data is sensitive, mitigation can be performed by limiting what can be done inside the trusted environment. For instance, if the purpose of a confidential application is to process sensitive data in a cloud environment to produce results for the data owner, preventing communications would suffice to protect data confidentiality even with unknown code. In this work, we focus on the first and second cases, where either the application or data confidentiality is critical.

We instantiate our platform with a TEE from Intel, the Software Guard eXtensions (SGX) [13]. SGX is endowed with protected memory areas named enclaves, where application and data reside during execution. Enclaves can protect data and code against disclosure and modification by non-authorized third parties, and even from high privileged software such as the operating system and hypervisors. In order to provide confidentiality during storage, data and application are stored encrypted, and the decryption keys are sent to a trusted Key Management Service.

Data traceability is addressed by using blockchain and smart contracts. The main feature of a blockchain is that every data recorded is considered to be immutable due to the computational requirements to subvert the chain. The smart contracts are the interface of the blockchain and, in DNAT, they are used to grant access right to a given piece of data from a specific application. This access is granted by recording a tuple $< application, data, requester >$ on the blockchain, and such a registry is used to retrieve the decryption keys. Note that these decryption keys are only provided to the TEE after an attestation of the code loaded in the enclaves.

To show the applicability and feasibility of DNAT, we performed experiments in scenarios within the health care context. We consider as use case the application of widely known machine learning (ML) models such as logistic regression, support vector machine, and random forest. The chosen application was applied to a pair of open health care datasets [8, 18] and a sensitive one [14]. However, the DNAT platform could also be applied in contexts other than health care, contexts in which confidential execution is desired due to the sensitivity of the data and typical privacy approaches, which transforms and discards information, are not adequate.

The experiments show that tracking and confidentiality are achieved at some cost. Because tracking information is recorded on the blockchain, the DNAT user will pay a fee to have the operation consolidated. In our experiments, this fee varied from less than 1 cent to 61 cents on the dollar, depending on the operation and on the desired time of consolidation. When it comes to the confidential execution, in addition to the computing resource costs of the ML training, there is a time cost associated with the TEE overhead and proportional to the workload size. Even with the costs described above, some applications will deem worthful to pay such values to be executed confidentially and to restrict access to sensitive data to a protected environment and in an auditable fashion.

The rest of this paper is organized as follows. The next section describes the theoretical background necessary to understand the fundamental aspects of the DNAT platform and presents the threat model. Section 3 presents the architecture and provides an overall description of each component. The workflow of the operations available to end-users is described in Section 4. A security analysis is provided in Section 5. In Section 6 we present the experiment design. Evaluation results and their corresponding analysis are presented in Section 7. The most related works are discussed in Section 8. Finally, in Section 9, we highlight the main conclusions.

2 BACKGROUND

2.1 Attestation and Secure Containers

To ensure integrity and confidentiality, Intel SGX offers local and remote attestation features [2], which can be used by a third part to ensure that the expected chunk of code is being executed inside an enclave, on an authentic SGX-capable server. Nevertheless, using SGX is not a straightforward task. Some limitations, such as the inability to directly issue system calls, make the learning curve steep. In this work, we use SCONE [3], a Secure Container Environment that uses SGX to protect containerized processes. SCONE simplifies the process of porting applications to SGX and also offers tools for integration with Docker and Kubernetes.

To use SCONE in a remote environment, a client enables the creation of configuration files that spawn containers and prepare the environment for secure communication with them. During container startup, sensitive configuration, such as credentials, are securely passed to the application (injected into the application enclave either as environment variables or command line parameters). The passing of confidential information is done with the help of the Configuration and Attestation Service (CAS). The CAS is an SGX-based component that stores the confidential information and releases this data only to parties that have been attested, which has proved that the code running is the expected one. In a scenario with an application running in a remote environment, the CAS can run both as a service local to the infrastructure (e.g., within a cloud provider) or as an external service, on the users' premises. The connection between the CAS and the running applications is done through a second service, the Local Attestation Service (LAS), which runs on every host.

2.2 Blockchain and Smart Contracts

In this work, we use Ethereum [9], a blockchain-based platform that supports smart contracts. Contracts are persisted in the blockchain and thus are comprised of a set of immutable and publicly auditable functions. Transactions performed by any Ethereum user can trigger these functions, and more importantly, they can change the state of the blockchain by appending information on it.

It is crucial to notice that there is a cost associated with transactions handled by a contract. The user firing a smart contract transaction must pay for some miner to execute it. This price is a reward for its expenses with memory, storage, and energy. Therefore, each transaction must carry the *gas price* and *start gas* fields. Gas price is the fee to pay the miner per computational step, and thus it affects the time it takes to have a transaction included in a block on the chain since miners typically prioritize transactions yielding higher revenues. Start gas is the total gas that the user firing such contract execution is willing to pay. The total gas is necessary to avoid infinite loops, possibly caused by bugs, from spending all the funds of an externally owned account.

Most of the blockchain applications try offloading from the blockchain by keeping on-chain only metadata and the essential and mandatory computation. In addition, if one is interested just in examining the contents of the blockchain, she can interact with an Ethereum Provider, without smart contract mediation. Providers are nodes containing all the blockchain (full node) or at least the headers of the blocks (light nodes). Lightweight nodes are, in some cases, sufficient due to their capability of checking the existence of an event by using a Bloom filter (BF) data structure. Since a BF is a cheap form of storing information, we also use such structure to diminish the storage costs in the smart contract of DNAT.

2.3 Threat Model

We assume an adversary that controls the machines that runs DNAT. They could, for instance, install arbitrary software. The goal of the attacker is to steal encryption keys or data that is being processed in the machines. Nevertheless, we assume that all Intel SGX basic operations are bug-free. This assumption also includes the toolset provided by the SCONE platform used in our system. In addition, due to Intel SGX's attestation process, code has its integrity guaranteed by an SHA256 hash. We also assume that the attestation process cannot be circumvented.

Side-channel attacks are also not in Intel's threat model. To address this issue we could use an approach similar to the one proposed by Oleksenko et al. (2018), which has been validated also with the SCONE runtime. Roughly, the idea resides on the fact that page table and L1/L2 cache-timing side-channel attacks typically rely on an abnormally high rate of asynchronous enclave exits or adversary-controlled sibling hyperthreads. In this case, occupying the hyperthreads with trusted threads (e.g., sibling threads for the same application or even to a helper enclave thread that is created simply to occupy the hyperthread) and using a monitor for enclave exit rates has been shown to be effective against L1/L2 and page table attacks. As Varys is not yet enabled in release versions of SCONE, our current experiments do not consider this protection, but its future addition would not require any change in the architecture or application. In the meantime disabling hyperthreading should reduce the attack surface.

