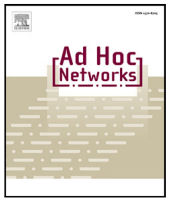
Ad Hoc Networks 147 (2023) 103191

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc



Tamper-proof access control for IoT clouds using enclaves

Guilherme A. Thomaz [a,*], Matheus B. Guerra [a], Matteo Sammarco [b], Marcin Detyniecki [b], Miguel Elias M. Campista [a]

[a] Grupo de Teleinformática e Automação (GTA), COPPE, Universidade Federal do Rio de Janeiro (UFRJ), RJ, Brazil
[b] AXA, Paris, France

ARTICLE INFO

Keywords:
Trusted computing
Access control
Internet of Things
Cloud computing

ABSTRACT

Internet of Things (IoT) devices rely on cloud computing for processing user-sensitive data, like health recordings and geolocalization. In this case, security primitives like cryptography and certificate-based authentication does not prevent the cloud provider from acting against the privacy policy. This paper presents a framework for clouds to execute arbitrarily complex processing tasks over IoT data while maintaining the access control policies over the client's control. We rely on a memory enclave to enforce that the cloud follows personal and customizable access policies and analyzed the security properties of our scheme. The performance evaluation reveals that these robust security improvements come with a latency overhead of just 0.1 ms, confirming the system's viability. The system leverages multi-threaded processing inside an enclave to process thousands of client messages per second, achieving high scalability. This work also contributes with a microbenchmark that identifies how much each step of an enclave application influences the performance and evaluates the enclave viability for performing realistic IoT data processing.

1. Introduction

Internet of Things (IoT) provides connectivity and intelligence to billions of devices in home appliances, industrial plants, vehicles, and agriculture [1–3]. Data collected by IoT sensors are usually sent to a remote server since these devices offer very limited storage and processing capabilities. Companies use the collected data to provide services that promise more comfort and automation to the customers by leveraging cloud infrastructure. Cloud computing reduces costs by instantiating multiple virtual machines on the same physical hardware. In addition, clouds provide elasticity, as physical machines can dynamically and efficiently allocate virtual resources based on demand [4].

Today, many applications rely on IoT data in clouds, e.g., AI-based natural language processing or control of critical industrial processes, which depend on confidential data from clients. At the same time, researchers have already shown that commercial IoT systems are vulnerable to attacks, which allow the attacker to control smart-home devices, such as door locks [5]. Hence, there is a gap in ensuring clients that their data is exclusively used as agreed with companies, even considering the increasing level of legal regulation [6]. Clients then lose control over their data, and companies can misuse it to obtain commercial advantages [7,8]. Current architectures that employ encryption and authentication fail if the company controlling the cloud acts maliciously, as it can run any software without clients' consent.

This paper proposes an IoT cloud architecture in which the client customizes the actions upon sensitive data sent to the cloud and who will have access to it. The system leverages enclaves, which are isolated regions of main memory that prevent access for reading and writing by any computer component, even those with higher privileges, such as the operating system and hypervisors. Enclaves are state-of-art solutions to secure data collection and aggregation, manage cryptographic keys, and secure databases [9–12]. In these proposals, however, the client does not decide which entities have access to its data and which entities cannot. In this sense, our main contribution to the current literature is to ensure that the access control policies to access clients' data are complied with by the cloud server, even when it tries to act maliciously. This work also stands out from other architectures by ensuring cloud security without requiring a specific database, type of sensor, or communication protocol, promoting its adoption in IoT infrastructures already in the market.

The server implementation uses the Intel Software Guard Extensions (SGX) technology to perform processing in enclaves, store persistent data with integrity and confidentiality, and prove to the client that the system is secure. Performance evaluation results reveal that the platform provides security for data processing in enclaves, introducing an imperceptible latency overhead for each client. The system is also scalable, as it processes thousands of requests per second, attesting

* Corresponding author.
E-mail address: guiaraujo@gta.ufrj.br (G.A. Thomaz).

https://doi.org/10.1016/j.adhoc.2023.103191
Received 25 August 2022; Received in revised form 20 March 2023; Accepted 24 April 2023
Available online 2 May 2023

its viability in IoT infrastructures. In addition, we confirm that the main sources of overhead introduced by enclaves are the initialization and the entry processes. For optimized performance, the system: (*i*) initializes the enclave only on the server startup; (*ii*) entries the enclave just once for each message received from the client; and (*iii*) executes only the critical security tasks inside the enclave.

Compared with our previous work,[1] the CACIC (Trusted Access Control Using Enclaves for Internet of Things Data, translating from Portuguese) architecture now relies on the coexistence of multiple threads inside an enclave instead of a single one to handle each publish/query request. This modification improves the performance of CACIC by order of magnitude as enclave initialization imposes a significant overhead. In addition to this implementation improvement, we also analyze CACIC under the microscope using our benchmarking tool based on a high-precision timer. This evaluation explains why queries perform better than publications and allows us to dissect CACIC to highlight the overhead added by particular procedures. We improve the assessment of CACIC by demonstrating the impact of realistic processing tasks, such as IoT data aggregation. Our experiments show that CACIC performs thousands of publications and queries per second and does not introduce a perceptible latency for the final user, even for realistic IoT processing tasks.

The remaining of this article is organized as follows. Section 2 describes the IoT cloud scenario and attacker model. Section 3 introduces trusted computing technology, which enables processing in isolated memory enclaves. Section 4 details the proposed architecture, specifying the modules, protocols, and message formats developed. This section also highlights the consequences of each design decision in the security properties of the system. Section 5 provides the implementation benchmark results. Section 6 discusses the related work. Finally, Section 7 concludes the paper and presents directions for future work.

2. Infrastructure and threat model

Fig. 1 shows a typical IoT infrastructure where sensors send the collected data to be processed in a remote server due to their low computational power. The access points apply some pre-processing over the data, like encryption and data correction, composing the edge computing [13]. Then, the data is sent to virtual machines instantiated in the cloud, which apply more computationally intensive processing tasks over a massive amount of data, like machine learning models training [14]. Companies and devices must query some of these data in the cloud to make business decisions and automate processes.

Once the data is uploaded to the cloud, the client loses control over what will be done with the data if the servers are unreliable. Eibl et al. demonstrate that data regarding energy consumption processed in the cloud reveal private information, such as the number of people using a facility at a given time [15]. In another scenario, a malicious agent can impersonate an authentic client to fabricate fake data to access an electronic lock or harm an automated industrial production. These examples show that cloud servers must always meet the following requirements:

1. **confidentiality**: sensitive data cannot be revealed to any client;
2. **integrity**: data cannot be changed by any client
3. **authentication**: clients must identify themselves to send and receive data; and
4. **access control**: clients must choose who has the access permissions to their data and what can be done with it.

[1] Previous work available at http://www.gta.ufrj.br/ftp/gta/TechReports/TGS22.pdf. This paper is written in Portuguese and accepted in a Brazilian conference (SBRC 2022).

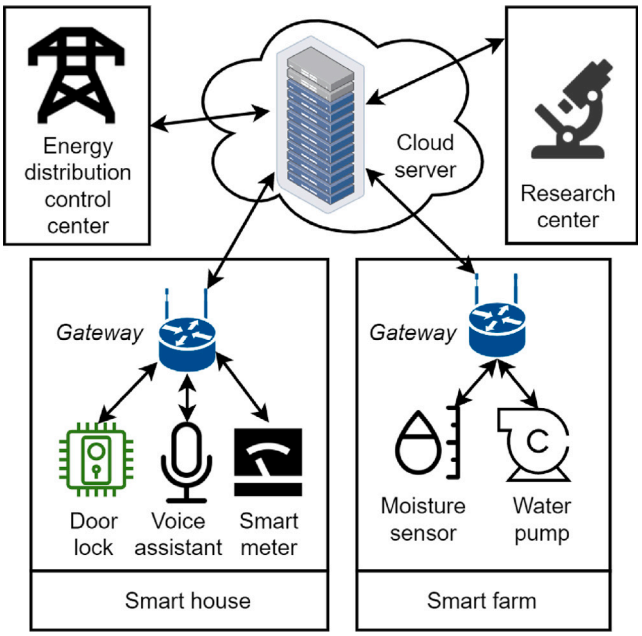


Fig. 1. Typical IoT scenario where devices send data to the cloud. The cloud server must process the data and provide services to other devices, companies, and institutions. Clients' privacy is at risk if the cloud uses their data maliciously.

Intrusion Detection Systems (IDS) are efficient in preventing network intrusion attacks that could compromise system availability and data security [16]. However, traditional security schemes assume that the cloud provider is trustworthy and uses the clients' data according to the privacy policy [17,18]. We consider a threat model where the attacker can be a malicious insider client or a compromised operating system. Therefore, the attacker can get privileged access to the server [19] and can:

1. read or write any file;
2. intercept, read, re-transmit or fabricate any network package.
3. execute or modify any application;

Nevertheless, the attacker cannot:

1. access the place where data is generated, like residences or factories where sensors, actuators, and local area network gateways are installed;
2. tamper with cryptographic primitives;
3. perform physical attacks on the CPU package or side-channel attacks.

