Journal of Internet Services and Applications

RESEARCH

Open Access

Security and privacy aware data aggregation on cloud computing

Leandro V. Silva*, Pedro Barbosa, Rodolfo Marinho and Andrey Brito

Abstract
The use of cloud computing has become common due to advantages such as low cost and sizing of computing resources according to demand. However, it also raises security and privacy concerns, because critical data – for example, in IoT applications – are stored and processed in the cloud. This paper proposes a software architecture that supports multiple approaches to secure data aggregation. For validation purposes, this software architecture was used in the development of applications for smart grids, computing instantaneous consumption of a region and the monthly bill of an individual consumer. The consumption data can be collected by smart meters, enabling consumers to reduce electricity costs through close monitoring. However, such data may reveal sensitive information if no privacy techniques are applied. Therefore, the proposed software architecture proved to be viable from experiments with techniques such as homomorphic encryption and hardware security extensions (Intel SGX).

Keywords: Software architecture, Security, Privacy, Cloud computing, Homomorphic encryption, Smart grids

1 Introduction

Cloud computing is a term meaning online hosted services. These services are accessible through the Internet, metaphorically called the "cloud" [1]. From a business point of view, this computing model is very attractive. Software, hardware, and physical infrastructure expenses are drastically reduced as contracting of computational resources is done in smaller grains and the expansion or reduction of these resources can be done in an automated way, according to the demand. Moreover, the operation of the infrastructure is delegated to a provider that, due to its scale, tends to be much more efficient.

Despite all the advantages cited, the use of cloud computing brings security and privacy concerns. In a poll of the Cloud Industry Forum [2] for the United Kingdom, the two main inhibitors for cloud computing adoption are concerns about data security and privacy, mentioned by 70% and 61% of the respondents, respectively. In most cases, cloud services are provided on a shared infrastructure and, thus, additional attacks – both external and internal – can occur [3], such as stealing passwords for accessing the cloud service or exploits in the provided application programming interface (API) [4].

In certain use cases – such as smart grids – confidentiality requirements tend to be greater due to its large volume of sensitive data. In this context, sensitive data, such as the fine grained energy consumption of each consumer, should be handled safely, since they can reveal a lot of information about consumers, e.g., excerpts from the day that there are no individuals in the household, arrival and departure times, or rest periods, which reveal their behavior patterns.

This paper proposes a software architecture to enable the use of cloud computing in applications with strict security and privacy requirements. This architecture considers how the components of an application can be integrated so that the privacy and security of user data are guaranteed. The sensitive part of the processing is therefore isolated and the architecture considers different strategies for aggregating sensitive data in environments where there are no guarantees of full confidentiality.

This software architecture can be applied in many situations where sensitive data is collected and must be aggregated in order to anonymize and produce meaningful information. For example, in elections, electronic devices could be used to periodically send partial results to aggregators in the cloud in order to calculate the final voting result. Voters' identities and partial results would not leak

*Correspondence: leandro@ufcg.edu.br
Universidade Federal de Campina Grande, 58429-140 Campina Grande, Brazil

because safe aggregators would handle this information in an isolated environment.

In our work, we consider two implementations of the secure aggregation module, one based on homomorphic encryption, which is completely programmed in software, but imposes high additional processing costs, and another based on Intel SGX (Software Guard eXtensions) [5] technology, which does not impose high computational costs, but requires processors that support the software guard instruction set[1]. The first enables computations to be performed on ciphertext without compromising encryption. For example, a homomorphic search system allows locating terms in encrypted databases without disclosing any information about the database or about the term being searched. The second allows processes to be run in a protected mode, where memory and execution are protected against access, even from users or processes with higher levels of privilege, avoiding the need to perform computations in ciphertext.

In our experiments, two use cases were addressed: the calculation of instant energy consumption in a region and the calculation of consumers' monthly bill. The aggregation strategy with Intel SGX proved to be much more efficient than homomorphic encryption. However, since homomorphic encryption does not have specific hardware requirements, this strategy may be feasible for applications where the focus is not on the volume of data.

Finally, certain limitations were identified during the development of the work: (*i*) to be feasible, the set of computations for the fully homomorphic encryption is limited, which makes it much more difficult to develop arbitrary data processing modules; (*ii*) the use of Intel SGX prevents certain operations from being performed, such as system calls (syscalls), which has implications for the type of code that will run in a protected manner; and (*iii*) Intel SGX also has memory usage limitations, 128 MB per host in the current implementation, but with the ability to paginate memory.

The contributions of this research are: (*i*) a software architecture for data aggregation on the cloud composed by two secure approaches; (*ii*) a practical use of the proposed architecture in smart grids; (*iii*) a comparison and discussion about advantages, disadvantages and limitations of homomorphic encryption and Intel SGX for aggregation; and (*iv*) a performance analysis of Intel SGX in different environments – bare metal, virtual machines and Docker containers.

The rest of the paper is organized as follows. The related work is discussed in Section 2. Section 3 depicts the software architecture designed to ensure secure data aggregation. The evaluation method is explained in Section 4, the results are presented in Section 5 and are discussed in Section 6. A threat analysis of this solution is discussed in Section 7. Finally, Section 8 highlights the main conclusions of the paper and its limitations.

2 Related work

In this section, several related works on cloud architectures, privacy techniques and cryptography are discussed.

2.1 Cloud architectures and platforms

Reinhold et al. [6] describe a hybrid architecture, where part of the components are hosted in a private cloud and part in a public cloud. The storage of encrypted data is hosted in a public cloud. The application, with the logic and data processing, is hosted in a private cloud, where there are minor concerns with security and privacy. Only clients have access to private keys, so, data is only treated in its pure form when the client is authenticated. Finally, whenever data needs to be stored, it goes through an encryption server, which stays in the private cloud, and then is sent to the storage server in the public cloud.

Bohli et al. [7] study different architecture patterns for resource distribution across multiple cloud computing service providers: application replication, which allows operations to be sent across different clouds and compare if the results are the same; layers partitioning of an application into distinct clouds, to separate, for example, the logic of the application in one cloud and the database in another; partitioning the application logic into different clouds and partitioning the data into distinct clouds. The authors conclude that there is no optimal strategy for all cases, since the implementation of the suggested standards is not trivial and all have security breaches.

2.2 Privacy techniques

Possible uses of Intel SGX were discussed by Hoekstra et al. [8]. They present examples of applications that make use of the Intel SGX capabilities, as well as an application architecture considering an application partition between parts that demand security and should be executed within enclaves, and parts that do not require security, which can be run out of enclaves.