With respect to the blockchain, logs recorded on it are considered immutable due to the difficulty of creating a longer Ethereum chain. To avoid obtaining corrupt information from the blockchain, we assume that Providers used in DNAT are a trustworthy interface to the ledger. Providers can be authenticated by their TLS certificates. In addition, attacks to compromise the availability of the blockchain, such as distributed denial of service attacks, are deemed ineffective due to the high level of distribution of the blockchain.

3 ARCHITECTURE

In this Section, we describe the proposed architecture for DNAT, which is illustrated in Figure 1.

An asset owner o_i, where i identifies the owner, willing to register an asset $\alpha_{i,h}$ with content hash h, be it a dataset $\delta_{i,h}$ or an application $\pi_{i,h}$, is required to provide a manifest $\mu_{i,h}$. The manifest $\mu_{i,h}$ is a text document that will be signed by o_i, and describes the properties and usage of $\alpha_{i,h}$. For $\delta_{i,h}$, $\mu_{i,h}$ may also contain a set of applications $\mathcal{P} \mid \pi_{j,h'}, j \neq i \wedge h' \neq h$, that could consume

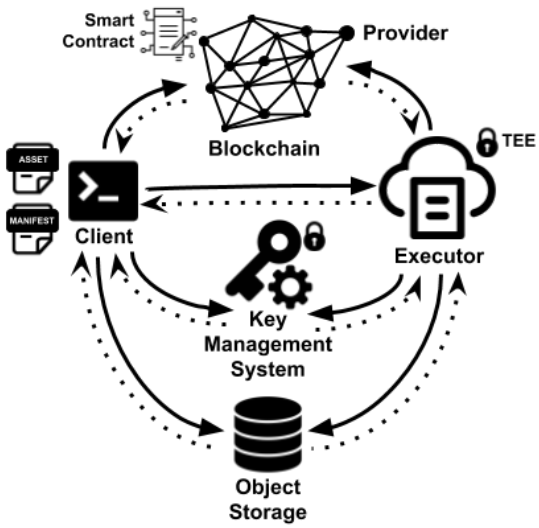

Figure 1: Architecture of the platform.

$\delta_{i,h}$ without compensation, but with logging. We name this list the asset's *whitelist*.

The *Client* component is the interface for interacting with the platform. It allows asset registration, listing and tracking, acquisition of access rights, request of application execution, and revocation of assets' availability.

To enforce tracking, we leverage the *Blockchain* component, instantiated with Ethereum. We take advantage of its decentralized nature, immutable log records for execution requests, and compensation of transactions. Furthermore, the blockchain stores the DNAT *Smart Contract*, which provides the interface for the operations that can change the state of the ledger: registration, acquisition of access rights, and asset revocation. These operations are consolidated by publishing (extra) information about the assets on the blockchain.

The *Provider* is the entry point for all operations performed in the blockchain. The Provider may manage its own Ethereum node or forward requests and transactions to a public node. In DNAT, o_i specifies on $\mu_{i,h}$ her trusted Provider(s) to check compensation before allowing the usage of $\alpha_{i,h}$; execution only proceeds upon consensus between the Providers from the dataset and application.

The *Object Storage* is responsible for storing the objects of assets, which are identified by their hash. In this work, we consider the Interplanetary File System (IPFS) as the Object Storage. Later, the hashes are used to recover data from IPFS nodes when execution is requested. Also, due to the sensitive nature of the data, it is recommended to encrypt the data before sending it to IPFS nodes. Therefore, even if an attacker gets a hold of the hash that identifies the data on IPFS, the attacker does not have the appropriate key to decrypt the retrieved content.

From the trusted execution point of view, we propose the *Executor*. This component is responsible for executing $\pi_{i,h}$ over $\delta_{j,h'}$, where i may be equal or different from j and $h \neq h'$. When a user demands an execution, transactions that will eventually persist an

event on the blockchain are triggered. The Executor, then, receives this event containing the information about that execution, including the address of the user requesting it, and the hashes of the assets to be used. As seen in Figure 1, the Executor is an SGX machine. Thus, it has the proper capabilities to perform secure processing of sensitive data, guaranteeing its integrity and confidentiality.

The Executor also inspects the ledger history to check whether the permissions of execution are properly recorded. This component runs inside an enclave using the SCONE framework. Therefore, the trusted part of the Executor is the core SGX element of the proposed solution. It consists of a module executing inside an enclave which is responsible for decrypting the data recovered from IPFS and adequately executing the assets.

Currently, DNAT supports applications written in the Python programming language. To make use of the SGX capabilities of the Executor, a special third-party interpreter is used, *scone-python* [19], which allows for the execution of Python code inside SGX enclaves. When executing *scone-python*, the Executor component should be attested in order to obtain the secrets entrusted to it by the asset's owner, and then, it becomes able to execute the application with the corresponding dataset properly.

We also assume there is a service external to this framework acting as a Key Management System, which here we instantiate with the *SCONE CAS* technology. CAS is a component of the SCONE framework responsible for storing secrets and handing it back to the querying code if it passes attestation. This means that if a service communicates with the CAS to ask for a secret, the service only passes on the information if the one requesting it can be trusted (i.e., has the expected `MREnclave` and runs with a permissioned CPU in the adequate execution mode). CAS can be a service running in a cloud provider (supporting the SCONE framework), or a service maintained by a third-party (e.g., the asset owner himself). In summary, the CAS associates an application identifier and an `MREnclave` to a set of secrets.

4 WORKFLOW OF OPERATIONS

The DNAT platform provides six operations: i) asset registration; ii) acquisition of assets' access rights; iii) application execution; and iv) assets' catalog; v) asset's usage tracking; vi) asset's publication revocation. The workflow of all these operations is described next.

4.1 Asset Registration

The workflow of the asset register feature is illustrated in Figure 2.