To meet the security requirements with this threat model, we leverage trusted computing using enclaves, as described in the next section.

3. Trusted computing using enclaves

Trusted computing constitutes a set of technologies that offers security using hardware resources. The most commonly used trusted computing technology in cloud computing is the Software Guard Extensions (SGX), available in the most recent Intel Xeon processors. Intel SGX uses special x86 architecture instructions to instantiate isolated regions in the computer's main memory, called enclaves. Enclave contents are encrypted in main memory and are only decrypted inside the CPU using a unique key that never leaves the device and cannot be accessed externally [20]. Fig. 2 shows that the application first initializes the enclave by creating and transferring memory pages to the enclave reserved memory region, which imposes a significant overhead.

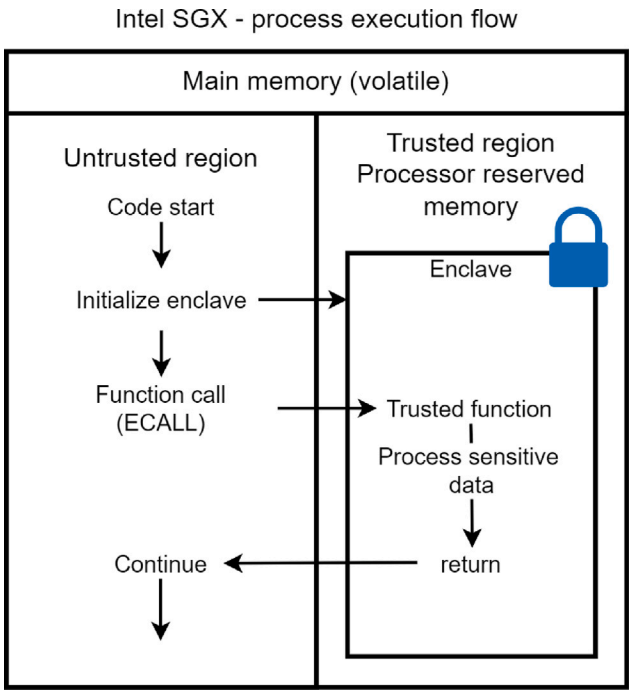


Fig. 2. An application initializes an enclave and calls a trusted function (`ECALL` instruction) that processes confidential data in an isolated environment, with privileged access by the processor.

The SGX instructions that initialize the enclave also compute a hash-based measurement of its content at the microarchitectural level so that no software can tamper with the result. When an insecure application needs to execute sensitive computations, it can access an enclave routine through an `ECALL` instruction, which inserts overhead by copying data from untrusted memory to the enclave reserved memory. The enclave entry also involves security checks at the microarchitectural level to ensure confidentiality and integrity of code and data even if the BIOS, the operating system, the hypervisor, or the super-user are malicious [21].

Before sending its data to an enclave in the cloud, the client must ask the server to perform a remote attestation. The attestation proves that the server is running on a legitimate SGX-enabled platform with the expected code [20]. Intel's Data Center Attestation Primitives (DCAP) is an attestation mechanism based on public key infrastructure (PKI) [22]. A pre-made Intel enclave, called quoting enclave, uses a unique private key, in the format of the x509 certificate, to sign the enclave measurement done in the initialization step. This signed measurement is forwarded to the client, and the enclave is considered trusted if the signature can be decrypted using the data center public key provided on the certificate. The client can also compare the measurement with an expected value to ensure that the enclave code loaded into the main memory is the expected. The attestation protocol is based on Elliptic Curve Diffie–Hellman Key Exchange (ECDHKE) between client and server enclave to transmit the report using symmetric encryption [23]. If the attestation is successful, the client can transmit confidential secrets over this encrypted channel so that only the enclave can decrypt them.

Intel also offers a service called Enhanced Privacy ID (EPID), where the client needs to access an online centralized Intel Attestation Server (IAS) to verify the validity of the signature [24,25]. DCAP eliminates relying on a single verification point since Intel servers are only requested in the provisioning step. In this step, Intel servers provide the quoting enclave with the private key that never leaves this enclave used to sign the measurement, the corresponding public key, and the lists of revoked certificates.

Enclaves have the sealing capability, which allows data to be persistently stored in secondary storage out of the main memory region of the enclave. The data is encrypted with a key derived from a hard-coded secret inside the CPU. As this key is never visible outside the enclave scope, sealed data can only be recovered by an enclave [26].

A limitation of using enclaves regards the attestation procedure which requires the code to be publicly available for the attesters [27]. In fact, the enclave code is already available outside the protected memory before it is loaded into the enclave. Also, distributed computing is more complex with enclaves because they all must attest to each other. Furthermore, transferring data into an enclave is a known source of overhead. Thus, the overhead introduced by distributing a task into multiple enclaves may become even more significant [28].

The threat model of SGX assumes that the attacker cannot perform a physical attack inside the CPU package to steal hard-coded secrets or tamper with the microcode. However, a well-known SGX vulnerability that compromises enclave confidentiality is side-channel attacks. These software attacks can happen through OS kernel manipulation to infer the application memory access pattern by measuring computation time [29]. The most common side-channel attack regarding SGX is the cache side-channel attack, in which the attacker uses a malicious process to fill the cache used by a given core executing enclave code. Then, every time enclave pages are accessed, they are loaded into the cache, evicting existing attacker pages. Periodically, the malicious process accesses its data, which filled the cache before. If the access time to a certain location becomes high, this means that the content there was evicted. This location was then accessed by the enclave [30]. Information obtained this way is reverse engineered into actual data since the attacker has access to how the OS kernel maps the application memory. Many solutions have been proposed to protect SGX enclaves from side-channel attacks but eliminating this threat is still an open problem in the literature that goes beyond the scope of this paper [31].

4. System architecture

Fig. 3 illustrates the architecture composed of clients that send data to or receive data from a trusted cloud server through its access points. The communication uses Hyper Text Transfer Protocol Secure (HTTPS) [32] because it is a secure and widely adopted protocol in web servers. However, the sole deployment of HTTPS is not enough to meet the security requirements proposed in Section 2. Therefore, we leverage memory enclaves to equip our system with the four procedures described below. Fig. 4 summarizes the protocols that clients and the server follow to accomplish these procedures.

4.1. Protocols for secure procedures

Initially, the client access point must register in the platform (1) to share its symmetric communication key (`CK`) with the server, used to encrypt sent data and decrypt received data. In the registration, the client access point must attest that the server is trusted (2), as presented in Section 3. The access point sends the key to the enclave if the attestation is successful. The attestation procedure involves not only one message but multiple messages in both directions during the key exchange. We indicate that by using double arrows in the diagram. The server seals the key before writing it on disk and associates this key with the client public key used to identify who is sending the following messages.

When the access point receives data from sensors for publication (3), it assembles a message (`M[publication]`, 4) with the following fields:

```
M[publication]
=[pub|nonce|type|size|CK(data|perm|nonce)].
```

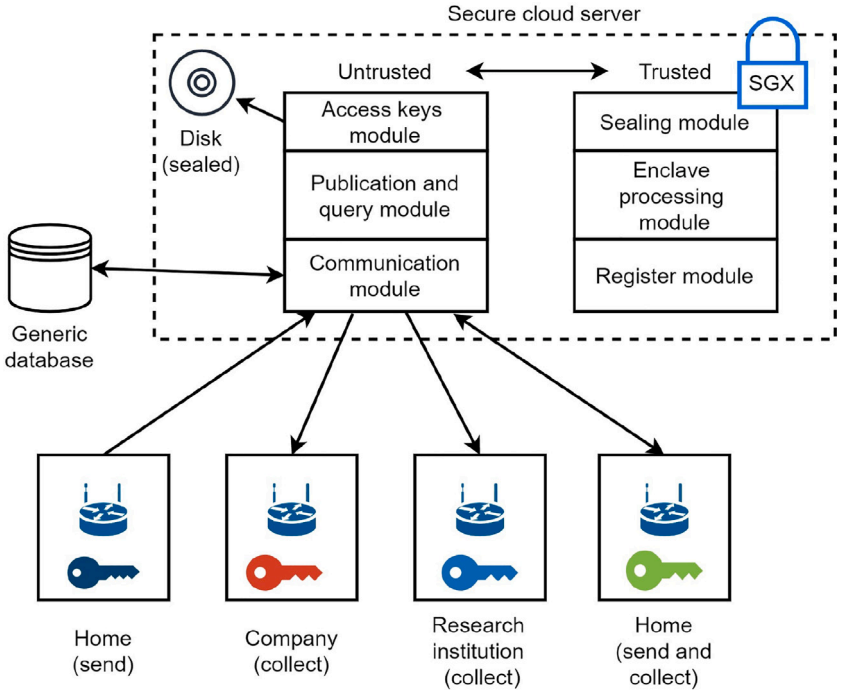


Fig. 3. Client access points send and receive data encrypted with their registered communication keys. Data received by the cloud is decrypted, processed, and encrypted in enclaves before being published.