A toolkit for building protocols is presented by Barbosa et al. [9]. It extends the warranties of isolated execution environments, such as Intel SGX, by using its ability to perform remote attestation. So, it is possible to establish key exchange protocols between a remote participant and an isolated execution environment, in a secure way. These protocols are defined by the combination of a passively secure key exchange protocol and the use of an arbitrary attestation protocol.

2.3 Cryptography

On secure computing, Rivest, Adleman and Dertouzous [10] created the concept of homomorphic encryption. This is due to the fact that RSA, a cryptographic system developed by Rivest, Shamir and Adleman [11], has a partial homomorphism, called multiplicative homomorphism. Using unpadded RSA, it is possible to multiply two encrypted values and the result will still be the encrypted multiplication.

Fully homomorphic models are characterized by allowing operations of addition and multiplication in encrypted blocks, so that the value returned is an encryption of the result of the operations applied on the original data. Although the concept is not recent, these models were considered purely theoretical until Gentry [12] proposed a valid system, using ideal lattices. However, efficiency issues are still a barrier. Currently, all types of fully homomorphic encryption schemes proposed still have a long evolutionary path before being used in practice [13].

3 The software architecture

The proposed solution can be applied in a way that developers will not have to worry about every aspect of the architecture, only the aggregation part. It has four types of components: message bus, producers, aggregators and consumers. A message bus is responsible for communication between producers, aggregators and data consumers. After being produced, the data will be published on the message bus and, at some point, will be consumed and treated by aggregators, who may perform arbitrary operations, capable of perform them safely. Subsequently, the aggregate data will be consumed by applications. An illustrative schema of the architecture can be seen in Fig. 1.

All the illustrated components follow well defined interfaces, therefore, it is possible to use different types of producers, message buses, aggregators or consumers, as long as they correctly implement the specification. The following sections detail the major components of this architecture.

3.1 Message buses

A message bus is responsible for the exchange of information between **members** – producers, aggregators or consumers – in a transparent way, consequently, a producer can, for example, create messages without knowing more details, such as physical location or IP address, about the aggregators. This reduces the coupling and ensures scalability.

Each component of the architecture must subscribe to a **message topic**. All aggregators or consumers of a topic will receive the messages sent by the producers. In addition, exclusive consumption of new messages or of all messages retained from a certain topic is allowed. It is worth mentioning also that any party can send messages in a topic in order to request data, to perform intermediate steps on data. Finally, the amount of messages stored for a topic may be configurable.

This component allows the secure exchange of every message. Each party can have a certificate issued by a certification unit as a form of authentication, preventing intruders from sending or receiving messages. In addition, messages with sensitive content must be encrypted. It is the role of the aggregator, described in Section 3.3, to transform sensitive and encrypted information, such as individual consumptions measured by smart meters of households, into puretext aggregate information, such as consumption over a longer period of time or consumption of a region.

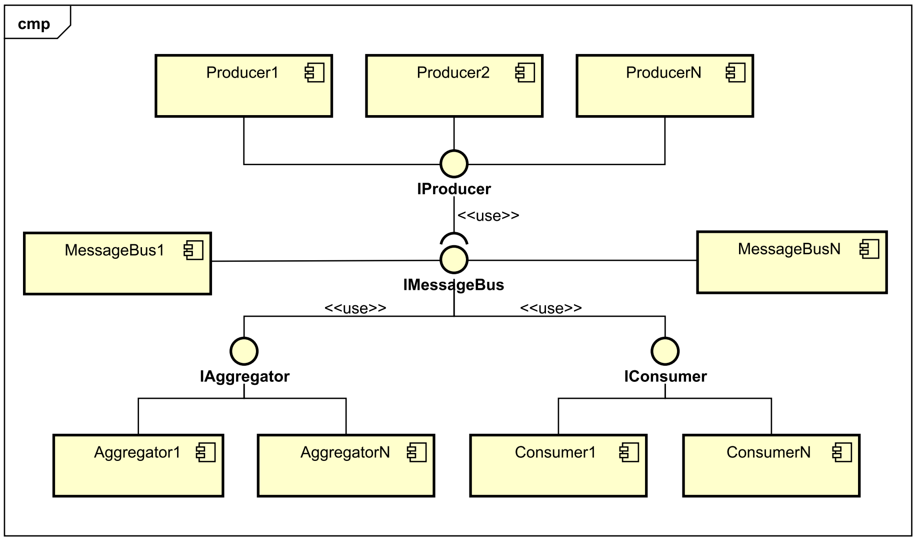

Fig. 1 Simplified schema of proposed software architecture

The basic operations supported by a message bus are:

- ***operationMode***: the operation mode can be **common** or **secure**. If it's **secure**, the message bus will require valid certificates for every member. A certificate is **valid** if it is registered in the message bus;
- ***register***: allows a certificate to be registered in the message bus to be used as an authentication form;
- ***connect***: first operation called by members before subscribing to a **topic**. If the **operation mode** is **secure**, a valid certificate is needed;
- ***subscribe***: a member can subscribe a message topic to **publish** and **consume** messages.
- ***publish***: publishes a message in a subscribed topic.
- ***consume***: consumes messages from a subscribed topic.

3.2 Producers and consumers

As stated before, **producers** are responsible for generating data, which will be published in topics on the message bus. When necessary, these data will be processed by **aggregators** and eventually will be received by **consumers** – applications, for example.

Sensitive data produced can not be used to threaten the security and privacy of their owners, so each type of producer must provide strategies to ensure this. The strategy will depend on the type of aggregator being used.

Both producers and consumers support similar operations. The list is shown below:

- ***getCertificate***: returns its public certificate to be used for authentication purposes;
- ***connectToBus***: performs the connection to the message bus before sending or receiving any data;
- ***subscribeToTopic***: contacts the message bus in order to subscribe to a topic.
- ***publishIntoBus***: publishes a message in a subscribed topic. This operation is used by **producers**.
- ***consumeFromBus***: consumes messages from a subscribed topic. Operation used by **producers** – when intermediary steps are needed for aggregation – and by **consumers**.

3.3 Aggregators

After consuming the producer's data of message topics on the message bus, it is often necessary to perform operations on them. These operations, such as sum, multiplication or grouping are performed by **aggregators**.

The purpose of this component is to transform sensitive data in order to prevent critical information from being discovered while ensuring that it is still relevant to consumers.

Any aggregation must be safe, so these components work in tandem with the producers because the data sent must be computable by the respective aggregator. For example, if a producer sends ciphertext generated by an encryption algorithm, the responsible aggregator must have means to process that.

The operations supported by this component are:

- All the operations from **producers** and **consumers**;
- **aggregate**: aggregates data sent by producers. All the data is collected using the **consumeFromBus** operation.