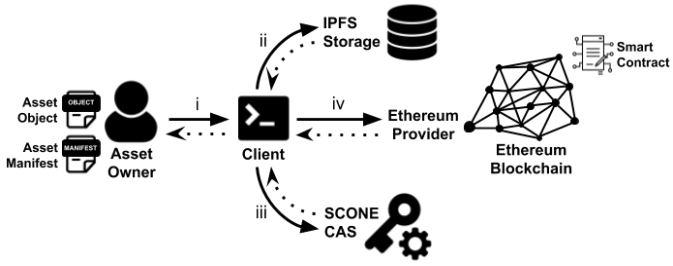

Figure 2: Asset registration operation.

Before submitting an asset registration transaction to the smart contract, some operations must be done. First, the Client component generates a symmetric key $\kappa_{i,h}$ and uses it to encrypt $\alpha_{i,h}$. For the key, i refers to the user, as before, and h to the hash of the asset that was encrypted with that key. Assume the function $\epsilon(\alpha_{i,h}, \kappa_{i,h})$ returns the encrypted asset $\alpha_{i,h}$ with key $\kappa_{i,h}$. Then, $\epsilon(\alpha_{i,h}, \kappa_{i,h})$ is sent to the IPFS Storage (operation ii), and $\kappa_{i,h}$ along with additional information about trusted Executors are sent to the SCONE CAS (operation iii). The response given by IPFS is the hash of $\epsilon(\alpha_{i,h}, \kappa_{i,h})$, which is used as a unique identification of it. Thus, the hash of $\epsilon(\alpha_{i,h}, \kappa_{i,h})$ may be used to obtain information about $\alpha_{i,h}$ in DNAT.

The register transaction submitted to the smart contract (operation iv) requires a few arguments. Due to the sensitiveness or intellectual property of some assets, the owner must define a compensation value, in DNAT measured in Wei (the smallest unit of Ethereum cryptocurrency). Furthermore, for $\delta_{i,h}$, free permission can be granted to specific applications. For instance, it could be in the interest of the data owner (o_i) to allow open-source applications to compute quality metrics, such as statistical parity difference, that in turn could be used by arbitrary users to comprehend better the properties of $\delta_{i,h}$ (before actually acquiring it). In this case, o_i must provide the hashes of these permissioned applications to identify them and grant free access to $\delta_{i,h}$. Due to the blockchain storage cost, these hashes are stored in a 256-bit BF structure in the smart contract. The content of the BF bit array is generated at the Client side to decrease the amount of computation performed over the smart contract.

Another valuable property is the asset integrity. The register transaction also carries an SHA-256 hash of $\alpha_{i,h}$ (other than the hash of $\epsilon(\alpha_{i,h}, \kappa_{i,h})$), which is used for integrity verification of $\alpha_{i,h}$ before execution. All the information about $\alpha_{i,h}$ is described by the manifest $\mu_{i,h}$ generated by the Client with all the information provided by o_i. The manifest is publicly available in the IPFS (operation ii) so that potential consumers can check asset's information before the acquisition of rights. The hash of $\mu_{i,h}$ is sent to the smart contract and, as the hash of $\epsilon(\alpha_{i,h}, \kappa_{i,h})$ is used to retrieve $\epsilon(\alpha_{i,h}, \kappa_{i,h})$ from the IPFS, the hash of $\mu_{i,h}$ is used to retrieve $\mu_{i,h}$.

In summary, the transaction sent to the smart contract contains the following parameters: asset name, value, hash of $\epsilon(\alpha_{i,h}, \kappa_{i,h})$, hash of $\mu_{i,h}$, hash of $\alpha_{i,h}$, asset type (application/data), and the pre-computed BF bit array for the whitelist.

4.2 Acquiring Assets' Access Rights

The acquisition of assets' access rights is depicted in Figure 3.

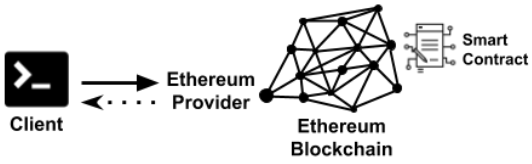

Figure 3: Acquisition, catalog, tracking, and revocation workflow.

To acquire the rights to execute $\pi_{i,h}$ over $\delta_{j,h'}$, where i may be equal or different from j and $h \neq h'$, a final user has to utilize the Client to send a transaction to an Ethereum Provider, which is the entry point of smart contracts. The user needs to provide the hash of $\epsilon(\pi_{i,h}, \kappa_{i,h})$, the hash of $\epsilon(\delta_{j,h'}, \kappa_{j,h'})$, its Ethereum credentials, and a sufficient amount of Wei to compensate o_i and o_j. Once the transaction is mined, i.e., selected by miners to be processed and included in the Ethereum Blockchain, the final user could request any cloud provider supporting the DNAT Executor component to run $\pi_{i,h}$ over $\delta_{j,h'}$. Also, it is recommended that the final user pins the acquired assets, even though it is encrypted, in an own IPFS node, in order to avoid its deletion after compensating o_i and o_j.

In DNAT, some applications are allowed to use some datasets without compensation. However, even in this case, it is necessary to explicitly persist the access permission on the blockchain, for tracking purposes. In this sort of transaction, a flag indicates if the final user intends to acquire the access right to $\delta_{i,h}$ without compensation. If this flag is set, then the smart contract verifies if the BF of $\delta_{i,h}$ contains the desired application ($\pi_{j,h'}$) by means of hashing, and in positive case, it persists the tuple $< \delta_{i,h}, \pi_{j,h'}, final\_user, free >$ on the event log history. Finally, note that the BF may yield false positives, returning that $\pi_{j,h'}$ could consume $\delta_{i,h}$ for free when the $\pi_{j,h'}$ does not have this free permission. This problem is solved within the application execution workflow.

4.3 Application Execution

The steps involved in the execution of an acquired application over an acquired dataset are depicted by Figure 4.

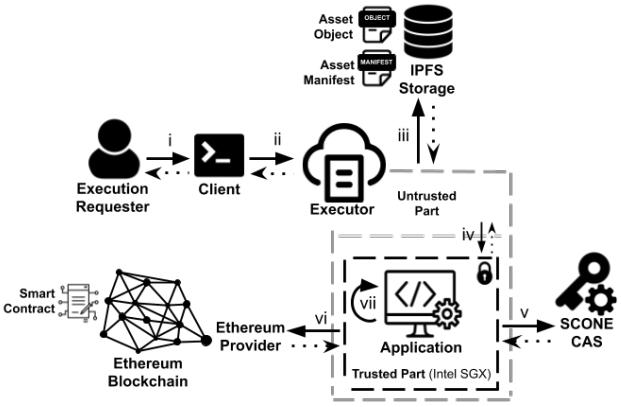

Figure 4: Application execution workflow.

