Brito *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2019) 8:19
https://doi.org/10.1186/s13677-019-0141-z

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Secure end-to-end processing of smart metering data

Andrey Brito[1*], Christof Fetzer[2], Stefan Köpsell[2*], Peter Pietzuch[3], Marcelo Pasin[4], Pascal Felber[4], Keiko Fonseca[5], Marcelo Rosa[5], Luiz Gomes-Jr.[5], Rodrigo Riella[6], Charles Prado[7], Luiz F. Rust[7], Daniel E. Lucani[8], Márton Sipos[8], László Nagy[8] and Marcell Fehér[8]

Abstract

Cloud computing considerably reduces the costs of deploying applications through on-demand, automated and fine-granular allocation of resources. Even in private settings, cloud computing platforms enable agile and self-service management, which means that physical resources are shared more efficiently. Cloud computing considerably reduces the costs of deploying applications through on-demand, automated and fine-granular allocation of resources. Even in private settings, cloud computing platforms enable agile and self-service management, which means that physical resources are shared more efficiently. Nevertheless, using shared infrastructures also creates more opportunities for attacks and data breaches. In this paper, we describe the SecureCloud approach. The SecureCloud project aims to enable confidentiality and integrity of data and applications running in potentially untrusted cloud environments. The project leverages technologies such as Intel SGX, OpenStack and Kubernetes to provide a cloud platform that supports secure applications. In addition, the project provides tools that help generating cloud-native, secure applications and services that can be deployed on potentially untrusted clouds. The results have been validated in a real-world smart grid scenario to enable a data workflow that is protected end-to-end: from the collection of data to the generation of high-level information such as fraud alerts.

Keywords: Cloud computing, Security, Trusted execution, Smart grids, Privacy, Confidential computing

Introduction

Cloud computing has emerged as the main paradigm for managing and delivering services over the Internet. The automated, elastic and fine-granular provisioning of computing resources reduces the barrier for the deployment of applications for both small and large enterprises. Nevertheless, confidentiality, integrity and availability of applications and their data are of immediate concern to almost all organizations that use cloud computing. This is particularly true for organizations that must comply with strict confidentiality, availability and integrity policies, including society's most critical infrastructures, such as finance, utilities, health care and smart grids.

The SecureCloud project focused on removing technical impediments to dependable cloud computing, overtaking the barriers to a broader adoption of cloud computing. Therefore, the project developed technologies that can help secure computing resources to be provided quickly, by using familiar tools and paradigms to derive meaningful, actionable information from low-level data in a secure and efficient fashion. With this goal, the SecureCloud project makes use of state-of-the-art technologies such as OpenStack[1], Kubernetes[2], and Intel SGX [1], generating tools or extensions that enable the deployment of secure applications.

The validation of the developed technologies has been conducted in the general context of smart grids. This application domain, as many others, is affected by the increasing amount of data generated by a large range of device types (e.g., meters and sensors in the distribution or transmission systems) and by the sensitivity of such data (e.g., revealing habits from individual consumers

*Correspondence: andrey@computacao.ufcg.edu.br; stefan.koepsell@tu-dresden.de
[1]Universidade Federal de Campina Grande, Campina Grande, Brazil
[2]Technische Universität Dresden, Dresden, Germany
Full list of author information is available at the end of the article

[1]https://www.openstack.org
[2]https://www.kubernetes.io

or opening vulnerabilities in the operation of the power system).

The project considers several applications in the smart grids domain to demonstrate the feasibility and appropriateness of the SecureCloud platform for big data processing. In this paper we focus on two of them, which are part of a *smart metering big data analytics* use case:

- **Data validation**: computes a billing report, applying not only the billing logic, but also considering the completeness of the dataset and verifying the integrity of individual measurements;
- **Fraud detection**: estimates the likelihood of an individual consumer to be involved in frauds.[3]

The rest of the paper is organized as follows. We introduce the use case scenario in the "Smart metering applications" section and discuss the SecureCloud approach in the following section, "The SecureCloud approach". The "SQL-to-KVS converter – cascaDB" section and "Confidential batch job processor – asperathos" section discuss the components used in the two applications. The "Related work" section discusses other academic and industry consortia that address similar issues. The "Conclusion" section concludes the paper with some final remarks.