4 Evaluation

Two use cases were chosen for implementation as proofs of concept to evaluate the proposed solution. Both are part of the context of smart grids and have requirements regarding data confidentiality. The use cases are discussed in Sections 4.1 and 4.2, the technical aspects are explained in Section 4.3 and the developed aggregators are shown in Sections 4.4 and 4.5.

4.1 Calculation of regional energy consumption

The growing need of electric energy resources motivated both by the government and the industry, seek for alternative ways to provide energy and, especially, to improve the management of the electricity grid. On the other hand, increasing efficiency and balancing the energy grid is not a trivial task. For this, one option is to use smart meters, which can periodically measure and report energy consumption [14].

As stated earlier, periodic measurement of energy consumption causes concerns about consumer privacy, since it is possible to infer personal information from what is collected, such as appliance usages in the residence, as well as the presence and number of inhabitants [15]. In more extreme cases, it is even possible to identify the television channel being watched [16].

An alternative to provide relevant information for energy balancing, while maintaining consumer privacy, is to aggregate consumption data into groups of households, or regions.

4.2 Calculation of the monthly consumption bill

The second use case is to calculate, in a secure and private manner, the monthly consumption bill for each consumer. This approach allows the utility provider to issue invoices considering the aggregate consumptions and allows the consumers or the providers to make use of certain benefits of detailed measurement – calculation of instantaneous consumption of the region or local visualization of instant consumption by the consumer – without privacy threats, guaranteeing that detailed data will not be used by the utility or partners to infer consumer habits.

4.3 Technical aspects

For the experiments' execution, the proofs of concept were implemented following the architecture explained in Section 3. Figure 2 illustrates an instantiation of the proposed software architecture for the considered use cases.

Among the above components, smart meters (producers) and aggregators were developed during the research. For the communication between producers and aggregators/consumers, the Apache Kafka[2] solution was chosen, since it provides all the services required for the message bus, including the secure exchange of messages, allowing the authentication of those involved and the use of secure communication channels.

Each smart meter is simulated by a thread in Java that generates random consumption values and publishes on the message bus. All meters in a region r are subscribed to the same topic of the message bus. The region aggregator r is a running process that is also subscribed to the topic and is responsible for grouping the measurements for each time instant t, and, when the whole set of consumption data of the region's meter is collected for an instant t, it is possible to calculate the sum of these values.

For the calculation of the monthly consumption bill using the homomorphic aggregator, also developed in Java, each meter makes use of k public and private key pairs, one pair for each measurement, with a circular reuse according to the module of the measurement identifier. Finally, at each end of the cycle, the aggregation process is executed for that time interval.

For the Intel SGX aggregator, a fixed symmetric key, previously shared with the aggregator via the remote attestation process, is used by each meter. In addition, a nonce is generated randomly for each measurement. The secure exchange of keys between smart meters and the aggregator is explained in a detailed way in Section 4.5.

Finally, the two approaches are robust against situations where the aggregator publishes aggregated results but allows the inference of instantaneous values. In the homomorphic approach, the producer participates in the aggregation process and can identify situations where the aggregations use little data or have overlapping intervals. In the Intel SGX approach, the producer validated the attester's code initially and it can not be updated without a new attestation.

4.4 Homomorphic aggregator

This aggregator uses homomorphic encryption to perform operations in the data in a secure and private manner. Many asymmetric algorithms have homomorphic properties, such as unpadded RSA [10] and Paillier [17], described in Appendix. To be more precise, unpadded RSA is a multiplicative homomorphic cryptosystem, while Paillier is an additive homomorphic cryptosystem.

Although unpadded RSA and Paillier present interesting homomorphic properties, the presented case study here considers an approach based on the ElGamal cryptosystem – aggregations using other homomorphic encryptions could be performed, however. For example, Garcia et al. [18] and Erkin et al. [19] present approaches for aggregating energy consumption measurements using variations of the Paillier cryptosystem.

The approach used in our case study is based on the scheme proposed by Busom et al. [20], which applies the ElGamal cryptography system [21]. By default, it has the multiplicative homomorphic property but a small

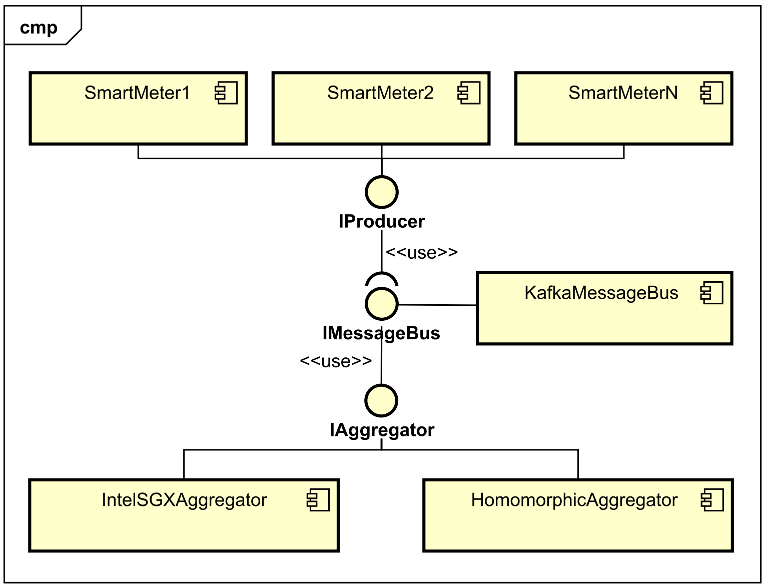

Fig. 2 Schema of the application for calculating aggregate values of energy consumption

modification enables addition, as seen in [22]. In addition, it is possible to combine it with a threshold encryption scheme [23], where all involved parties need to collaborate to decrypt the ciphertext.

The ElGamal cryptosystem proceeds as follows:

- *Set up*: a large prime q is chosen. Next, a generator g of the cyclic group $\mathbb{Z}_q^*$ is selected.
- *Key generation*: a secret key x is generated by setting its value as a random number $x \in_R \mathbb{Z}_q^*$. The corresponding public key is computed as $y = g^x$.
- *Encryption*: a message $m \in G$ is encrypted under public key y by taking a random number $r \in_R \mathbb{Z}_q^*$ and computing $c = g^r$ and $d = m \cdot y^r$. The ElGamal encryption of m under public key $y, E_y(m)$, is the tuple (c, d).
- *Decryption*: a ciphertext $E_y(m)$ is decrypted using the private key x by computing $m = d \cdot c^{-x}$.