The first task is performed in the untrusted part of the Executor component, and it consists in retrieving the $\epsilon(\delta_{i,h}, \kappa_{i,h})$ and $\epsilon(\pi_{j,h'}, \kappa_{j,h'})$ from the IPFS (operation iii). Then, the trusted part of the Executor component, the one running inside the SGX enclave, is launched twice, because the secrets of different assets can not be handed simmultaneously to the same SCONE session. In each of these two executions, the CAS delivers the secrets of each asset along with a fresh CAS generated key, which is specific to $\delta_{i,h}$ and $\pi_{j,h'}$. Let us call such a key $\kappa_{ij,hh'}$. These secrets, i.e., $\kappa_{i,h}$, $\kappa_{j,h'}$ and $\kappa_{ij,hh'}$, are only released upon CAS attestation of the Executor component. Once both assets have been re-encrypted with $\kappa_{ij,hh'}$, the Executor is launched once again, and this time it will be able to run $\epsilon(\pi_{j,h'}, \kappa_{ij,hh'})$ over $\epsilon(\delta_{i,h}, \kappa_{ij,hh'})$ because, after CAS attestation, it will be granted access to $\kappa_{ij,hh'}$.

Application execution proceeds as follows. The trusted part of the Executor searches for the specific tuple $< \delta_{i,h}, \pi_{j,h'}, final\_user >$ in the Ethereum blockchain (operation vi). This request is directed to a public trusted Ethereum Provider node such as Infura [6]. Note that the tuple is recorded on the blockchain event log history. The Executor component only proceeds with the application execution if the proper tuple is persisted on the ledger.

There is a particular case where the application's rights can be granted for free, and this is explicitly logged in the event history. On the smart contract side, this is done with the aid of a BF. However, the BF may yield false positives (either intentionally or not). The false positive would, for instance, allow an attacker o_j to consume $\delta_{i,h}$ for free by inserting a proper nonce in $\pi_{j,h'}$. To avoid such an undesired situation, if the access rights' were obtained by special free permission, it is necessary to run an extra validation step before actually running $\pi_{j,h'}$. The mitigation consists in verifying whether the hash of the application (h') is in the permissioned application's list located in $\mu_{i,h}$. Because $\mu_{i,h}$ resides in the IPFS storage, it is immutable and trustworthy.

Before application execution, assets' integrity is checked through the comparison of their content SHA-256 hash with those recorded in the blockchain. This integrity check is required to make sure the data stored by the untrusted part of the Executor itself was not tampered with. Finally, the trusted part executes the application with the corresponding dataset. The results are then sent back to the Client and, therefore, to the final user.

4.4 Assets' Catalog, Track, and Revocation

The workflow of cataloging, tracking, and revoking operations are similar to the acquisition of asset's access rights, and thus, they can also be illustrated by Figure 3.

Before acquiring the access rights' of an asset, a user first needs to decide which application and dataset satisfy her necessities. This decision is facilitated through the catalog operation, which returns the assets with their associated manifest containing a detailed description of its features. To perform such operation, the user has to inform the Client how many blocks she wants to search for the registered assets in the blockchain. This argument has a direct impact on the response time, as presented in the evaluation section. If no argument is provided, then the whole blockchain is scanned.

At any moment, the asset owners can track the usage of their data and application. Likewise, the catalog operation, tracking requires the range of blocks considered for the search. The following information comprises a track response: application executed, input data, acquisition timestamp, and ID of execution requester. Both catalog and track are read-only operations performed by searching for event signatures on the log history.

Finally, whenever an asset owner is no longer interested in keeping her asset available, she can revoke it from DNAT. Note, however, that users who have already acquired the access rights for some asset will still retain the right to use it. On the other hand, new access rights' acquisition is blocked. In contrast to catalog and track operations, revoking changes the state of the smart contract.

5 SECURITY ANALYSIS

A possible successful attack is commonly known as "data escape". For instance, a confidential application could simply return the input data as response. However, this type of attack can happen to possibly any system running confidential applications if we assume that, as external parties cannot see the application, there is no guarantee about what the code being executed is doing, consequently making it possible for data to be returned. Therefore, we assume that our system handles only one sensitive asset at a time: either the application or data is confidential.

A critical step to assure DNAT's security is in the deployment of the Executor. In this deployment, the asymmetric key used in the re-encryption process must be securely generated. Otherwise, the infrastructure provider would be able to decrypt the assets after re-encryption. This could happen because, between the re-encryption phase and the execution phase, the assets would be persisted on disk and the infrastructure provider, in possession of the private key, would be able to decrypt such assets. Therefore, in order to enforce a secure re-encryption, the infrastructure provider must request a key pair to the KMS during the Executor deployment. Since the KMS is an SGX-based component, the infrastructure provider would only have access to the public key; the private key would be delivered to the trusted part of the Executor after integrity attestation of the application. This master key can then be used to generate other one-time keys to be used in the re-encryption process.

The false positives yielded by queries to Bloom filter do not pose security issues to DNAT because the Executor component does an extra verification by examining the manifest of the dataset. Nevertheless, choosing appropriate parameters for the Bloom filter is a good practice as it results in less noise (i.e., invalid transactions) for audit operations. Furthermore, an attacker would have to pay for miners to process their transactions for each attack attempt.

6 EVALUATION

6.1 Experiments and Environment

Some relatively well-known ML algorithms, such as logistic regression (LR), support vector machine (SVM), and random forest (RF), were used to train models over sensitive data. Since the training was performed without actually requiring a developer to inspect the raw data, such a use case evidences that the DNAT platform can be applied in a privacy-preserving fashion. Moreover, while traditional privacy techniques distort the data in ways that may compromise the results of training, our platform preserves the original data. We also evaluate the performance of the platform giving a glance at the cost of each component of DNAT, either on-chain or off-chain, to show its performance.

Three data sources are used in the evaluation process. The first one, the Cleveland heart dataset [8], is composed of cardiac exams done on 303 patients, where the features are values from such as maximum heart rate achieved, and the target variable is the diagnosis of heart disease. The second one, comprised of 462 exams, is the South Africa heart dataset [18]. It is a retrospective sample of males in a heart-disease high-risk region of the Western Cape, South Africa. The last one, the Medical Information Mart for Intensive Care III (or MIMIC3, for short) [14], contains data on the admission of tens of thousands of patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012.