The symbol | symbolizes the concatenation of *bytes*. The public key (pub) identifies the client to the server, which locates the communication key (CK) sealed on disk. The data type identifies the originating device so that the platform applies the appropriate processing for each case. Client access permissions (perm) permit the server to be aware of who can access the published data using a list of public keys. In addition, permissions are encrypted together with the data, as they are private information. The nonce is sent to avoid replay attacks. Once the publication message is received, the server enclave retrieves the communication key, decrypts the data, verifies if the nonce is fresh, and applies some processing, depending on the data type (5). The enclave also verifies if the encrypted nonce is consistent with the plain-text nonce to confirm that an authenticated client constructed the message with the correct CK.

To illustrate a common application of the system, we describe the aggregation of energy consumption data of a client. The client sends a M[publication] identifying the data as a smart meter data using type field. After receiving the message and identifying the type, the server reads from the database other energy consumption data from the client. Therefore, the enclave can decrypt the samples and calculate a sum, as described in Algorithm 1. With the proposed architecture, a control center can use this data to make decisions regarding energy distribution planning, for example, without gaining access to confidential consumption patterns. Other proposals use homomorphic cryptography for computing the sum while the data is encrypted. However, enclaves stand out for introducing a much smaller processing delay, as well as allowing arbitrarily complex operations on the data, such as filtering [33]. Before publication in the database, the data is sealed with access permissions and nonce.

In the query procedure, the client access point sends a message M[query] (7) with the following format:

$$M[query]=[pub|nonce|size|req|CK(nonce)],$$

where pub is the public key that identifies the client and req is the command used to locate and request the data in the database. The request is forwarded to the database (8) and the response is the sealed data (9). Then, the server enclave retrieves the client communication key (CK), decrypts the nonce with the CK and verifies if the nonce is fresh and consistent with the plain-text nonce (10). Therefore, it unseals the data received from the database and checks if the access permissions (perm stored with the data) allow access to the data by the interested client (10). If so, the enclave encrypts the result with the CK and sends it to the access point. Finally, the access point decrypts the received data using the CK, making it available to the interested device (11).

An advantage of our proposal is that it does not impose any computational requirements or rigid message format on the sensor. This advantage is a consequence of delegating to the access point the key establishment and data encryption, regardless of the message format used by the sensors. The diversity of IoT devices in terms of protocols and performance specifications justifies this design choice, making the proposal flexible and device-agnostic [34]. The architecture is database-agnostic, so the query message has a flexible format according to the application. The cloud provider is free to deploy the database on the same machine as the server, on a separate server, or even in a distributed fashion. Our approach is not agnostic regarding the access point since we assume it must have the computational power to perform encryption and HTTPS communication. That way, the sensor can employ more lightweight network protocols, such as the Constrained Application Protocol (CoAP), since the access point can act as a proxy. The access point translates the message by identifying the data source (type), encrypting the data together with access permissions, preparing the publication message M[publication], and forwarding it using the widely adopted HTTPS.

A differential of the architecture is the possibility for the client to configure access permissions on the access point. The access point adds the field perm to the publication message (M[publication]) so that the enclave prevents or allows access when a client requests this data. A client can prevent the biomedical signals from their smartwatch from being made available to the watch manufacturer but can allow them to be accessed by a biomedical research institution, for example. This is only possible because the client has established a trustworthy relationship with the trusted execution environment in the cloud.

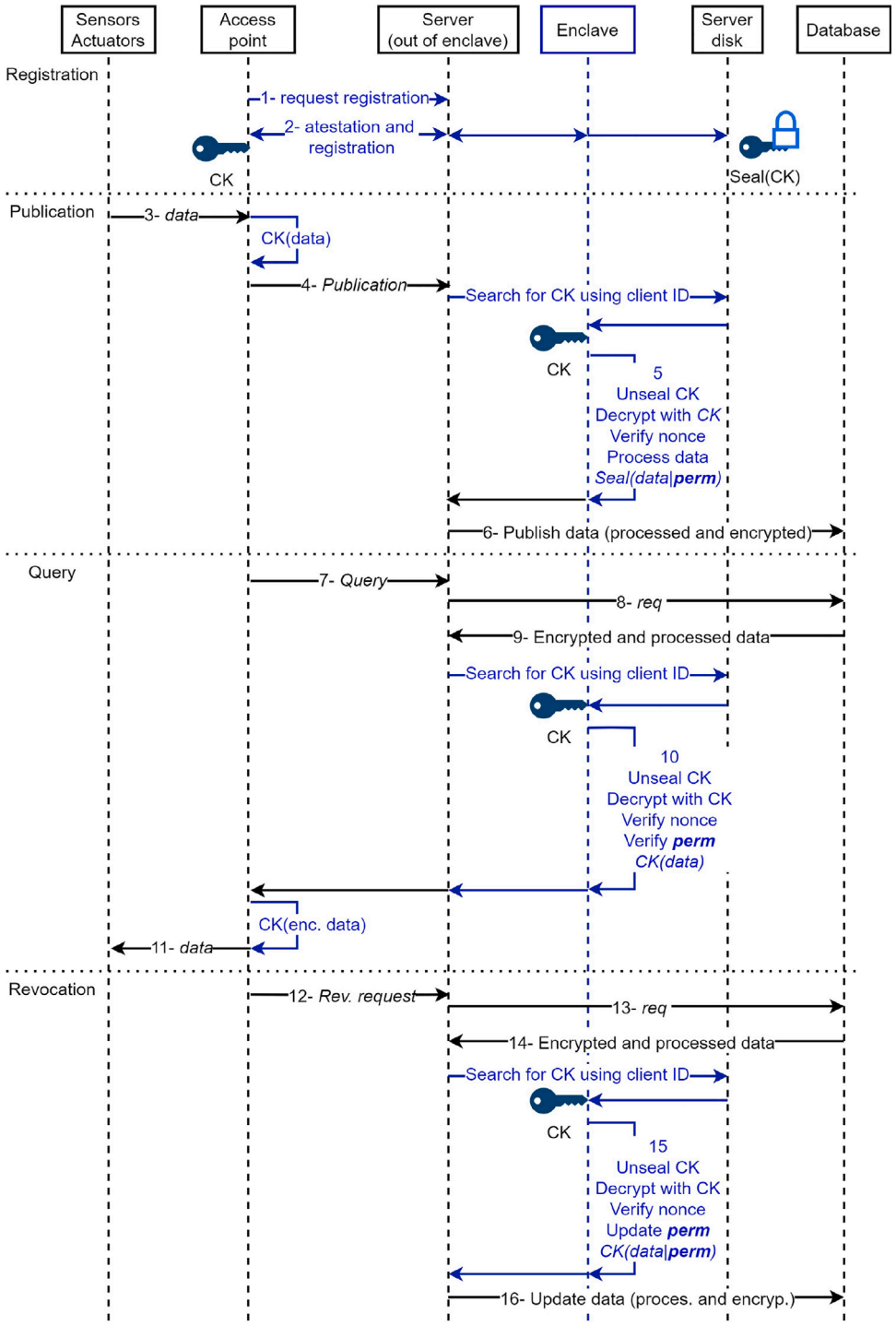


Fig. 4. Data in transit is encrypted with the communication key (CK) and stored data is encrypted with the enclave-exclusive sealing key. Only the enclave decrypts and processes the data in the cloud. The blue color highlights the security mechanisms introduced by the architecture. The customization of access permissions (perm) by the client is an innovation of the proposal.

This access control mechanism is based on sticky policies, first proposed by Karjoth et al. [35]. Sticky policies are stored and transmitted along with the data, providing the data producer with in-depth control over access rules for every single data unit [36]. This approach relies on a Trusted Authority (TA) for managing decryption keys and enforcing access control, which is the enclave in our proposal. For evaluating the system, we implement the policies as Access Control Lists (ACL) containing the identity of the clients allowed to access the data. Although this implementation is simple, the architecture supports other access control attributes like expiration timestamps or restrictions for the data usage.

The client can revoke the access permission or even completely remove the data, sending a revocation message in the following format:

```
M[revocation]
=[pub|nonce|size|req|CK(perm|nonce)].
```

The server first queries the data published together with the access permissions from the database using the `req` command (13, 14). The

Algorithm 1 Algorithm for aggregating data inside the enclave. The MAC and nonce checks are omitted. The enclave uses the result to build an encrypted sample for publication.

Input: $(size, enc\_data[size], sealed\_CK)$
Output: sum
  $sum \leftarrow 0$
  $CK \leftarrow$ Unseal$(sealed_CK)$
  $counter \leftarrow 0$
  while $counter \neq size$ **do**
    $data \leftarrow$ Decrypt$(enc_data[counter], CK)$
    $payload \leftarrow$ GetPayload$(data)$
    $sum \leftarrow sum + payload$
    $counter \leftarrow counter + 1$
  end while

enclave verifies the nonce and unseals the data as described for other procedures. Therefore, it updates the access permissions of this data with the new permissions sent on the `perm` field (15). If this field in the message is null, this means that the data must be removed from the database. If not, it overwrites the database entry with new data containing new access permissions (16). In essence, the revocation procedure is a combination of querying old data and publishing a new one.