Given messages m_1 and m_2, we can obtain an encryption of $m_1 \cdot m_2$ by computing:

$$\begin{aligned} E_y(m_1) \cdot E_y(m_2) &= (c_1 \cdot c_2, d_1 \cdot d_2) \\ &= \left(g^{r_1+r_2}, m_1 \cdot m_2 \cdot y^{r_1+r_2}\right) \\ &= E_y(m_1 \cdot m_2). \end{aligned}$$

Hence, ElGamal is a multiplicative homomorphic cryptosystem.

To calculate the total consumption in a region, Busom et al. [20] propose a protocol which uses an additive ElGamal cryptosystem. Given $E_y\left(g^{m_1}\right)$ and $E_y\left(g^{m_2}\right)$, then, $E_y\left(g^{m_1}\right) \cdot E_y\left(g^{m_2}\right) = E_y\left(g^{m_1} \cdot g^{m_2}\right) = E_y\left(g^{m_1+m_2}\right)$.

For this type of aggregation, each producer $p \in [1, n]$, where n is the number of producers in a common topic, must have the following items:

- A large prime number q (2048 bits minimum) and a generator g of order q from the multiplicative group G of $\mathbb{Z}_q^*$;
- A private key x_p;
- A public key $y_p = g^{x_p}$;
- A certificate $cert_p$, to be validated by a certification unit;

A configuration phase is mandatory whenever new producers subscribe to a message thread. The procedure is described below:

1. The aggregator sends a configuration request message;
2. Each producer sends y_p and $cert_p$ to the topic;
3. The aggregator checks the validity of each $cert_p$ and inserts a message with $\{y_1, ..., y_n\}$ and $\{cert_1, ..., cert_n\}$ on the topic;
4. Each producer checks the validity of each $cert_p$ and calculates a global public key $y = \prod_{p=1}^{n} y_p$.

The following steps will be performed whenever a data collection is needed for aggregation:

1. The aggregator sends a data request message or, alternatively, the producers can initiate a transmission periodically;
2. Each producer p generates a random number $z_p \in \mathbb{Z}_q^*$ and calculates $C_p = E_y\left(g^{v_p+z_p}\right) = (c_p, d_p)$, where v_p represents the value collected by p and the E_y represents the ElGamal encryption function;
3. All values C_p are published in the message bus topic associated with that producer (for example, the region where the meter is installed);
4. The aggregator performs its computation: $C = \left(\prod_{p=1}^{n} c_p, \prod_{p=1}^{n} d_p\right)$ and inserts C in the topic;
5. Each producer calculates $T_p = c^{x_p}.g^{z_p}$ and publishes it in the topic;
6. The aggregator is able to compute $D = d.\left(\prod_{p=1}^{n} T_p\right)^{-1}$, where $d = \prod_{p=1}^{n} d_p$;
7. Finally, it is possible to obtain $V = \sum_{p=1}^{n} v_p$ by calculating $\log_g D$;
8. The aggregator publishes the result obtained in the same or another topic of the message bus, so that it is available to potential consumers.

Since it's a secure multiparty process with multiple steps, the raw data generated by every producer would leak only if all the other participants were compromised. This security is increased when more participants exist.

4.5 Intel SGX aggregator

The Intel SGX aggregator uses the technology of same name to provide security and privacy of the sensitive data by aggregating it in protected areas of memory (**enclaves**), inaccessible even by users with high privileges. In our implementation, we used the AES Galois/Counter Mode (AES-GCM) symmetric encryption algorithm described in [24] – key size of 128-bit – for the exchange of confidential messages between the producers and the aggregator.

In order to agree on the 128-bit key to be used for secure communication, each producer $p \in [1, n]$ of a topic communicates with and attests the aggregator, i.e., verifies that the correct aggregator has been established in an SGX enclave, by performing the **Remote Attestation** process described in [25]. In a nutshell, the Remote Attestation process leverages SGX capabilities to produce a criptographic signature of the contents of SGX enclaves and to digitally sign it using a key that is only accessible by the platform processor, and uses an external attestation service – currently, only an attestation service provided by Intel (**IAS**) can be used – to verify that the produced information indeed came from the expected SGX enclave. The Remote Attestation process also has an underlying key exchange scheme based on the elliptic curve Diffie-Hellman (ECDH) key exchange protocol. As a result of this process, each producer will have both (*i*) successfully verified the integrity of the aggregator and (*ii*) derived the 128-bit key k_p used for secure communication – which is also derived by the aggregator inside the enclave.

When receiving encrypted confidential data, the aggregator can decrypt the message using the same AES-GCM algorithm, and then perform aggregation on the decrypted sensitive data. The security and privacy of the encrypted data is achieved through the guarantees provided by the SGX enclaves [5].

The following steps are performed to aggregate collected data at each time instant:

1. The aggregator sends a data request message or, alternatively, the producers can initiate a transmission periodically;
2. Each producer p creates a random value (**nonce**) n_p, and calculates $C_p, M_p = G_e\left(v_p, n_p, k_p\right)$ where C_p is the value collected by p after the AES-GCM encryption, M_p is the message authentication code of v_p, which is the value measured by p, and the function G_e is the AES-GCM function in encryption mode.
3. Each producer publishes C_p, M_p and n_p to the topic;
4. The aggregator calculates $V = \sum_{p=1}^{n} v_p$ by computing $\sum_{p=1}^{n} G_d\left(C_p, n_p, k_p, M_p\right)$, where the function G_d is the AES-GCM function in the decryption mode, that also verifies the integrity of each of the messages received in the topic;
5. The aggregator publishes the result obtained on the same or another topic of the message bus, so that it becomes available to potential consumers.

The sensitive data processed by this aggregator is free of information leakage because the memory inside an SGX enclave is encrypted. Also, all the data is exchanged using a secure connection, established by the **Remote Attestation** process. This process also ensures that the running code inside an SGX enclave was not modified.

5 Results

After the proofs of concept were developed, experiments were executed in order to evaluate the feasability of every aggregator. The chosen metric to compare the aggregators was **response time**, because many cloud providers charge for the time their resources were used. Tests were made to measure response times for approaches using the homomorphic and the Intel SGX aggregator, and also using an aggregator without security or privacy guarantees – important to measure the overhead imposed by the secure aggregators.

The tests were executed using a machine with Intel Kaby Lake Processor (7th generation) i7 7700HQ and 16 GB of RAM. The chosen operating system was Ubuntu 16.04 and, depending of the aggregator, the experiments were ran using the host, using Docker Containers and also using Virtual Machines (VMs).

For the Intel SGX aggregator, the KVM SGX [3] and QEMU SGX [4] were installed to allow virtual machines to use SGX instructions.