To represent a very low load, but still based on real data, we leveraged Cleveland's and South Africa's, both with a file size of approximately 18 KB. To evaluate the platform with different factors of higher loads, we created sub-samples of MIMIC3, with a pre-processed size of 1.7 GB. In the sub-sampling process, we selected the most recent exams, considering the last six months (50578 exams, 8.2 MB total), the last year (370294 exams, 61 MB total), the last two years (1188297 exams, 195 MB total), and the last four years (3079828 exams, 504 MB total).

To process the experiments, we make use of distinct virtual machines (VM) to host the Client, the SCONE CAS, the IPFS storage, and the Executor component. The setup of these VMs is described in Table 1. Furthermore, in order to draw realistic results, we use the Ethereum Ropsten test net to host the DNAT's blockchain component. It uses a proof of work consensus similar to the one used in Ethereum's main net, has the benefit of being free, and, in addition, enables the measurement of the amount of gas spent in each smart contract operation.

Component	vCPU	RAM	Storage
Client	1	2GB	20GB
SCONE CAS	2	2GB	20GB
IPFS	2	4GB	20GB
Executor	4	20GB	20GB

Table 1: Configuration of VMs for each DNAT component.

The experiment design is divided into two categories: off-chain and on-chain evaluations. In this work, any operation changing the state of the blockchain is considered to be on-chain. Thus, in the off-chain assessment, we measure the impact of the Client, IPFS, and Executor components on the overall time of DNAT's operations. In the on-chain evaluation, we discuss the average financial cost of DNAT's smart contract functions: register, acquire and revoke. Moreover, we also discuss the time costs to have a transaction securely persisted on the blockchain.

6.1.1 Off-chain assessment. Since all DNAT's operations have at least a portion of computation performed off-chain, we start the evaluation by drawing a baseline measurement of such operations.

The time cost of the off-chain part of the *register operation* is mainly caused by some preliminary computation performed over the asset file. Thus, we instrumented the core data operation costs (reading, encryption, hashing, and storage). For this experiment, we tried out the 6-months sample and the 4-years sample generated from the MIMIC3 workload, to assess a small and a large workload.

Following a similar approach, we also evaluated the costs of the operations to acquire an asset's access right and to revoke an asset. This assessment was done by measuring the latency of the Client while performing these operations.

With respect to the blockchain, *catalog* and *track* are read-only operations. Their time costs are affected by the number of blocks inspected (*eth_getLogs*) and by the number of calls (*eth_call*) performed. Whenever some information about an asset is queried to

the blockchain, a call is performed. As the number of found assets grows, we expect an increase in the number of calls. For this reason, we measured the latency of the number of calls performed and the impact of searching in different quantities of blocks.

To gauge the *application execution* performance, we first applied the ML models over low load data, the datasets of Cleveland and South Africa. In this first set of experiments, the amount of Enclave Page Cache (EPC) memory for the Executor VM ranged from 16 MB to 90 MB, so that we could draw some conclusions on the impact of EPC size over the application's execution time. Moreover, we also ran all the applications without the SGX TEE. Then, to investigate a more representative setting, we tried out heavier workloads, the 6-months, 1-year, 2-years, and 4-years sub-samples of MIMIC3, with a VM containing 90 MB of EPC. In this work, datasets with a size close to or greater than the highest possible EPC size (90 MB) are considered heavy because it will impose memory pagination. Note that the application size also contributes to the occurrence of pagination since it also resides within the enclave.

For ML training, which is the focus of our evaluation, each of these datasets is stored as different assets. We believe that having the information of each patient comprised of different assets would not be realistic since data operators are hospitals or scientific organizations. Moreover, having the information of all patients wrapped in a single asset favours the performance of DNAT's operations because there will be a single "data operation" (reading, encryption, hashing, storage, and appending to the blockchain) for each asset. However, in the context of classification, the final user is typically a single patient that would like to utilize the model trained to obtain some information. In this case, it is natural that information regarding a patient is stored in a single asset.

6.1.2 On-chain assessment. *Register*, *acquire* and *revoke* are operations that change the state of the blockchain. Here we assess the amount of gas for each operation, their corresponding financial costs, and the time to be persisted on the blockchain.

7 RESULTS AND DISCUSSIONS

Figure 5 presents the latency of the register operation, repeated 30 times for different datasets, the 6-months and 4-years sub-samples from MIMIC3, detailing the total time and the time regarding its core data operations (reading, encryption, hashing, and storage).

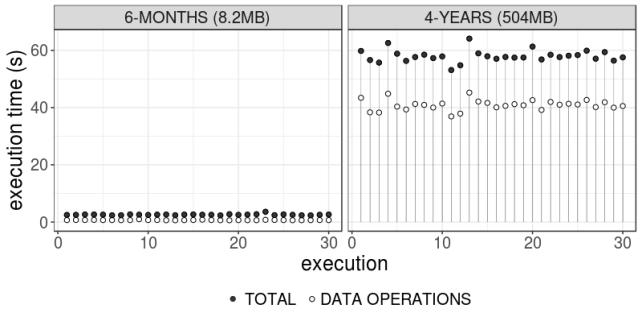

Figure 5: Latency for 30 executions of the register operation.

As expected, the execution time increases as the workload size increases. Also, it can be observed that core data operations like reading, encrypting, hashing, and storing, typically consumes more time as the file size increases.

The acquisition of asset's access right and revoke operations include little computation off-chain, and thus their running times are proportional to the payload of the request sent to the smart contract. The median time of acquiring and revoking, for 30 replications, is 2.84 and 1.6 seconds, respectively. However, this is not the time it takes to have the information persisted on the ledger, because it also depends on the gas price set for the transaction and on the average time for including a block on the chain. At the time of this writing, November 2019, according to [10], the average time it takes to include a block on the Ethereum main net chain was approximately 13 seconds. Also, according to [7], on December 3rd, 2019, to have a transaction persisted faster on the ledger, i.e., within 2 minutes, it is recommended to set the gas price to ten gwei, while with one gwei a transaction would be recorded on the chain within 30 minutes. Note, however, that transactions with higher gas price can, in extraordinary situations, take more time to be included on the chain since there is no deterministic mandatory procedure that the miners should follow to select transactions.