4.2. Security properties

This section describes how the mechanisms described in Section 4.1 can make the system resilient to the threat model described in Section 2. We do not prove the system security, since formal methods are out of the scope of this paper. However, we would like to provide a pointer on this area that could be interesting for readers [37]. An attacker who gains privileged access control over the server may read data from arbitrary files. The system stores two sensible information in the file system: the client's communication keys (`CK`) and the client's data in the database, along with the access permissions. With the `CK`, the attacker can decrypt the data and access permissions sent on the client publication messages. It can even use the `CK` to build a valid query request (`M[query]`), send it to the server enclave, and decrypt the result, violating the access control policies and authentication requirements. To avoid these attacks, the server encrypts the communication keys with the sealing key. We assume that no one can tamper with the SGX hardware, making it impossible for the attacker to obtain the sealing key. The access point generates the `CK` during the attestation, but this does not affect the security of the proposal, as the attacker model assumes that the client is trustworthy. The remote client is responsible for maintaining its `CK` confidential. The database is also encrypted with the sealing key. Since all data encrypted with the sealing key is also accompanied by a Message Authentication Code (MAC), the attacker cannot write fake data into the database or modify a `CK`. We do not deal with system availability issues such as deleting contents from the disk or simply shutting down the server by the cloud provider.

As described in Section 2, the attacker can also intercept network packets to read or modify their content. Data and database requests are encrypted with the communication key and are accompanied by a MAC, ensuring communication confidentiality and integrity, respectively. An attacker cannot replicate old messages because of the `nonce` freshness verification performed by the enclave. The enclave compares the plaintext `nonce` with the decrypted `nonce` using the `CK`. That way, an attacker interested in generating a random `nonce` to fabricate a fake message will not be able to encrypt the `nonce` since it does not have access to the client `CK`. Availability attacks such as Distributed Denial of Service attacks (DDoS) or packet drop attacks are not considered.

The attacks described above only manipulate the I/O (network or disk), and do not involve modifying currently running applications or running a malicious application. As described in Section 2, the attacker can modify the existing code to introduce a malicious task such as revealing client data in the clear. The OS is also not trusted, so an attacker can read, write or execute code in any memory region outside the one reserved for the enclave. To avoid these attacks, the cloud developer must design the applications to process the data inside the enclave, to ensure confidentiality and integrity at runtime. All the sensible information, like `CK`, data, access permissions, and encrypted message fields are only decrypted after entering the enclave and only encrypted before leaving the enclave, as shown in Fig. 4. The attestation protocol at the registration phase guarantees that the server does not falsify the code that runs inside the enclave, as the client checks if the signed measurement of the enclave code matches an expected value. As described in Section 3, the signature relies on a trusted system enclave owned by Intel that cannot be tampered with by any other component in the software stack. The client does not proceed with the communication if the code measurement is not the expected one, or if the signature cannot be verified, indicating that the server does not leverage a genuine enclave. That way, the client has complete control over who accesses its data, as access requests are processed in a trusted and isolated environment, even if the server is controlled by a malicious agent.

5. Implementation and results

The performance evaluation aims to: (i) verify if the architecture is scalable, (ii) measure the processing latency introduced by enclaves, and (iii) verify how the architecture deals with a processing task that operates over a large amount of data instead of just publishing the received data. IoT cloud systems need to serve a large number of clients at the same time, given the growing number of devices. This challenge justifies evaluating the performance of security proposals with a large number of clients [38]. The experiments evaluate the number of requests the platform processes per second and the time for publishing and querying data. The trusted server was implemented in a computer with an Intel i9-10900 CPU 2.80 GHz, 32 GB RAM, and 20 threads. This machine also sends messages for querying and publishing synthetic data in the *Ultralight* 2.0 format, a standard adopted by the FIWARE platform for developing intelligent applications used in other works [11,39–41]. Still, the data format supported by the platform is generic, as described in Section 4. A Software Development Kit (SDK) for using the Intel SGX trusted instructions in C++ language was used.[2]

We organize our results into four subsections. Section 5.1 presents the results concerning the maximum processing rate in publications per second and queries per second. It also presents the latency increase when the system is at its maximum processing capacity. Section 5.2 presents the latency overhead introduced by the enclave. Section 5.3 presents the microbenchmark results, detailing the influence of each step on the total overhead. Section 5.4 presents the time for aggregating data samples. The idea is to confirm the system viability for realistic IoT processing. Section 5.5 presents the server resource usage while processing requests, revealing how much of the server CPU and memory capacity is used. We do not measure the registration time since the Intel attestation procedure is not a contribution from our work, and other works already evaluated its performance in the literature [32,42]. The attestation time greatly depends on the network latency because of message exchanging, as presented in our previous work [24]. Also, the attestation procedure is executed only once by the user when bootstrapping the system. The results use 95% confidence interval.

5.1. System scalability

The first experiment evaluates the number of publication messages a server can process per second, which includes steps 4 to 6, as denoted

[2] Project repository available at https://github.com/GTA-UFRJ-team/TACIoT.

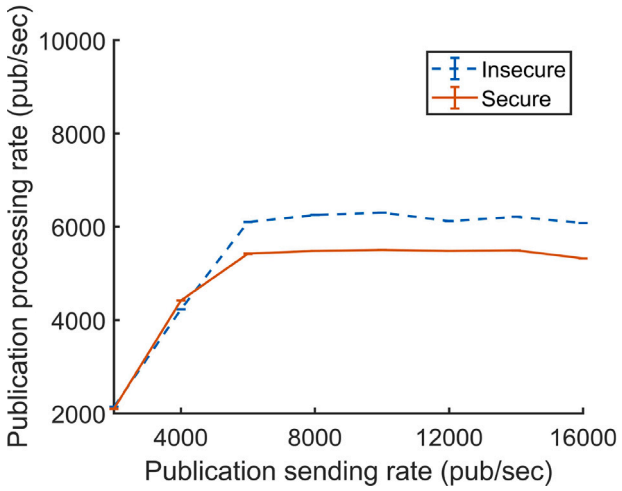


Fig. 5. The enclave reduces the maximum publication rate by 12%, but the system still processes thousands of publications per second.

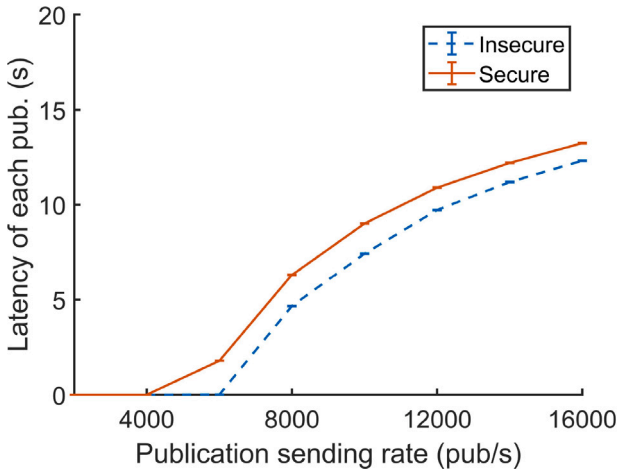


Fig. 6. The latency quickly increases when the system achieves its maximum publication rate since new messages are inserted into a queue.

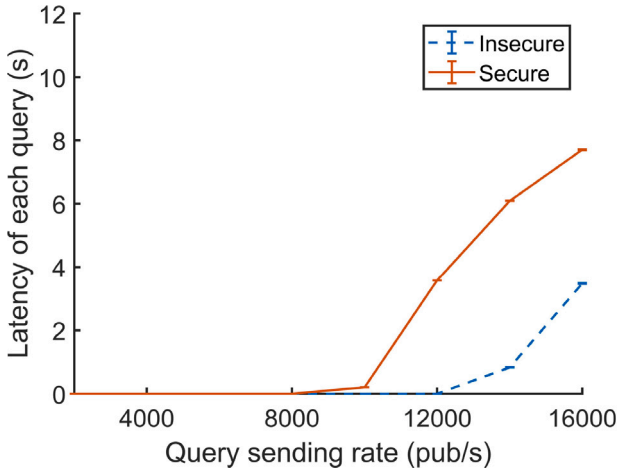


Fig. 7. This result is analogous to Fig. 6. The system processes more queries than publications per second.

in Fig. 4. We reproduce clients sending publication messages at the same time with a constant rate of publications per second (pub/sec)

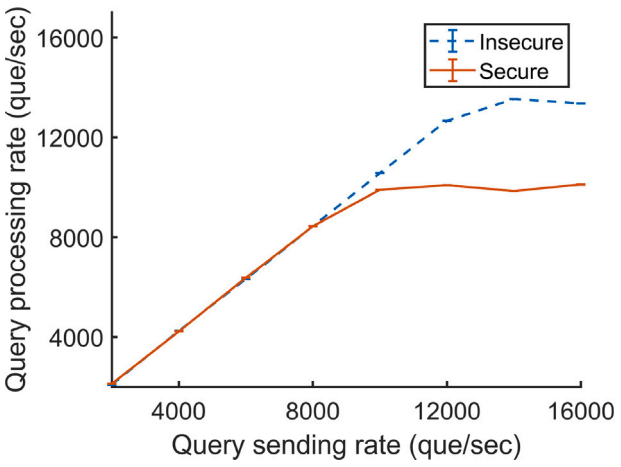


Fig. 8. This result is analogous to Fig. 5. The reduction in message processing rate due to the enclave is more significant in that case since queries are faster.

and measure the server processing rate. For this, we use the wrk2 tool for sending constant HTTP workloads and evaluating the performance statistics [43]. The experiment was repeated without the enclave (step 5 in Fig. 4) to evaluate the overhead added by the proposed security mechanisms. Fig. 5 shows that the publication rate increases rapidly with the sending rate as the system processes requests in parallel. For less than 4k messages sent per second, the processing rate is similar to the sending rate, suggesting that, until this point, the server does not reach its processing limit yet. As the sending rate increases, the curve inclination decreases until the processing rate stabilizes at a maximum value, representing the server processing capacity limit. The maximum processing rate is 5500 pub/sec for the secure server (with enclave) and 6250 pub/sec for the insecure server (without enclave). This represents a 12% decrease in the maximum publication processing rate, which is small considering that our system can still serve thousands of clients per second with a much stronger threat model.