Smart metering applications

The smart metering scenario comprises the communication and data storage structure of *Advanced Metering Infrastructure* (AMI) systems, from the smart meters up to the *Metering Data Collector* (MDC), including the metering database. Figure 1 shows the complete communication, processing, and storage structure needed to validate a smart metering management scenario. Initially, this application was deployed using only real smart meters to validate this process. The smart meters communicate with an aggregator module using a mesh network based on a customized Zigbee stack for the transport layer [2]. As application protocol, we used a modified version of the Brazilian protocol for electronic metering reading, named ABNT NBR 14522 [3]. The modifications were necessary to enable the protocol to run in an asynchronous communication system, as well as to add some features, such as data encryption and to command load connection and disconnection. With this modification in the smart meter, all communication steps in the data workflow are encrypted. As shown in the figure, it is also possible to use different key lengths or algorithms.

[3]Frauds in electrical metering systems are endemic in some regions of Brazil, being responsible for approximately 27 *TWh* of energy loss annually. In general, energy theft accounts for 6% of the total distributed energy in Brazil according to the Brazilian Association of Electrical Power Distributors (see http://www.abradee.com.br/setor-de-distribuicao/furto-e-fraude-de-energia/ – in Portuguese).

A smart meter simulator software was then added to enabled experimenting with increasing number of connections with the MDC, making it model real communication demands more closely. It also provided flexibility when testing for the whole AMI as well as the SecureCloud platform in terms of data volume. This software is able to simulate up to 65,000 smart meters communicating and sending data simultaneously. The simulated data is created based on energy consumption data acquired from real customers.

Secure data communication is required to gather the metering data since the smart meters are located outside of the SecureCloud platform's scope. Data communication between the smart meters and the MDC, from physical to transport layers, must go through two different networks. In the pathway, aggregators work as bridges connecting smart meters to the authentication application in the cloud.

The smart metering management structure is inherently distributed, having usually hundreds of smart meters connected to one aggregator and several aggregators connected to authentication applications. Thus, the structure should enable several parallel authentication microservices to communicate with the aggregators acting as external clients and services.

The confidentiality and integrity of metering data must be protected. *Confidentiality* is important to maintain customers' privacy. *Integrity* is needed to protect the power utility against fraud attempts. Therefore, data processing within microservices in the cloud should be secure. The SecureCloud platform leverages SCONE [4] as the runtime for services and applications. As will be detailed in the next section, SCONE guarantees that applications are executed in protected regions of a processor's memory, shielding data and encryption keys even from software with higher privileged levels. These protected memory regions are named *enclaves*.

Besides the basic link-encryption between the different components, e.g., smart meters to authenticator and authenticator to MDC, smart meter data is additionally protected by applying end-to-end encryption and authentication. An RSA key pair and the public key of the MDC are embedded into each smart meter, enabling mutual authentication. To ensure end-to-end security, all ABNT request and response messages are encrypted. Since the MDC application and the database are located within the boundaries of a secured cloud platform, the data is decrypted inside the enclaves in the authentication application and stored using the platform database service. Data is also signed using cryptographic keys provided by INMETRO, the (Brazilian) National Institute for Metrology, Quality and Technology. These signatures will be later used for auditing of the bills. Finally, the data is stored into a secure key-value store (KVS) and consumed by applications.