The catalog and track operations are mainly affected by the number of blocks examined and by the number of calls performed on each block, and thus their execution times are computed as the sum of these two latencies. Figure 6 shows the median latency for scanning different number of blocks, and Figure 7 presents the latency when performing different number of calls. Both experiments were executed with 30 repetitions. As a reference, 186606 blocks were mined in the Ethereum main network in June 2019.

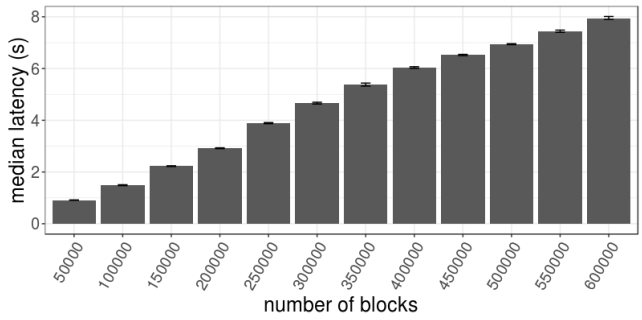

Figure 6: Median latency of 30 event queries on blockchain for different number of blocks.

The first conclusion we can draw from Figures 6 and 7 is that the latency of searching for events in the blockchain grows almost linearly with the number of blocks and calls. Therefore, the response time for the catalog operation may vary according to the number of assets retrieved. If there were, for instance, 1000 assets to be listed, and if for each asset five read operations were performed, then, assuming 100 calls lasts approximately 31 seconds, it would take approximately 25 minutes. Although this time is considerable, cloud or infrastructure providers that support DNAT could cache and index such data, eliminating the latency for such queries. Finally, if these assets were searched within 2 million blocks, which would

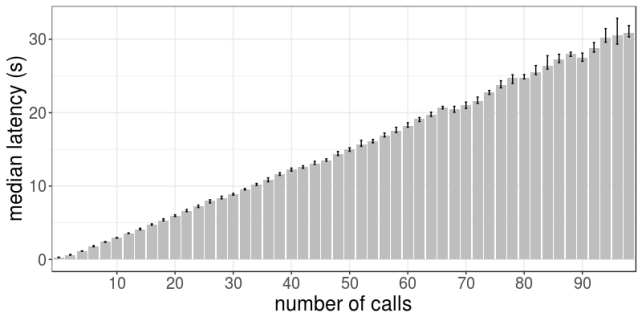

Figure 7: Median latency of 30 queries for asset information on blockchain for different number of calls.

mean approximately the last 300 days (assuming a block time of 13 seconds [10]), then, according to the results presented on Figure 6, an extra 32 seconds (approximately) would be summed to this time.

The last off-chain operation is the execution of the application. Figure 8 presents the execution time for different ML models (logistic regression, random forest, and support vector machine), with different EPC sizes, and over the heart datasets of Cleveland and South Africa. A confidence interval with a confidence level of 95% is also plot over the median of 30 executions. The overhead the platform adds is presented in Figure 9, for the LR model and using the Cleveland and South Africa heart datasets.

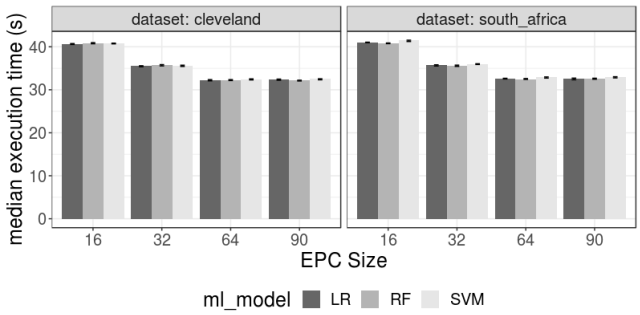

Figure 8: Median execution time of different ML models over Cleveland and South Africa heart datasets. The durations include the overhead of the platform.

From Figure 8, we conclude that the EPC size has some impact, as increasing the EPC size decreases the median execution time. This reduction is attributed to a decrease in the intensity of memory page swap operations, copying data from the enclave to the main memory. Note that this process requires encrypting and decrypting in order to protect data confidentiality [20]. However, we also noticed that there is no significant difference from the execution with EPC size equal to 64 MB and 90 MB since there is an overlap on their confidence interval, and we believe that the reason is that for this small workload, 64 MB of EPC is sufficient to avoid page swapping. Furthermore, one can observe that for applications running such small workloads, the overhead added by the platform due to re-encrypting assets with a common key, environment attestation, and access rights verification, is not marginal.

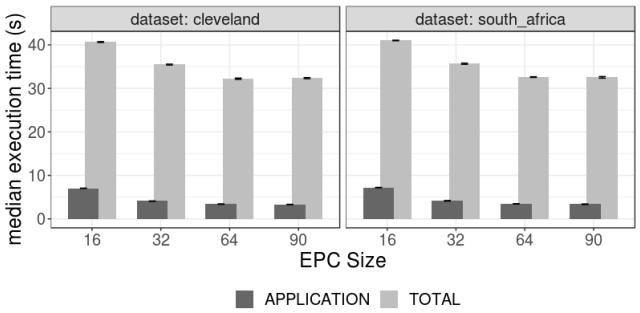

Figure 9: Median execution time for LR over Cleveland and South Africa heart datasets. "Application" represents the ML training time, and "total" stands for the time of training summed with the overhead of the platform.

We also experimented execution of heavier workloads. Table 2 presents the confidence interval (95% confidence level) over the median of the execution time for 30 repetitions, using the MIMIC3 sub-samples and the logistic regression model. We ran this model both as a standalone application (without SGX) and within DNAT, in order to comprehend the overhead added by additional operations performed by the platform. The overhead ratio reflects how much slower is the application execution within DNAT. Finally, Figure 10 presents the execution time of the application (within the SGX TEE) and the total time DNAT takes to complete the execution workflow.

MIMIC3 Range	App	DNAT	Ratio
Six months (8.2 MB)	[1.8, 1.9]	[194.4, 195.3]	102.9
One year (61 MB)	[10.1, 10.2]	[661.3, 665.9]	65.1
Two years (195 MB)	[29.5, 29.7]	[1564.9,1575.1]	52.9
Four years (504 MB)	[78.9, 80.4]	[3941.2, 3968.9]	49.9

Table 2: Confidence interval for the median time of application execution (LR) over MIMIC3, as a standalone application (ML training without SGX), and within DNAT.