Table 1 contains the description, input parameters and the environment for every test conducted. For the unencrypted and the Intel SGX Aggregator, the number of households varied from 50 to 1500 to show how these aggregators perform in small, medium and big regions. In preliminar experiments, we verified that these intervals covered a sufficient range to determine if the response time grows linearly or exponentially depending of the number of households. For the Homomorphic Aggregator, the upper bound was 250 because values higher than that would lead to very long duration tests.

The purpose of the first experiment, shown in Fig. 3, was to evaluate the time required for using Intel SGX in different environments. The response time overhead imposed by the Intel SGX technology is notable, specially when used in virtual machines, as shown in Fig. 4. On the other hand, the cost imposed by Docker Containers is barely noticeable.

The Homomorphic Aggregator results can be seen in Fig. 5. We decided to use two measurements' interval to show that, for smaller periods, this aggregator may not be a good choice. It's better to use this approach when data is collected in longer periods of time.

To sum up, Table 2 shows a comparison between the two developed secure aggregators. Despite the high overhead of Homomorphic Aggregator, it may be a good option when a cloud provider doesn't support the Intel SGX technology. A detailed discussion is in the next section.

6 Discussion

As it can be seen in the graphs and the table presented in the previous section, the data aggregation approach using Intel SGX yields much lower response times than the homomorphic encryption approach. Nevertheless, each aggregation approach has advantages and disadvantages which are discussed below.

Despite being slower, homomorphic encryption has advantages that make it viable in certain applications. No

Table 1 The test case, environment, aggregator and input parameters used for every test

Aggregator	Test case	Environment	Number of households	Total simulated time	Measurements' interval
Unencrypted	Region consumption	Host, VM, Container	50, 100, 200, 250, 500, 750, 1000, 1250, 1500	1 day	60 s
Unencrypted	Monthly bill	Host, VM, Container	50, 100, 200, 250, 500, 750, 1000, 1250, 1500	30 days	60 s
Intel SGX	Region consumption	Host, VM, Container	50, 100, 200, 250, 500, 750, 1000, 1250, 1500	1 day	60 s
Intel SGX	Monthly bill	Host, VM, Container	50, 100, 200, 250, 500, 750, 1000, 1250, 1500	30 days	60 s
Homomorphic	Region consumption	Host	10, 50, 100, 200, 250	1 day	60, 900 s
Homomorphic	Monthly bill	Host	10, 50, 100, 200, 250	30 days	60, 900 s

specific hardware is demanded for its execution, making it easily deployable in various environments. Besides that, it can be implemented in different programming languages, since it is purely based in mathematics. Depending on the amount of data to be computed, its cost in terms of time can be negligible compared to the benefits of privacy and security enabled by this approach, once all computation is done over ciphertext, making all contents unknown to the service provider and any potential attackers.

Another clear advantage of the approach used for homomorphic aggregation is that the data consumer is unable to infer the individual measurements of a specific consumer, because it would be necessary that all other consumers were rogue ones. This is characteristic of threshold encryption.

On the other hand, the homomorphic approach has a limitation related to the operations that can be done over ciphertext, given that most cryptography algorithms are partially homomorphic, and the ones that allow arbitrary computations over ciphertext, like the one presented in [12], still demand very high processing times.

Another negative factor observed in homomorphic aggregation is that an additional message exchange is needed in order to compute the regional energy consumption. This is not trivially deployable because sending messages from the utility provider to the meters requires the meter to have an accessible address or to use a polling strategy to periodically verify if there are new messages, as in the second phase of the algorithm.

Intel SGX, in turn, has high performance and a low additional cost, if compared to the use of homomorphic encryption, because it does not need to perform any computation over ciphertext, whilst providing security and privacy guarantees to users' data, without demanding great efforts to implement applications that use the technology, given that Intel SGX has libraries that support creating, attesting and using enclaves. In that way, its use is interesting when needed hardware is available and there is a need for execution speed and privacy and security of the data involved.

Nevertheless, there are still some open questions regarding the complete integration between the Intel SGX solution and the cloud computing environment philosophy. While cloud computing solutions are based on the complete decoupling between physical hardware and applications, for instance through hardware

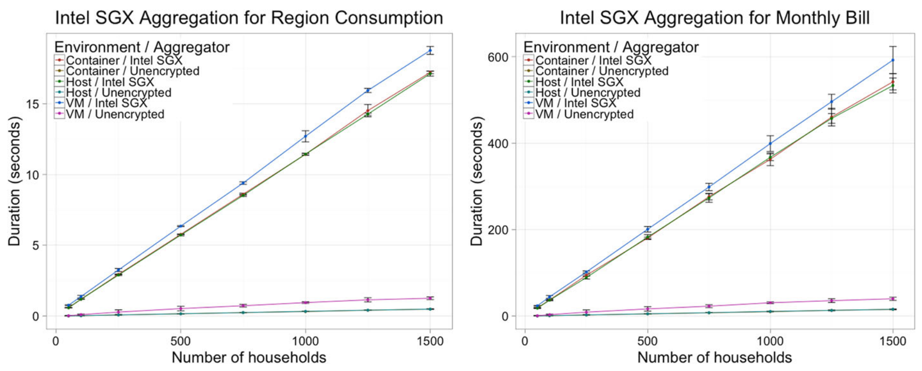

Fig. 3 Comparison between unencrypted and Intel SGX aggregators for region consumption and monthly bill calculation using host, virtual machines and docker containers

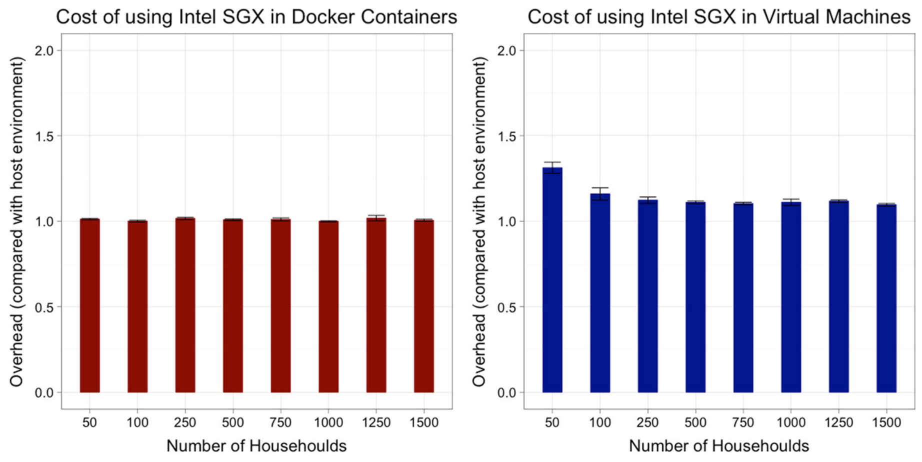

Fig. 4 Cost of using Intel SGX in docker containers and virtual machines

virtualization, Intel SGX directly binds the security of an application to the attestation process, which in turn demands Intel's participation with its attestation service.