Fig. 6 reveals that the latency of a single publication message is in the order of a few milliseconds when the system is below its processing rate limit. When the system achieves its maximum parallelism, every new message is inserted into a queue and is only processed when a thread becomes available. This is confirmed by the rapid growth in latency around the maximum processing rate. The client will notice a response time in the order of seconds in both secure and insecure cases when the server is at its parallelism limit. Repeating the experiments using a server with higher parallelism will lead to a higher maximum processing rate.

The second experiment evaluates the number of query messages processed per second (que/sec) by the server, which includes steps 7 to 10 in Fig. 4. We follow the same methodology described for the first experiment. The results presented in Fig. 8 are analogous to the first experiment, with a maximum rate equal to 13,540 que/sec without enclaves and 10,110 que/sec with enclaves. In both cases, the query rate is higher than the publication rate, suggesting that the query procedure is faster than the publication procedure. The overhead added by the enclave in the query is not that negligible, leading to a performance drop of 25% compared with the insecure case.

Fig. 7 is analogous to Fig. 6, confirming that the latency increases rapidly when the server reaches its maximum processing rate. For the query messages, this maximum processing rate is much higher than for publication messages. As we increase the sending rate, the curve for the secure server starts to grow before the curve for the insecure server. This happens because the maximum rate is much lower when using the enclave.

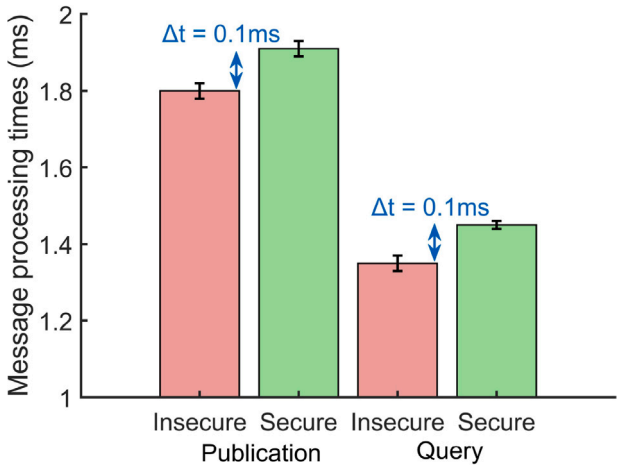


Fig. 9. The enclave adds an imperceptible latency overhead for the clients. The enclave does not perform any computation-intensive processing and the overhead is mainly due to the ECALL.

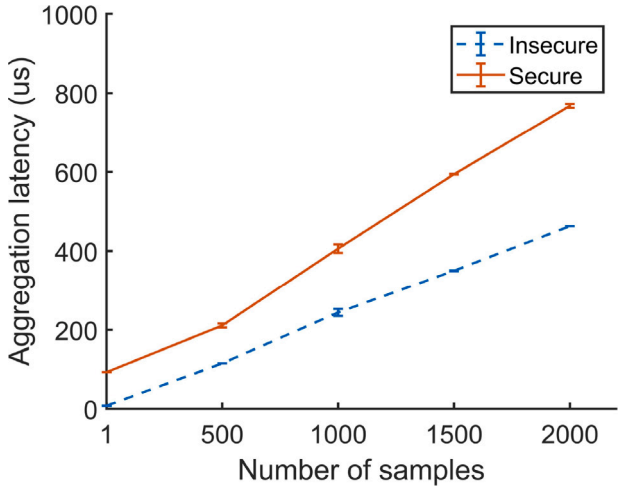


Fig. 10. The enclave overhead increases as more data is transferred to it. However, the time for reading the samples from the database is much higher than this enclave overhead.

5.2. Message latency

The third experiment evaluates the time between the client transmission of a single request and the reception of a successful response using the wrk2 tool. This experiment aims to verify if the enclave adds significant overhead in response time. Fig. 9 shows that, in both cases, the enclave adds only 0.1 ms of delay, which is imperceptible for most applications. The enclave does not apply expansive computation over the published data since the experiment only evaluates the publication/query messages processing times. The time introduced by the enclave processing is the same for publication and query because the system executes three sealing/unsealing and encryption/decryption operations in both cases, as presented in steps 5 and 10 of Fig. 4. The experiment also confirms that publication time is approximately 25% higher than the query time for both secure and insecure servers. This result explains why the maximum query rate of the server is much higher than the maximum publication rate, as discussed before. The time added by the enclave has more influence on the total query time than on the total publication time since the query procedure is faster. This explains the significant reduction in the maximum query rate caused by the enclave, as discussed before. These experiments confirm an inverse relationship between the latency of a single request in seconds and the maximum rate in requests per second. The performance of the revocation procedure must be analogous as it is a combination of querying old data and publishing it with updated access permissions.

5.3. Microbenchmarking

The previous experiments treated the server as a black box since we leverage the wrk2 tool to simulate clients interacting with the server through HTTP requests and measure the time and the rate by the responses. The fourth experiment analyzes the system under the microscope, detailing how much each procedure contributes to the total overhead. For this, we developed our own custom benchmarking tool based on high-precision timers. The microbenchmarking tool individually measures the elapsed time of each critical code section and computes statistics, such as mean and variance, at the end of an experiment epoch, with 1 µs resolution. This measurement does not include the overhead added by HTTP processing. Table 1 presents the mean elapsed time in microseconds for each procedure. After repeatedly running the experiments, we observe that the confidence interval is negligible (lower than the resolution). The first row represents enclave initialization, which includes creating and transferring memory pages to the processor reserved memory, as described in Section 3. The following four rows relate to the publication procedure. In that case, the bottleneck is the time to write to the secondary storage, which does not depend on the proposed architecture. The last four rows relate to the query procedure. In that case, the smaller time to access the database led to much better performance. The bottleneck for the query becomes the enclave, explaining why removing the enclave from the query procedure results in a substantial performance improvement (Figs. 8 and 7).

Table 1
Publication and query latency microbenchmark results. The publication time is dominated by the disk writing and the query time is dominated by the enclave.

Procedure	Elapsed time (µs)
Enclave initialization	5848
Enclave publication message processing	105
Database write	387
Other procedures	42
Total publication time (not considering HTTP)	534
Enclave query message processing	105
Database read	56
Other procedures	74
Total query time (not considering HTTP)	235

Previously, we proposed initializing and destroying an enclave for processing each publication and query message. Table 1 revealed that just the enclave initialization time is 10 times higher than the total publication time and 25 times higher than the total query time, dominating the total overhead. This very slow initialization imposes a significant performance bottleneck in our previous implementation. Now, the server takes advantage of multi-threaded processing inside a single enclave to initialize the enclave only on the startup, optimizing the system performance by order of magnitudes.