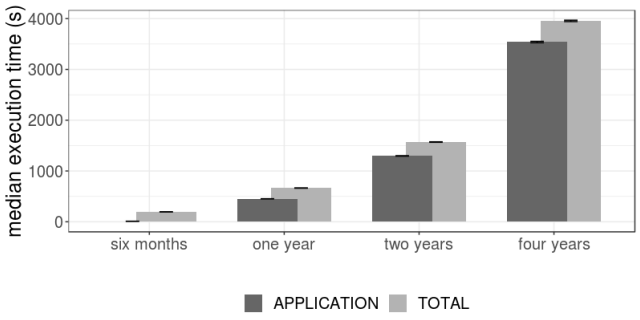

Figure 10: Median execution time of LR over MIMIC3.

From Table 2, it can be observed that, when comparing the conventional (without SGX) and the DNAT executions, the overhead

ratio increases from 49.9 to 102.9. From Figure 10, we conclude that for the small workload (6-months), the overhead added by the basic operations of the Executor (environment attestation, asset transfer, and decryption) is not marginal. However, as workload size increases, the basic operations of the Executor become less impacting, and the SGX TEE becomes the main reason of execution latency. Nevertheless, we have to consider that training a model over some data is not a task performed frequently. It could be performed monthly, or even at longer time intervals, as the ones considered in the samples. For instance, 3.25 minutes is negligible if we consider this application would run once every six months. Such scenarios evidence DNAT's practicality. In addition, as depicted in Figure 10, which complements Figure 9, the overhead introduced by the DNAT platform is negligible with larger workloads.

We conclude the evaluation with the financial costs of on-chain operations. Here we rely on information provided by [7] on June 14th, 2019, when a fast transaction (persisted in less than 2 minutes) required on average ten gwei, and a low safe transaction (persisted in less than 30 minutes) required on average one gwei. Table 3 presents the on-chain costs of contract creation, asset registration, acquisition of asset's access right (with and without Bloom filter verification), and revocation of asset's availability on the platform.

Operation	Gas Usage	Cost (USD)	
		Safe Low	Fast
Contract Creation	1800437	$0.466	$4.660
Asset Registration	236517	$0.061	$0.612
Asset Acquisition	47455	$0.012	$0.123
Asset Acquisition (BF)	41150	$0.011	$0.107
Asset Revocation	14923	$0.004	$0.039

Table 3: On-chain operation costs with ether price on June 14th, 2019.

From Table 3, it is possible to note that the asset acquisition and asset revocation operations can be performed at an approximate cost of one cent of a dollar, considering a safe low-cost transaction. To be persisted faster, in 10 seconds, twelve cents of a dollar would suffice. Note that the cost of acquisition with BF verification is a bit cheaper than the cost of asset acquisition without BF verification because the latter changes the state of the smart contract. The asset registration demands 6.1 cents and 61 cents of a dollar, for the low and fast approaches respectively. However, it is essential to highlight that an asset requires a single registration, and depending on its price, when a single user acquires the access rights to use it, this value may be returned. This is also the case for contract creation. Although it has a high cost to be persisted, a single contract is enough to mediate the access and track the usage of multiple assets, from multiple owners, by multiple final users.

8 RELATED WORK

ProvChain [15] is an architecture to track cloud data operations by embedding provenance data into blockchain. For privacy purposes, user id, as well as data operations, are hashed before being stored on the provenance database. The operation history is hashed into Merkle tree nodes, and the root node is anchored to a blockchain transaction. Thus, if the provenance database was tampered with, then the hash of all operations history will not match the root node recorded in the blockchain. ProvChain performance was measured within the OwnCloud application. Hooks and listeners were inserted for every user operation over files, and such events fire provenance data generation. The provenance service adds an average time overhead of 6.5%, but no financial evaluation is provided.

SmartProvenance [17] and Controllable Blockchain Data Management model (CBDM) [23] use the Ethereum blockchain to record provenance data of documents. They share a similar goal of avoiding unauthorized document modification. Thus, all the changes must pass a validation process in order to be persisted. SmartProvenance applies off-chain verification scripts before actually accepting changes and logging them to the blockchain. In both SmartProvenance and CBDM, every provenance change has to be approved through a voting process performed with the aid of a smart contract.

Ali et al. [1] propose a secure data provenance framework for cloud-centric Internet of Things through blockchain smart contracts. The blockchain is used to store the device metadata concerning its identity, the collected data, and periodic traffic profiles. In the proposed architecture, a gateway component is in charge of hashing the data and persisting such value in the blockchain, a strategy applied by most of the works due to the high storage costs of the blockchain. Likewise ProvChain's strategy, only the root of a Merkle tree is stored on the blockchain, the data itself is kept in cloud storage. If a third party desires to consume the uploaded data, it must provide its public key to the device provenance contract, an approach required by DNAT's Key Management Service. The contract ensures device identity provenance of application services using the data stored in the cloud, which is a very similar goal of DNAT, except that DNAT guarantees confidential execution of applications over data.

In the topic of trusted executions, including blockchain solutions, there is the work of Brandenburger et al. [4]. They propose a combination of the Hyperledger Fabric blockchain with Intel SGX in order to run smart contracts confidentially. The similarity between DNAT and their solution is that input data is kept secret. However, in their approach, the smart contract is the application itself, and thus its results are persisted on the blockchain, while in DNAT results are persisted on external storage such as IPFS.

There are also some commercial initiatives whose goals are partially aligned to DNAT's, such as Enigma and iExec. Zyskind et al. presented Enigma [24] to circumvent blockchain's lack of scalability and confidential computation. They argue that pieces of the smart contract could be offloaded from the blockchain. This enhances scalability and allows privacy-enforcing computation, mitigating the slow and public nature of blockchain. Smart contracts are programmed in a high-level code that allows specifying which parts run publicly, on-chain, and which should run privately, off-chain. To guarantee the confidentiality of data, Enigma relies on secure multi-party computation, a technique that splits data and provides such unreadable pieces of data to different processing nodes.

iExec [11] started as a blockchain-based outsource of opportunistic desktop grid resources. Ethereum and its smart contracts were regarded as a form of managing distributed resources, creating a resource marketplace. Fedak et al. conceived the proof-of-contribution protocol, designed to promote trust between final users

and workers (compute resource providers). The protocol discourages workers from delivering wrong results seeking compensation by employing a majority voting process on the (hashed) result, in such a way that these malicious workers are financially penalized. One of the similarities between DNAT and iExec is the possibility of publication of datasets and applications.