Another negative point concerning the use of Intel SGX is its limited API support, which requires the development of applications in C/C++ and the impossibility, due to security reasons, of executing system calls from code executed within enclaves, which in turn may difficult the portability of applications to be used with the technology. Lastly, another point that must be taken into account by any application that uses Intel SGX is that there is currently a hard limit of only 128 *MB* of memory that can be used for creating and executing enclaves per host hardware. Swapping can be used to encrypt memory before exporting it outside the processor, but this imposes additional loads.

7 Threat analysis

The adversary model of the proposed software architecture considers the possibility of an attacker in obtaining individual and sensitive values from the aggregators. In this section, we discuss possible weaknesses in both components: Intel SGX and homomorphic aggregators.

7.1 Vulnerabilities in Intel SGX

With the Intel SGX aggregator, to be able to obtain individual values, the adversary needs to extract such data from the enclave. Possibilities for this are described below.

- **Side channel attacks**: SGX protects against many types of attacks, even from privileged users and softwares. However, a side-channel adversary is able to gather statistics from the CPU regarding execution and may be able to use them to deduce characteristics of the software being executed and the data being processed (side-channel analysis). Examples of analyses are power statistics, performance statistics including platform cache misses, branch statistics via timing, and information on pages accessed via page tables. It is well documented that SGX does not

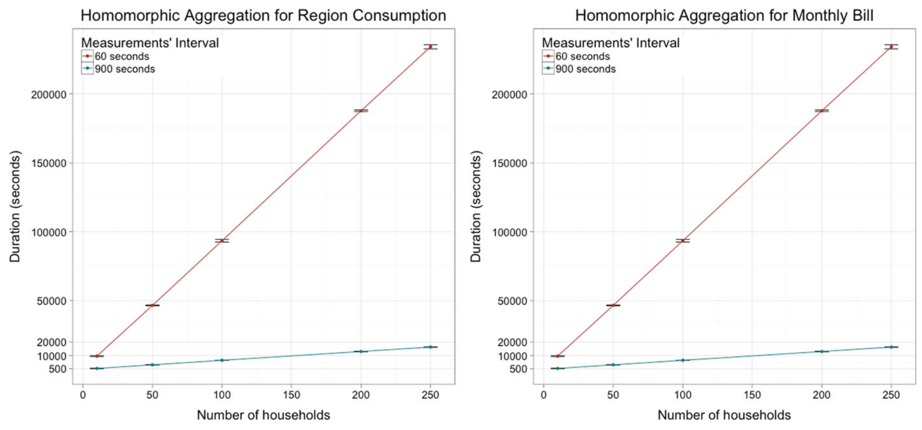

Fig. 5 Homomorphic aggregation for region consumption and monthly bill using host environment

Table 2 Comparison of Intel SGX and homomorphic aggregators. The response times are in seconds

Test case	Households	Intel SGX		Homomorphic		
		Total time	One measurement	Total time	One measurement	Overhead
Region consumption	50	0.584 ± 0.008	< 0.001	1523 ± 12.601	1.058	2608
Region consumption	100	1.186 ± 0.009	< 0.001	3018.6 ± 13.675	2.096	2545
Region consumption	200	2.428 ± 0.062	0.002	6060 ± 45.299	4.208	2496
Monthly bill	50	17.6 ± 1.110	< 0.001	46510.6 ± 319.974	1.077	2643
Monthly bill	100	36.2 ± 1.360	< 0.001	93412.8 ± 940.763	2.162	2580
Monthly bill	200	75.2 ± 2.964	0.002	187803.8 ± 564.790	4.347	2497

defend against side-channel adversaries [26, 27].

- **Software vulnerabilities**: SGX components are complex and unlikely to be bug-free, like any other software. There are drivers, libraries, dependencies, and complex instructions available for developers. Moreover, enclave developers may make mistakes and even the so called protected areas may contain common vulnerabilities like stack based buffer overflows, dangling pointers and uncontrolled format strings. This problem is boosted by the limited portion of memory of of the enclave because it affects the effectiveness of the Address Space Layout Randomization (ASLR).
  ASLR is a security technique involved in protection from many types of attacks. In order to prevent an attacker from reliably jumping to, for example, a particular exploited function in memory, ASLR randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries. With SGX, because the memory space for an enclave is quite small, a simple brute forcing mechanism can easily identify the correct address. We made experiments and observed that for different executions, an element has its addresses changed by only two bytes, meaning that the randomization is for approximately only 65536 possibilities. This is very small, considering that an attacker can, for example, increase the attack success probability through the injection of NOP slides before a malicious code [28].
- **Decrypting**: To obtain individual and sensitive data, attackers may try to break the secure communication channel established through the remote attestation process. The exchange of an ephemeral symmetric key during the remote attestation process is performed through the execution of an elliptic curve Diffie–Hellman (ECDH) handshake. Therefore, vulnerabilities rely on the possibility to solve the elliptic curve discrete logarithm problem.
  Man-in-the-middle attacks are mitigated through the usage of certificates authenticated by Intel during the remote attestation process.
  Symmetric encryption is used in the secure communication (AES GCM or CTR) and also by the Memory Encryption Engine to encrypt the enclave using AES CTR. Attackers can try to decrypt them by brute forcing and them obtain individual and sensitive data. However, at present, such attack is not computationally feasible and there is no known practical attack that would allow someone without knowledge of the key to read encrypted data when such algorithms are correctly implemented. AES has proven to be a reliable cipher, and the only practical successful attacks against AES have leveraged side channel attacks on weaknesses found in the implementation or key management of specific AES-based encryption products.

7.2 Vulnerabilities in homomorphic encryption

With the homomorphic aggregator, the adversary needs to decrypt the data. Possibilities for this are described below.