5.4. Use case: Aggregation

The fifth experiment evaluates the time taken to perform realistic data processing over some fictitious IoT data samples. We consider the use case of aggregation, described in Section 4. Fig. 10 presents the time to aggregate the samples with and without enclaves. We measure the time for aggregating 1; 500; 1000; 1500; and 2000 samples to

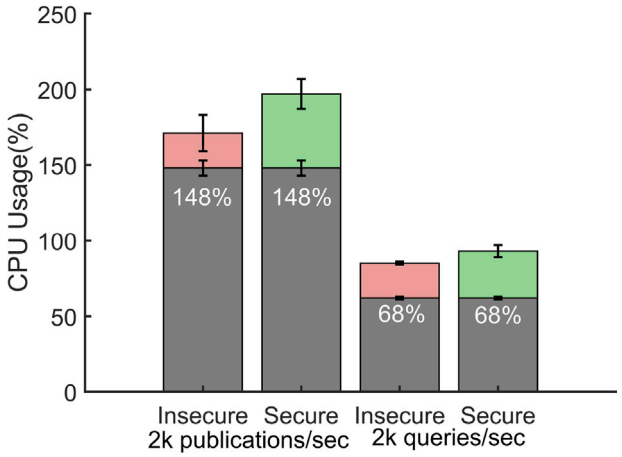


Fig. 11. Percentage of CPU time for the server process, considering the sum of all CPUs. The resource usage is dominated by the HTTPS protocol implementation, as seen in gray.

verify the influence of the number of samples. When the number of samples is 1, the system just decrypts the data and does not perform any sum. In both cases, the time depends linearly on the number of data aggregated. This is expected since the asymptotic complexity of the Algorithm 1 is $O(n)$, where n is the number of samples, considering that it repeats the same efficient decryption and sum operations for every data sample. However, the aggregation time with the enclave is always higher. This is a consequence of an additional step to execute an `ECALL` instruction to call the enclave and transfer the data buffer by reference before starting the described algorithm. Other works demonstrated that the `ECALL` instruction is one of the main performance overheads introduced by enclave utilization, going from 4 K clock cycles up to 20 k clock cycles, depending on the amount of data transferred to the enclave [44,45]. The difference between the secure and insecure curves represents the overhead added by entering the enclave. As the amount of data transferred to the enclave grows, the time taken to enter the enclave also increases, and the curves diverge. This is compatible with the results presented since the secure curve inclination is higher than the insecure case.

The time for reading a single sample from the secondary storage is approximately 56 µs, as presented in Table 1. For thousands of samples, the total reading time is in the order of tens of milliseconds, which is much higher than the time for processing these samples. As the number of samples grows, the overhead introduced by the enclave increases, but the time for reading these samples from the database increases as well. This result suggests that the overhead introduced by the enclave can be imperceptible for the client if the application involves intensive I/O operations, as the bottleneck is on disk read. These results are promising, considering that many applications, such as those requiring machine learning model training, are I/O intensive. Even though this is a straightforward conclusion, we still need more experiments to confirm.

5.5. Resources usage

The sixth experiment evaluates the CPU usage in terms of the percentage (%) of CPU time the kernel dedicated to the server process and physical memory usage in MB. We used the `psrecord` tool to monitor the server's resources while receiving 2 k HTTPS messages per second for 30 s. Fig. 11 shows the CPU consumption. The values can be greater than 100% because the result is the sum of all percentages of time dedicated to the process by all CPUs. Thus, the maximum CPU usage for the entire 20-core machine is 2000%. We initially disabled the publication and query functionalities to measure the resource usage by the HTTPS protocol alone, i.e., to create a baseline, presented in Fig. 11 in gray bars. In that case, the mean CPU usage for receiving HTTPS messages is 148% for publications and 68% for queries. Publication messages tend to be larger, taking more time to process by the HTTPS library. Afterward, we re-enabled the system functionalities to measure the impact of the proposed publication and query protocols on resource usage. For that, we define the CPU usage overhead as the difference between the CPU usage while processing the requests and the CPU usage baseline, as defined before. For the insecure case, the CPU usage overhead is 24% for both publications and queries. For the secure case, on the other hand, the CPU usage overhead is 50% for publications and 32% for queries. The architecture does not impose significant CPU usage, considering that the overhead introduced by the security procedures is not very significant. Furthermore, it is evident that the HTTPS protocol dominates CPU usage and that the values are very distant from the maximum server capacity. The memory consumption maintained stable at around 7 MB for all experiments, indicating that the operations are not memory intensive. Repeating the experiment with rates greater than 2 k messages per second leads to similar results. These results confirm that high message processing rates do not require an architecture with lots of CPU and memory resources.

5.6. Validity discussion

This section discusses some precautions taken to ensure the validity of the results and presents some limitations for generalizing the conclusions [46]. Experiments 1, 2, 3, 4, and 6 aim to confirm the impact of the architecture and protocol, presented in Section 4, on scalability, latency, and resource usage. These experiments are direct measurements of two situations contrasting only by the presence or not of security procedures, confirming the construct validity. For internal validation, we repeated the experiments and performed statistical mean and standard deviation estimations to reduce stochastic errors. Also, we leverage widely adopted tools such as wrk2, C++ high-precision timers, and psrecord for conducting measurements to minimize the chance of systematic errors in instrumentation. Also, the obtained results lead to similar and coherent conclusions: (i) publication performs worse than queries; (ii) the enclave is the main bottleneck only for queries; and (iii) for publications, the time for writing into the secondary storage is the main bottleneck.

The validity of the results presents some limitations. Experiment 5 indicates that the system performs well in aggregating large amounts of data. However, more experiments must be run to ensure that the architecture does not hinder arbitrarily complex tasks. The proposed architecture is agnostic to the database, but the data is directly written into a file on the disk during the experiments. Therefore, in a production environment, the results depend on the overhead introduced by the deployed database. Still, the experiments confirm that even with the security procedures proposed, the system is scalable and presents low latency. The developer is responsible for choosing a scalable and resource-efficient database to achieve high-performance requirements. Experiments run on a single computer, whereas cloud computing usually leverages virtualization and distributed processing and storage. Future works must deal with intensive processing tasks distributed between multiple virtual machines and multiple enclaves to extend the result's external validity to scenarios more similar to the cloud.

6. Related work

Trusted computing technologies are used in the literature to protect data at every component of IoT infrastructure when the user controlling the component can be malicious. Therefore, the works presented here can be divided into the following categories: (i) trusted sensor [9]; (ii) trusted database [12]; (iii) trusted consumer [11]; and (iv) trusted middleware [10,47,48].

Yang et al. implement a custom trusted computing platform for Industrial Internet of Things (IIoT) to ensure the reliability of published data in a scenario where sensors and actuators can be compromised [9]. The proposal uses a blockchain to maintain a transparent, auditable, and distributed record of the data, as well as to perform more complex processing on the data with smart contracts [49]. The paper does not deal with the security of data sent to the cloud. Moreover, the system is not completely confidential, since the data published in the distributed ledger is public.

Priebe et al. developed a database engine running inside memory enclaves using Intel SGX [12]. The system is promising for the IoT architecture, in which sensors publish and query data in the cloud, as requests are processed in an environment isolated from the administrator and the operating system. The proposal focuses on data storage and is not a complete solution for IoT. In a scenario where the cloud is malicious, the attacker gains access to the data processed in the main memory before it is published. Zegzhda et al. describe the use of SGX functionalities to provide cloud Platform-as-a-Service (PaaS) services for loading and running generic web applications on enclaves [18]. The work neither focuses on sensitive IoT data storage nor implements an architecture.

Li et al. perform smart meters data aggregation, dynamic pricing, and consumption forecasting within memory enclaves [10]. These tasks require access to data from several clients and are prohibitive for devices with low computing power, forcing the users to deliver their data to a remote server. The architecture employs Intel SGX on the client access point and the remote server. Furthermore, it focuses on the secure availability of data for the energy company and does not implement an access control system for authorized customers to query the stored data. Trusted execution environments are not the only tools to ensure security while data is processed. Silva et al. compare the aggregation time of energy consumption measurements using trusted computing with Intel SGX and homomorphic computing [33]. The two aggregation mechanisms ensure runtime privacy. However, trusted computing performed ten thousand times faster, confirming that trusted execution environments are not only secure but also offer a much lower overhead.

Valadares et al. expand the architecture of the FIWARE smart applications development platform to provide sensitive sensor data to authenticated users [11]. The authors use enclaves in the cloud to store keys for encrypting data generated by producers and decrypting data queried by consumers. The architecture does not process data in the cloud, requiring the consumer to have a processor with SGX support, perform an attestation to receive the data decryption key, and process the data locally. The cloud platform implements a publish–subscribe server, responsible only for disseminating the data to authenticated consumers and managing the keys. The platform does not allow the producer to customize the actions done upon their data.

Ayoade et al. leverages trusted computing to process IoT data from different companies in a shared cloud middleware [47]. The paper highlights that, despite the high cost–benefit of cloud services, vulnerabilities in these environments allow a company to gain access to industrial secrets and sensitive customer data. The authors isolate data processing in enclaves and make the data from devices available only to the company that manufactured them. The system uses enclaves at the access point, assuming that the gateway can be malicious. The architecture proposed by Ayoade et al. is not optimized since the server initializes an enclave before processing each received message and performs an attestation before each data publication. In addition, the server performs entries and exits in enclaves much more frequently. The platform offers a performance overhead more significant than the one presented by our proposed architecture.

Anciaux et al. analyze personal data management systems for private clouds and describe the requirements for providing generic computations with a threat model that includes high privileged attackers [48]. Even though the authors do not provide implementations, they provide an overview design that would leverage TrustZone in mobile devices and SGX in the cloud for managing access control. In recent work, Carpentier et al. demonstrate a prototype of the architecture proposed in [48] based on an application for rewarding employees using bikes with green bonuses [50]. The core, running on an SGX enclave, includes a Role Based Access Control (RBAC) policy enforcement module and can be extended with other enclaves, named data tasks, for specific computations. The authors do not focus on an IoT scenario, assuming that the device is a smartphone with high computing capabilities. In another recent work, the authors have shown that the separation between core and modular data tasks introduces an average overhead of 25% [51]. However, more performance evaluations concerning throughput, latency, and resource usage are necessary to confirm that the architecture yields high throughput and low latency requirements for IoT.