Despite all the similarities between DNAT, Enigma and iExec, the main goal of DNAT is application usage and data traceability, while Enigma and iExec share only the common goal of confidential execution. Further, neither Enigma nor iExec provides official documents detailing the architecture regarding the application of SGX TEE's and its performance. However, we speculate that their approach could be similar to DNAT's, in which data and applications are only decrypted inside a TEE.

9 CONCLUSIONS

This work presents DNAT, a platform for tracking confidential executions of applications over data in a distributed system. To provide tracking, we use the blockchain technology, leveraging its decentralized nature, immutable log records, and support for compensation on the usage of valuable assets. Smart contracts are used to store information about the operations that can change the state of the ledger, like asset registration, acquisition of asset's access rights, and assets' provision revocation. Furthermore, DNAT offers other functionalities such as the proper execution of the application over a specific dataset. At this step, another contribution of this work relies on the use of TEE to provide confidential execution of applications. Our architecture uses Intel SGX in one of its components to make sure executions happen inside protected memory areas, and secrets are only delivered to trusted attested parties.

This work also conducts an evaluation set of tasks to show how DNAT performs with an illustrative health care application. The experiments showed that, when considering confidential executions, besides the regular costs regarding computing resources, there is a time cost linked to the overhead caused by the TEE approach, which can be considered proportional to the workload size.

Finally, contrasting DNAT to state of the art, related works propose solutions using TEE but do not focus on the tracking of assets' executions as done here. Future steps for this research might include different implementations for some components of the architecture, e.g., to support other TEE technologies. Furthermore, we also consider deepening investigations on how to improve the performance of confidential execution of applications.

ACKNOWLEDGMENT

This research was partially funded by the EU-BRA ATMOSPHERE project (EC and MCTIC/RNP, 4th Coordinated Call, H2020 Grant agreement no. 777154) and by Dell-EMC Brazil.

REFERENCES

[1] S. Ali, G. Wang, M. Z. A. Bhuiyan, and H. Jiang. 2018. Secure Data Provenance in Cloud-Centric Internet of Things via Blockchain Smart Contracts. In *2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation*. 991–998. https://doi.org/10.1109/SmartWorld.2018.00175

[2] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *HASP '13: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, Vol. 13. ACM.

[3] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, David Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, 689–703.

[4] Marcus Brandenburger, Christian Cachin, Rüdiger Kapitza, and Alessandro Sorniotti. 2018. Blockchain and Trusted Computing: Problems, Pitfalls, and a Solution for Hyperledger Fabric. *CoRR* abs/1805.08541 (2018). http://arxiv.org/abs/1805.08541

[5] European Commission. 2018. EU General Data Protection Regulation. http://eugdpr.org/.

[6] Consensys. 2019. Infura - Scalable Blockchain Infrastructure. http://infura.io/.

[7] Concourse Open Construction. 2019. Eth Gas Station. http://ethgasstation.info/.

[8] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[9] Ethereum. 2019. Ethereum: A blockchain app platform. http://www.ethereum.org/.

[10] Ethereum. 2019. Etherscan: Block Explorer and Analytics Platform for Ethereum. http://etherscan.io/chart/blocktime.

[11] Gilles Fedak, Haiwu He, Mircea Moca, Wassim Bendella, and Eduardo Alves. 2018. *iExec: Blockchain-Based Decentralized Cloud Computing*. Technical Report. 40 pages. http://iex.ec/wp-content/uploads/pdf/iExec-WPv3.0-English.pdf

[12] Varun Grover, Roger H.L. Chiang, Ting-Peng Liang, and Dongsong Zhang. 2018. Creating Strategic Business Value from Big Data Analytics: A Research Framework. *Journal of Management Information Systems* 35, 2 (2018), 388–423. arXiv:http://doi.org/10.1080/07421222.2018.1451951 http://doi.org/10.1080/07421222.2018.1451951

[13] Intel. 2015. Intel Software Guard Extensions. Cryptology ePrint Archive, Report 2016/086. http://software.intel.com/sites/default/.

[14] Alistair E. W. Johnson, Tom J. Pollard, Lu Shen, Li-wei H. Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. 2016. MIMICIII, a freely accessible critical care database. *Scientific Data* 3 (24 May 2016), 160035 EP –. http://doi.org/10.1038/sdata.2016.35

[15] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla. 2017. ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 468–477. https://doi.org/10.1109/CCGRID.2017.8

[16] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. 2018. Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 227–240. https://www.usenix.org/conference/atc18/presentation/oleksenko

[17] Aravind Ramachandran and Murat Kantarcioglu. 2018. SmartProvenance: A Distributed, Blockchain Based DataProvenance System. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY '18)*. ACM, New York, NY, USA, 35–42. http://doi.acm.org/10.1145/3176258.3176333

[18] JE Rossouw, JP Du Plessis, AJ Benadé, PC Jordaan, JP Kotzé, PL Jooste, and JJ Ferreira. 1983. Coronary risk factor screening in three rural communities. The CORIS baseline study. *South African medical journal* 64, 12 (September 1983), 430–436. http://europepmc.org/abstract/MED/6623218

[19] Scontain. 2020. SCONE-Python. https://sconedocs.github.io/Python/. [Online; Last access: June 03, 2020].

[20] R. Silva, P. Barbosa, and A. Brito. 2017. DynSGX: A Privacy Preserving Toolset for Dinamically Loading Functions into Intel(R) SGX Enclaves. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 314–321.

[21] The Guardian. 2017. Uber concealed massive hack that exposed data of 57m users and drivers. http://www.theguardian.com/technology/2017/nov/21/uber-data-hack-cyber-attack.

[22] The Guardian. 2018. Facebook to contact 87 million users affected by data breach. http://www.theguardian.com/technology/2018/apr/08/facebook-to-contact-the-87-million-users-affected-by-data-breach.

[23] Liehuang Zhu, Yulu Wu, Keke Gai, and Kim-Kwang Raymond Choo. 2019. Controllable and trustworthy blockchain-based cloud data management. *Future Generation Computer Systems* 91 (2019), 527 – 535. http://www.sciencedirect.com/science/article/pii/S0167739X18311993

[24] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized Computation Platform with Guaranteed Privacy. *CoRR* abs/1506.03471 (2015). arXiv:1506.03471 http://arxiv.org/abs/1506.03471