- **Solving a computationally intractable problem**: Asymmetric encryption algorithms which homomorphic properties rely on the difficulty in solving a problem in polynomial time. For example, unpadded RSA and ElGamal provide multiplicative homomorphic properties. RSA is based on the difficulty in factoring an integer [11] and ElGamal is based on the difficulty in computing a discrete logarithm in polynomial time [21]. Paillier is another asymmetric encryption algorithm based on the difficulty in factoring an integer, but its homomorphic property is additive [17].
  There are many algorithms to compute the discrete logarithm (but none of them run in polynomial time). One is the Pollard's rho algorithm for logarithms [29]. For factoring integers, the most efficient known is the general number field sieve (GNFS) [30].
- **Exploiting implementation pitfalls**: Since their publications, the asymmetric encryption systems have been analyzed for vulnerability by many researchers. They mostly illustrate the dangers of improper use of the algorithms. Indeed, securely implementing them is a nontrivial task. Dan Boneh

[31] presents some of these attacks on RSA and describes the underlying mathematical tools they use. For example, when using RSA, Wiener [32] shows that a small private exponent d results in a total break of the cryptosystem. A consequence of small private exponent d is that the public exponent e is large (recall that $e \cdot d \ mod \ N = 1$). On the other hand, when the public exponent e is too small, the encryption may be susceptible to attacks based on the Coppersmith theorem [33].
There are many other RSA attacks, such as Hastad's Broadcast attack [34], Franklin-Reiter Related Message attack [35], and Partial Key Exposure attack [36]. Other cryptosystems with homomorphic properties such as ElGamal and Paillier, when implemented improperly, also suffer from similar vulnerabilities.

8 Conclusion

Software requirements like security and privacy should not be ignored by applications that handle sensitive data and use cloud computing. In this paper, we describe a software architecture to address such requirements. This architecture allows the use of different strategies for private data aggregation. These strategies include the use of homomorphic encryption or technologies like Intel SGX.

For our evaluation, we identified two use cases involving concerns with the mentioned requirements. Given the use cases, proof of concept applications were developed in order to identify the advantages and disadvantages of each aggregation method. For homomorphic encryption, the main advantage identified was the viability to implement in any environment, although it is much less efficient. Intel SGX, on the other hand, used for the first time in a cloud computing orchestrator, yields much lower response times and allows performing various forms of computation on the sensitive data, but it demands a specific infrastructure from the service provider.

Because of the compromises that need to be made, natural future work include how to guide developers into the selection of the best approach for their application. Another direction is to combine different technologies could work together to strengthen privacy and security. One approach would be to combine different strategies with secure multi-party computation [37], to avoid that a compromised technology (e.g., SGX) would not impact on the overall application.

Endnotes

[1] Some examples of processors that support Intel SGX are the sixth and seventh generation processors of the Intel Core family and some recent Intel Xeon processors, such as the E3-1200 v5 family.

[2] http://kafka.apache.org

[3] https://github.com/01org/kvm-sgx/wiki

[4] https://github.com/01org/qemu-sgx/wiki

Appendix

The homomorphic aggregator presented in Section 4.4 uses a scheme proposed by Busom et al. [20], but other approaches are presented next.

Unpadded RSA cryptosystem

- *Set up*: two large primes q and p are chosen. Next, compute $N = p \cdot q$.
- *Key generation*: compute $\phi(N) = (p-1) \cdot (q-1)$. Next, select a prime number as a public exponent e such that $e \in [3, \phi(N))$ and $gcd(e, \phi(N)) = 1$, where gcd means the greatest common divisor. Also, compute the private exponent d such that $e \cdot d \equiv 1$. The public key is composed by (e, N) and the corresponding private key is composed by (d, N).
- *Encryption*: a message $m < N$ is encrypted under the public key by computing the ciphertext $E(m) = c = m^e \ mod \ N$.
- *Decryption*: a ciphertext $E(m)$ is decrypted using the private key, computing $m = c^d \ mod \ N$.

Given messages m_1 and m_2, we can obtain an encryption of $m_1 \cdot m_2$ by computing:

$$\begin{aligned} E(m_1) \cdot E(m_2) &= m_1^e \cdot m_2^e \ mod \ N \\ &= (m_1 \cdot m_2)^e \ mod \ N \\ &= E(m_1 \cdot m_2). \end{aligned}$$

Hence, unpadded RSA is a multiplicative homomorphic cryptosystem.

Paillier cryptosystem

Another asymmetric encryption algorithm is Paillier [17]. It works as follows:

- *Set up*: two large primes p and q are chosen, $N = p \cdot q$, and $\lambda = lcm(p-1, q-1)$, where lcm is the lower common multiple. A random number $g \in_R \mathbb{Z}^*_{N^2}$ is chosen in such a way that $gcd(b, N) = 1$, where $b = L\left(g^\lambda \ mod \ N^2\right)$ and $L(u) = \frac{(u-1)}{N}$.
- *Key generation*: let μ be the modular multiplicative inverse of b modulo N, i.e., $\mu = b^{-1} \ mod \ N$. Thus, the public key is $P_k = (N, g)$ and the private key is $S_k = (N, \lambda, \mu)$.
- *Encryption*: a message m is encrypted under public key P_k by taking a random number $r \in_R \mathbb{Z}^*_{N-1}$ and computing $E_{P_k}(m) = g^m \cdot r^N \ mod \ N^2$.
- *Decryption*: a ciphertext $c = E_{P_k}(m)$ is decrypted using the private key S_k by computing $m = L\left(c^\lambda \ modN^2\right) \cdot \mu \ modN$.

Given messages m_1 and m_2, we can obtain an encryption of $m_1 + m_2$ by computing:

$$\begin{aligned} E_{P_k}(m_1) \cdot E_{P_k}(m_2) &= g^{m_1} \cdot r_1^N \cdot g^{m_2} \cdot r_2^N \ mod\ N^2 \\ &= g^{m_1+m_2} \cdot (r_1 \cdot r_2)^N \ mod\ N^2 \\ &= E_{P_k}(m_1 + m_2). \end{aligned}$$

Hence, Paillier is an additive homomorphic cryptosystem.

Abbreviations
Intel SGX: Intel software guard eXtensions; VM: Virtual machine

Acknowledgements
We thank Amanda Souza and Gabriel Flores for providing an Intel SGX capable Docker container. A previous version of this paper was presented at SBSEG 2016 [38].

Funding
This research was partially funded by EU-BRA SecureCloud project (MCTI/RNP 3rd Coordinated Call) and by CNPq, Brazil.

Authors' contributions
LVS designed the software architecture, aided by AB. PB provided a code implementation for the chosen homomorphic encryption approach and discussed possible attacks to the proposed solution. LVS and RM developed the applications used in the experiments. RM also programmed the Intel SGX Remote Attestation process. All authors read and approved the final manuscript.