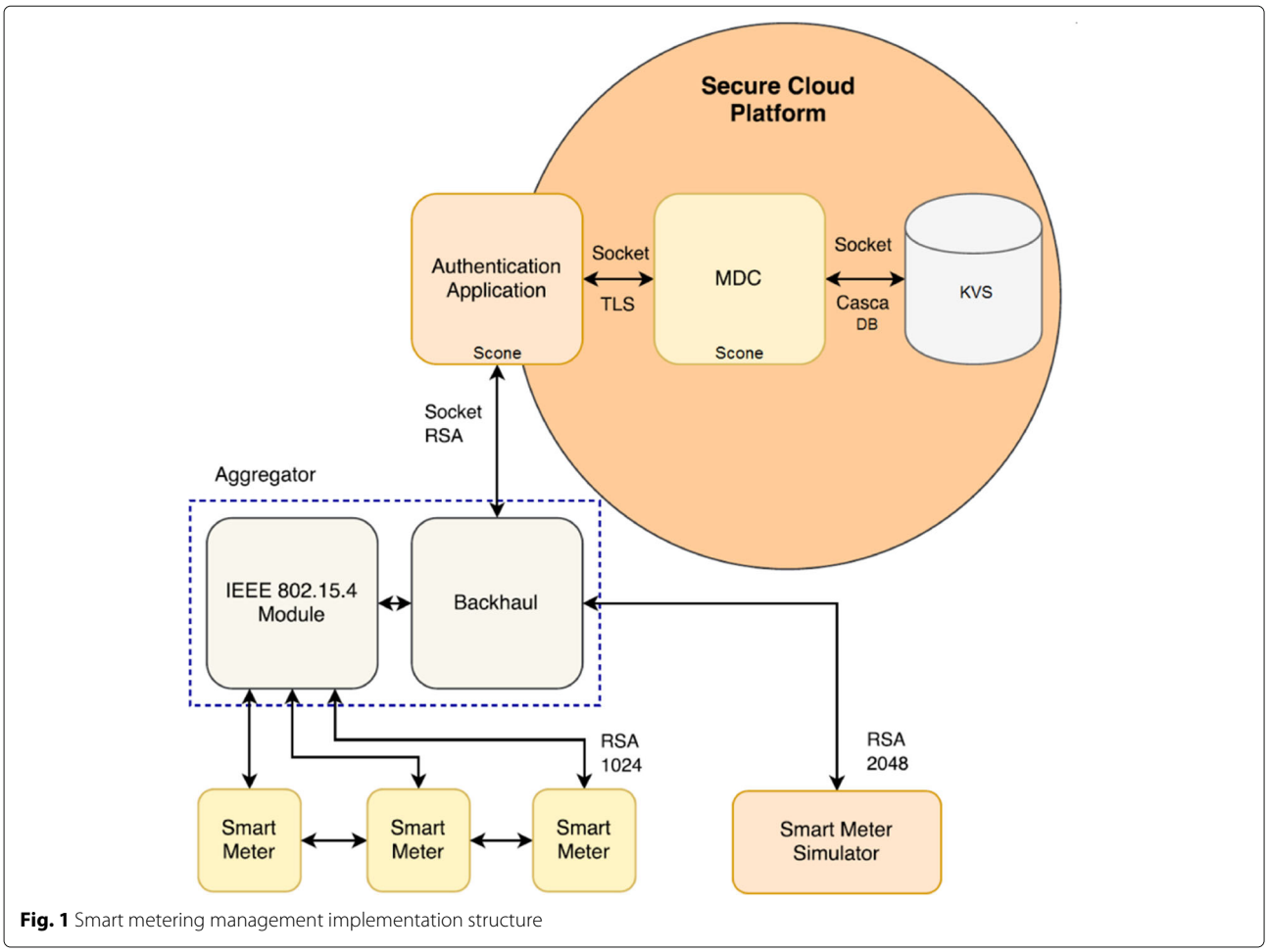

Fig. 1 Smart metering management implementation structure

Two applications that consume the data are considered here. The first application is for bill auditing and validation. It is composed of two main parts: one under control of the distribution company and does the billing, producing a bill and a hash value; while the other is controlled by INMETRO and, based on the hash, can check whether the bill considered all the measurements properly (e.g., no values were duplicated or missing).

The second application is for fraud detection. It processes all the measurements from consumers, considering also other demographic information (e.g., information specific to the individual consumer or of the type of industry) and generates a fraud risk evaluation. In particular, fraud attempts of large customers (i.e., with voltage supply greater than 1 kV) are typically much more sophisticated than frauds of residential customers. This type of customers have very high energy costs and, therefore, successful fraud attempts award large economical gains, fostering more sophisticated frauds.

The main idea of the fraud detection application is to use the stored measurements to detect variations in the customer's consumption behavior. Measurements, named *phasor reports*, are acquired every 15 minutes and contain data such as line voltage, current, angle, three-phase power, power factor, harmonic distortion, frequency, demand, and energy consumption. The basic concept behind the analysis includes the processing of the metering data by a classifier, based on self-organizing maps, in order to classify the behavior of consumption data. After the classification, the correlation coefficient matrix of the customer with the regular behavior is calculated, using a fixed time-window of one day. The results are used to create a ranking of potential fraudulent consumer units.