Unlike the papers cited above, this paper proposes a high-performance system that allows users to determine how their IoT data will be used in the cloud, even when entities with high-level privileges are malicious. This work optimizes the performance by relying on multi-threaded enclave execution and proposes a revocation protocol. In addition, this work develops a detailed security and performance analysis that confirm the advantages and limitations of enclave utilization in realistic IoT data processing scenarios. We have confirmed that our architecture is suitable for the diverse IoT settings since it does not depend on the database or communication protocol and does not require any specific functionality or minimum performance requirement for the devices. Another advantage of our system is that it applies arbitrarily complex processing to the data, which does not happen in proposals based on homomorphic cryptography, for example.

By default, SGX is not suitable for applications in which the code contains industrial secrets since enclaves do not ensure code confidentiality, as described in Section 3. Silva et al. propose a tool to protect SGX code privacy by dynamic loading code inside the enclave whenever needed [27], which may be useful for a CACIC extension. Also, we do not evaluate the performance of a computing task distributed among multiple enclaves. Some intensive tasks benefit from distributed computing, but the communication between enclaves can be computationally expansive, as presented in Section 3 [28] Mathematical proofs of security and defenses against attacks on the client side are out of our scope in this work.

7. Conclusion and future work

This paper proposed an architecture to protect access to IoT data transmitted, processed, and stored in the cloud. We considered a threat model with almost complete control over the system, such as a cloud administrator interested in obtaining some financial advantage over clients' data. The architecture detailed in the article uses trusted execution environments to address these issues since conventional encryption and intrusion detection schemes fail in this scenario. The performance analysis revealed that the enclave adds only 0.1 ms to the latency for publishing and querying data and that the system processes thousands of requests per second.

This work improves the system performance by order of magnitude by leveraging multi-threaded processing inside an enclave. We also analyzed the system under the microscope, concluding that the main performance bottleneck is the disk write time, in the case of publication, and the enclave, in the case of query. This analysis revealed that the initialization time is more significant than other procedures, confirming that initializing the enclave only on the system startup is an important design choice for performance enhancement.

In future work, we aim to extend the architecture for protecting the data within sensors and access points. We also plan to implement a system for clients to publish their own data processing codes to be protected in the cloud enclave. The performance evaluation must also be extended to analyze the use of enclaves for machine learning, deep

learning, and federated learning, given the relevance of these technologies in cloud applications. Enclave performance measurements are relevant since they help the research community optimize trusted computing systems and applications. Future performance analysis can use specialized tools for detailed microbenchmarking, revealing bottlenecks and improvement opportunities. Another promising future research direction is the implementation of distributed processing and storage architectures using enclaves. Finally, we must formally demonstrate the system's security.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Miguel Elias Mitre Campista reports financial support was provided by AXA Group.

Data availability

Manuscript provides a link to public code repository on GitHub.

Acknowledgments

This work was supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) [Finance Code 001]; CNPq [grant number E-26/211.144/2019]; FAPERJ [grant number E-26/202.689/2018]; and FAPESP [grant number 15/24494-8]. We also thank Dr. Ing. Otto C. M. B. Duarte (*in memorian*) for his essential support and tutoring that made this work possible.

References

[1] I. Lee, K. Lee, The Internet of Things (iot): Applications, investments, and challenges for enterprises, 2015, http://dx.doi.org/10.1016/j.bushor.2015.03.008.

[2] L. Gantert, M. Sammarco, M. Detyniecki, M.E.M. Campista, A supervised approach for corrective maintenance using spectral features from industrial sounds, in: 2021 IEEE 7th World Forum on Internet of Things, WF-IoT, IEEE, 2021, pp. 723–728, http://dx.doi.org/10.1109/WF-IoT51360.2021.9594966.

[3] F.M. Ortiz, M. Sammarco, L.H.M. Costa, M. Detyniecki, Applications and services using vehicular exteroceptive sensors: a survey, IEEE Trans. Intell. Veh. (2022) 1–20, http://dx.doi.org/10.1109/TIV.2022.3182218.

[4] M.M. Othman, A. El-Mousa, Internet of things & cloud computing internet of things as a service approach, in: 2020 11th International Conference on Information and Communication Systems, ICICS, 2020, pp. 318–323, http://dx.doi.org/10.1109/ICICS49469.2020.239503.

[5] E. Fernandes, J. Jung, A. Prakash, Security analysis of emerging smart home applications, in: 2016 IEEE Symposium on Security and Privacy, SP, IEEE, 2016, pp. 636–654, http://dx.doi.org/10.1109/SP.2016.44.

[6] S. Pearson, Trusted computing platforms, the next security solution, HP Labs 177 (2002).

[7] D. Shultz, When your voice betrays you, 2015, http://dx.doi.org/10.1126/science.347.6221.494.

[8] R. Shokri, V. Shmatikov, Privacy-preserving deep learning, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 1310–1321, http://dx.doi.org/10.1145/2810103.2813687.

[9] Q. Yang, H. Wang, X. Wu, T. Wang, S. Zhang, N. Liu, Secure blockchain platform for industrial iot with trusted computing hardware, 2021, http://dx.doi.org/10.1109/IOTM.001.2100043.

[10] S. Li, K. Xue, D.S. Wei, H. Yue, N. Yu, P. Hong, Secgrid: A secure and efficient sgx-enabled smart grid system with rich functionalities, IEEE Trans. Inf. Forensics Secur. 15 (2019) 1318–1330, http://dx.doi.org/10.1109/TIFS.2019.2938875.

[11] D.C.G. Valadares, M.S.L. da Silva, A.E.M. Brito, E.M. Salvador, Achieving data dissemination with security using fiware and intel software guard extensions (sgx), in: 2018 IEEE Symposium on Computers and Communications, ISCC, IEEE, 2018, pp. 1–7, http://dx.doi.org/10.1109/ISCC.2018.8538590.

[12] C. Priebe, K. Vaswani, M. Costa, Enclavedb: A secure database using sgx, in: 2018 IEEE Symposium on Security and Privacy, SP, IEEE, 2018, pp. 264–278, http://dx.doi.org/10.1109/SP.2018.00025.

[13] C.M. Franç, R.S. Couto, P.B. Velloso, Data imputation on iot gateways using machine learning, in: 2021 19th Mediterranean Communication and Computer Networking Conference, MedComNet, IEEE, 2021, pp. 1–8, http://dx.doi.org/10.1109/MedComNet52149.2021.9501243.

[14] L.A.C. Souza, G. Antonio F. Rebello, G.F. Camilo, L.C.B. Guimarães, O.C.M.B. Duarte, DFedForest: Decentralized federated forest, in: 2020 IEEE Blockchain, 2020, pp. 90–97, http://dx.doi.org/10.1109/Blockchain50366.2020.00019.

[15] G. Eibl, D. Engel, Influence of data granularity on nonintrusive appliance load monitoring, in: Proceedings of the 2nd ACM Workshop on Information Hiding and Multimedia Security, 2014, pp. 147–151, http://dx.doi.org/10.1145/2600918.2600920.

[16] L.C. Guimarães, G.A.F. Rebello, G.F. Camilo, L.A.C. de Souza, O.C. Duarte, A threat monitoring system for intelligent data analytics of network traffic, Ann. Telecommun. (2021) 1–16, http://dx.doi.org/10.1007/s12243-021-00893-5.

[17] C. Rong, S.T. Nguyen, M.G. Jaatun, Beyond lightning: A survey on security challenges in cloud computing, 2013, http://dx.doi.org/10.1016/j.compeleceng.2012.04.015, URL: https://www.sciencedirect.com/science/article/pii/S0045790612000870. Special issue on Recent Advanced Technologies and Theories for Grid and Cloud Computing and Bio-engineering.

[18] D.P. Zegzhda, E. Usov, A. Nikol'skii, E.Y. Pavlenko, Use of intel sgx to ensure the confidentiality of data of cloud users, 2017, http://dx.doi.org/10.3103/S0146411617080284.

[19] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, M. Russinovich, Vc3: Trustworthy data analytics in the cloud using sgx, in: 2015 IEEE Symposium on Security and Privacy, IEEE, 2015, pp. 38–54, http://dx.doi.org/10.1109/SP.2015.10.

[20] V. Costan, S. Devadas, Intel sgx explained, 2016, Cryptology ePrint Archive, Paper 2016/086. URL: https://eprint.iacr.org/2016/086, https://eprint.iacr.org/2016/086.

[21] D.C.G. Valadares, N.C. Will, J. Caminha, M.B. Perkusich, A. Perkusich, K.C. Gorgônio, Systematic literature review on the use of trusted execution environments to protect cloud/fog-based internet of things applications, IEEE Access 9 (2021) 80953–80969, http://dx.doi.org/10.1109/ACCESS.2021.3085524.

[22] V. Scarlata, S. Johnson, J. Beaney, P. Zmijewski, Supporting third party attestation for intel sgx with intel data center attestation primitives, 2018, White paper.