Ethics approval and consent to participate
No need. All data used was randomly generated, since it would not affect the experiments.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Received: 15 August 2017 Accepted: 22 January 2018
Published online: 02 April 2018

References

1. Markovic DS, Zivkovic D, Branovic I, Popovic R, Cvetkovic D. Smart power grid and cloud computing. Renew Sust Energ Rev. 2013;24:566–77.
2. Cloud Industry Forum. UK cloud adoption snapshot & trends for 2016: The business case for cloud. 2015. https://www.outsourcery.co.uk/media/1180/cloud-industry-forum-paper-15.pdf. Accessed 17 Aug 2017.
3. Pasupuleti SK, Ramalingam S, Buyya R. An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing. J Netw Comput Appl. 2016;64:12–22.
4. Younis YA, Merabti M, Kifayat K. Secure Cloud Computing for Critical Infrastructure : A Survey. In: The 14th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting. United Kingdom: Liverpool John Moores University; 2013.
5. McKeen F, Alexandrovich I, Berenzon A, Rozas CV, Shafi H, Shanbhogue V, Savagaonkar UR. Innovative instructions and software model for isolated execution. In: HASP '13 The Second Workshop on Hardware and Architectural Support for Security and Privacy. New York: ACM; 2013. p. 10.
6. Reinhold P, Benn W, Krause B, Goetz F, Labudde D. Hybrid cloud architecture for software-as-a-service provider to achieve higher privacy and decrease security concerns about cloud computing. In: Conf. Cloud Computing, GRIDs, and Virtualization (IEEE, 2014). Venice: The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization CLOUD COMPUTING; 2014. p. 94–9.
7. Bohli JM, Gruschka N, Jensen M, Iacono LL, Marnau N. Security and privacy-enhancing multicloud architectures. IEEE Trans Dependable Secure Comput. 2013;10(4):212–24.
8. Hoekstra M, Lal R, Pappachan P, Phegade V, Del Cuvillo J. Using innovative instructions to create trustworthy software solutions. In: HASP '13 The Second Workshop on Hardware and Architectural Support for Security and Privacy. New York: ACM; 2013. p. 11.
9. Barbosa M, Portela B, Scerri G, Warinschi B. Foundations of hardware-based attested computation and application to sgx. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). Congress Center Saar, Saarbrücken: IEEE; 2016. p. 245–60.
10. Rivest RL, Adleman L, Dertouzos ML. On data banks and privacy homomorphisms. Found Secure Comput. 1978;4(11):169–80.
11. Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Commun ACM. 1978;21(2):120–6.
12. Gentry C. A fully homomorphic encryption scheme: PhD thesis, Stanford University; 2009.
13. Naehrig M, Lauter K, Vaikuntanathan V. Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. New York: ACM; 2011. p. 113–24. CCS'11 the ACM Conference on Computer and Communications Security.
14. Erkin Z, Tsudik G. Private computation of spatial and temporal power consumption with smart meters. In: Proceedings of the 10th international conference on Applied Cryptography and Network Security. Singapore: Springer; 2012. p. 561–577.
15. Anderson R, Fuloria S. On the security economics of electricity metering. In: The Eighth Workshop on the Economics of Information Security (WEIS 2009). London: Citeseer; 2010.
16. Greveler U, Glösekötterz P, Justusy B, Loehr D. Multimedia content identification through smart meter power usage profiles. In: Proceedings of the International Conference on Information and Knowledge Engineering (IKE). Las Vegas: WorldComp; 2012. p. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).
17. Paillier P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. Berlin: Springer; 1999, pp. 223–38.
18. Garcia FD, Jacobs B. Privacy-Friendly Energy-Metering Via Homomorphic Encryption. In: Security and Trust Management 6th International Workshop, STM 2010. Athens: Springer; 2010. p. 226–38.
19. Erkin Z, Tsudik G. Private Computation of Spatial and Temporal Power Consumption with Smart Meters. In: Proc. of the 10th Int. Conf. on Applied Cryptography and Network Security (ACNS). Singapore: ACNS 2012; 2012. p. 561–77.
20. Busom N, Petrlic R, Sebé F, Sorge C, Valls M. Efficient smart metering based on homomorphic encryption. Comput Commun. 2016;82:95–101.
21. ElGamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. In: Proceedings of EUROCRYPT 84. A Workshop on the Theory and Application of Cryptographic Techniques. Paris: Springer; 1984. p. 10–18.
22. Cramer R, Gennaro R, Schoenmakers B. A secure and optimally efficient multi-authority election scheme. Eur Trans Telecommun. 1997;8(5): 481–90.
23. Saroj SK, Chauhan SK, Sharma AK, Vats S. Threshold cryptography based data security in cloud computing. In: Computational Intelligence & Communication Technology (CICT), 2015 IEEE International Conference On. India: IEEE; 2015. p. 202–7.
24. Dworkin MJ. Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. Technical report, Gaithersburg, MD, United States; 2007.
25. Anati I, Gueron S, Johnson S, Scarlata V. Innovative technology for cpu based attestation and sealing. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, vol. 13. Tel-Aviv, Israel: HASP '13 The Second Workshop on Hardware and Architectural Support for Security and Privacy; 2013.
26. Xu Y, Cui W, Peinado M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In: Proceedings of the 2015 IEEE Symposium on Security and Privacy. Fairmont San Jose: IEEE; 2015. p. 640–56.
27. Brasser F, Müller U, Dmitrienko A, Kostiainen K, Capkun S, Sadeghi A. Software grand exposure: SGX cache attacks are practical. CoRR abs/1702.07521. 2017:2–6.
28. Boneh D. Basic Control Hijacking Attacks. Accessed 14 Aug 2017. crypto.stanford.edu/cs155/lectures/02-ctrl-hijacking.pdf.
29. Pollard JM. Monte Carlo methods for index computation mod p. Math Comput. 1978;32:918–24.

30. Pomerance C. A tale of two sieves. Notices Amer Math Soc. 1996;43: 1473–85.
31. Boneh D. Twenty years of attacks on the rsa cryptosystem. Not of the AMS. 1999;46:203–13.
32. Wiener MJ. Cryptanalysis of short rsa secret exponents. IEEE Trans Inf Theor. 2006;36(3):553–8.
33. Coppersmith D. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. J Cryptol. 1997;10(4):233–60.
34. Hastad J. Solving simultaneous modular equations of low degree. SIAM J Comput. 1988;17(2):336–41.
35. Coppersmith D, Franklin M, Patarin J, Reiter M. Low-exponent rsa with related messages. In: Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques. EUROCRYPT'96. Berlin: Springer; 1996. p. 1–9.
36. Boneh D, Durfee G, Frankel Y. An attack on rsa given a small fraction of the private key bits. In: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology. ASIACRYPT '98. London: Springer; 1998. p. 25–34.
37. Cramer R, Damgård I, Maurer U. General secure multi-party computation from any linear secret-sharing scheme. In: International Conference on the Theory and Applications of Cryptographic Techniques. Bruges: Springer; 2000. p. 316–34.
38. Silva L, Silva R, Brito A, Barbosa P. Agregação de dados na nuvem com garantias de segurança e privacidade. In: Anais do XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Niterói, Rio de Janeiro: Sociedade Brasileira de Computação; 2016. p. 240–53.