[23] R. Haakegaard, J. Lang, The elliptic curve diffie-hellman (ecdh), 2015, Online at https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf.

[24] G.A. Thomaz, M.B. Guerra, M. Sammarco, M.E.M. Campista, Cacic: Controle de acesso confiável usando enclaves a dados em nuvem da internet das coisas, in: Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, SBC, 2022, pp. 573–586, http://dx.doi.org/10.5753/sbrc.2022.222377.

[25] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, F. Mckeen, Intel software guard extensions: EPID provisioning and attestation services, White Paper 1 (2016) 119.

[26] I. Anati, S. Gueron, S. Johnson, V. Scarlata, Innovative technology for cpu based attestation and sealing, 2013.

[27] R. Silva, P. Barbosa, A. Brito, 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, IEEE, 2017, pp. 314–321.

[28] T. Elgamal, K. Nahrstedt, Serdab: An iot framework for partitioning neural networks computation across multiple enclaves, in: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID, IEEE, 2020, pp. 519–528.

[29] Y. Zhang, M. Zhao, T. Li, H. Han, Survey of attacks and defenses against sgx, in: 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference, ITOEC, IEEE, 2020, pp. 1492–1496, http://dx.doi.org/10.1109/ITOEC49072.2020.9141835.

[30] F. Brasser, U. Mü, A. Dmitrienko, K. Kostiainen, S. Capkun, A.-R. Sadeghi, Software grand exposure: Sgx cache attacks are practical, in: Proceedings of the 11th USENIX Conference on Offensive Technologies, WOOT'17, USENIX Association, USA, 2017, p. 11.

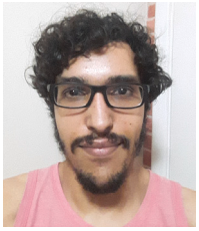
[31] A. Nilsson, P.N. Bideh, J. Brorsson, A survey of published attacks on intel sgx, 2020, pp. 1–11, http://dx.doi.org/10.48550/arXiv.2006.13598, arXiv preprint arXiv:2006.13598.

[32] J. Wang, Z. Hong, Y. Zhang, Y. Jin, Enabling security-enhanced attestation with intel sgx for remote terminal and iot, 2017, http://dx.doi.org/10.1109/TCAD.2017.2750067.

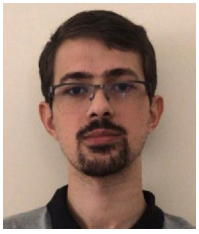
[33] L.V. Silva, P. Barbosa, R. Marinho, A. Brito, Security and privacy aware data aggregation on cloud computing, 2018, http://dx.doi.org/10.1186/s13174-018-0078-3.

[34] H. HaddadPajouh, A. Dehghantanha, R.M. Parizi, M. Aledhari, H. Karimipour, A survey on Internet of Things security: Requirements, challenges, and solutions, Internet Things 14 (2021) 100129, http://dx.doi.org/10.1016/j.iot.2019.100129.

[35] G. Karjoth, M. Schunter, M. Waidner, Privacy-enabled services for enterprises, in: Proceedings. 13th International Workshop on Database and Expert Systems Applications, IEEE, 2002, pp. 483–487.
[36] S. Sicari, A. Rizzardi, G. Dini, P. Perazzo, M. La Manna, A. Coen-Porisini, Attribute-based encryption and sticky policies for data access control in a smart home scenario: a comparison on networked smart object middleware, Int. J. Inf. Secur. 20 (2021) 695–713.
[37] P. Subramanyan, R. Sinha, I. Lebedev, S. Devadas, S.A. Seshia, A formal foundation for secure remote execution of enclaves, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 2435–2450.
[38] L. Hou, S. Zhao, X. Xiong, K. Zheng, P. Chatzimisios, M.S. Hossain, W. Xiang, Internet of Things cloud: Architecture and implementation, 2016, http://dx.doi.org/10.1109/MCOM.2016.1600398CM.
[39] Telefónica I+D official Open Source repositories, Fiware-iotagent-ul, 2016–2019, https://github.com/telefonicaid/iotagent-ul.
[40] R.S. Alonso, I. Sittón-Candanedo, Óscar García, J. Prieto, S. Rodríguez-González, An intelligent edge-iot platform for monitoring livestock and crops in a dairy farming scenario, 2020, http://dx.doi.org/10.1016/j.adhoc.2019.102047, URL: https://www.sciencedirect.com/science/article/pii/S1570870519306043.
[41] V. Araujo, K. Mitra, S. Saguna, C. Åhlund, Performance evaluation of fiware: A cloud-based iot platform for smart cities, J. Parallel Distrib. Comput. 132 (2019) 250–261, http://dx.doi.org/10.1016/j.jpdc.2018.12.010.
[42] M. Kucab, P. Boryło, P. Chołda, Remote attestation and integrity measurements with intel sgx for virtual machines, Comput. Secur. 106 (2021) 102300.
[43] Gil Tene, wrk2: a http benchmarking tool based mostly on wrk, 2015–2019, https://github.com/giltene/wrk2.
[44] O. Weisse, V. Bertacco, T. Austin, Regaining lost cycles with hotcalls: A fast interface for sgx secure enclaves, 2017, http://dx.doi.org/10.1145/3140659.3080208.
[45] N. Weichbrodt, P.-L. Aublin, R. Kapitza, Sgx-perf: A performance analysis tool for intel sgx enclaves, in: Proceedings of the 19th International Middleware Conference, 2018, pp. 201–213, http://dx.doi.org/10.1145/3274808.3274824.
[46] X. Zhou, Y. Jin, H. Zhang, S. Li, X. Huang, A map of threats to validity of systematic literature reviews in software engineering, in: 2016 23rd Asia-Pacific Software Engineering Conference, APSEC, IEEE, 2021, pp. 153–160.
[47] G. Ayoade, A. El-Ghamry, V. Karande, L. Khan, M. Alrahmawy, M.Z. Rashad, Secure data processing for Iot middleware systems, 2019, http://dx.doi.org/10.1007/s11227-018-2686-x.
[48] N. Anciaux, P. Bonnet, L. Bouganim, B. Nguyen, P. Pucheral, I.S. Popa, G. Scerri, Personal data management systems: The security and functionality standpoint, Inf. Syst. 80 (2019) 13–35.
[49] G.A. Thomaz, G.F. Camilo, L.A.C. de Souza, O.C.M. Duarte, Architecture and performance comparison of permissioned blockchains platforms for smart contracts, in: 2021 IEEE Global Communications Conference, GLOBECOM, IEEE, 2021, pp. 1–6.
[50] R. Carpentier, F. Thiant, I.S. Popa, N. Anciaux, L. Bouganim, An extensive and secure personal data management system using sgx, in: 25th International Conference on Extending Database Technology, 2022.
[51] R. Carpentier, I.S. Popa, N. Anciaux, Poster: Reducing data leakage on personal data management systems, in: 2021 IEEE European Symposium on Security and Privacy, EuroS & P, IEEE, 2021, pp. 716–718.



Guilherme Araujo Thomaz is an undergraduate student in Electronics and Computer Engineering at Federal University of Rio de Janeiro (UFRJ) and is a member of the Teleinformatics and Automation Group (GTA) research laboratory. He published national and international papers on permissioned blockchain platforms and trusted computing for Internet of Things in clouds. His interests are trusted computing, Internet of Things, federated learning, and cloud/edge computing.



Matheus Barreira Guerra is an undergraduate student at the Federal University of Rio de Janeiro coursing Electronics and Computer Engineering, Matheus Barreira Guerra is a member of the Teleinformatics and Automation Group (GTA) research laboratory. He is now working on adversarial federated learning paper that explores hardware-based solutions to attacks on AI models. His research interests are trusted computing, security, machine learning, AI, and federated learning.

Matteo Sammarco received his B.Sc. and M.Sc. degrees in Computer Engineering from Università degli Studi di Napoli Federico II, Italy, in 2008 and 2010, respectively. In 2014 he received the Ph.D. in Computer Science from Sorbonne Université, France. Currently, he a is research data scientist at Axa Group Operations, mainly working on machine learning applied to Smart Mobility, IoT, and Privacy-Preserving AI.

Marcin Detyniecki studied mathematics, physics and computer science at the Sorbonne Université Paris where he obtained his Ph.D. in AI in 2000. Between 2001 and 2014, he was a research scientist of the French National Center for Scientific Research (CNRS). He is currently Group Chief Data Scientist and Global Head of R&D at AXA Group Operations. His work focuses on Machine Learning, Artificial Intelligence, Computational Intelligence, Multimedia Retrieval, and Fair and Transparent AI.

Miguel Elias M. Campista is an associate professor at the Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil, since 2010. His major research interests include network and data science, wireless networking, and cloud and fog computing. Campista received his D.Sc. degree in Electrical Engineering from UFRJ in 2008 and spent 2012 in the LIP6 lab at Sorbonne Universit'e, Paris, France, as invited professor. He is an affiliate member of the Brazilian Academy of Sciences and an associate editor of Annals of Telecommunications, Springer.