Both applications discussed above work as batch processes with the following tasks:

- Periodic, independent tasks to analyze individual consumers;
- A task for billing validation, which requires only the measurements from a single customer and the result is a validated billing report;
- A task for fraud detection, which may require additional information from other records or tables

and may issue additional queries to the database; the result is a fraud risk analysis report.

The SecureCloud approach

The SecureCloud approach is depicted in Fig. 2. Services are divided into infrastructure services, which control the cloud resources, and platform services, which implement higher level services used in the development of applications. Applications can be developed in a multitude of programming languages. Simple applications may use no platform services, while others could need support for secure indirect communication or scalable storage. Services and applications execute over a runtime, which facilitates the usage of hardware security features. These layers are described in the next sections.

SecureCloud runtime

Securing data by guaranteeing confidentiality and integrity is extremely desirable to protect sensitive smart grid data from malicious attacks. There are currently many cloud provider-specific approaches that provide some level of confidentiality and integrity. Nevertheless, they lack the capability to prevent application data from being compromised by software with higher privilege levels, such as hypervisors [5, 6].

Intel's *Software Guard eXtensions* (SGX) [1, 7] is becoming increasingly popular. It is a hardware-based technology that guarantees data integrity and confidentiality, creating a Trusted Execution Environment (TEE) that protects the code even in cases where the operating system and the hypervisor are not to be trusted.

SGX operates with a modified memory engine, generating protected areas named enclaves [8]. To provide integrity capabilities, SGX also offers local and remote attestation features [7], in which a third party can ensure that only the expected code is being executed inside an enclave, and on an authentic SGX-capable server.

Microservices in the SecureCloud platform can be developed either by directly using the Intel SDK [1] or using the SCONE runtime [4]. In our case, the recommendation is to use SCONE for the SecureCloud application development, i.e., for SecureCloud microservices that are closely related to a given SecureCloud application. Indeed, SCONE provides better usability and relieves the developers from much of the complexity of SGX programming. Intel SDK has been used for developing certain core microservices part of the SecureCloud platform itself, such as those related to storage and communication. In this case, the flexibility of the lower level programming enables optimizing performance and resource consumption at the cost of much greater development complexity.

SCONE provides the necessary runtime environment – based on the `musl` library[4] – to execute a microservice within an SGX enclave. SCONE is a set of tools that provide the necessary support for compiling and packaging Docker containers from the microservice's source code. The SecureCloud runtime supports several programming languages, notably C, C++, Go, Fortran, Lua, Python, R, and Rust.

Finally, because of its availability and security guarantees, we currently focus on Intel SGX. Nevertheless, the SCONE approach abstracts most security concerns and, thus, other trusted execution environments could be eventually supported. To illustrate the performance cost of using SCONE with Python, the most popular programming language currently, we have selected a variety of benchmarks from the Python community[5] and created an interactive portal with the PySpeed tool to make it easier to compare performance of different execution alternatives[6]. A subset of the benchmarks is shown in Fig. 3. Four bars are shown for each benchmark, illustrating the slowdown or speed up when using two versions of Python (the default CPython interpreter and, PyPy, an alternative, more efficient implementation[7]), with or without SCONE. The first bar in each benchmark depicts the execution with PyPy, but without the SCONE runtime; the second shows the execution with PyPy and SCONE; the third, which is the reference value, corresponds to the execution with the default Python and without SCONE; finally, the fourth bar depicts the execution with the default Python interpreter and SCONE. In most of the cases, the overhead of using SCONE is small and can be mitigated by using a more efficient implementation of Python, such as PyPy.

In the case of our two applications, they have been developed to use the SCONE toolset: the Data Validation is written in C and compiled with SCONE, while the Fraud Detection has been developed in Python and runs on top of a Python interpreter compiled with SCONE. Both tools use infrastructure and platform services developed in different languages as detailed next.

Infrastructure services

The lower-level services in the SecureCloud ecosystem are named *SecureCloud Infrastructure Services* and comprise services necessary to deploy and execute SecureCloud microservices. Examples of infrastructure services are scheduling and orchestration, attestation, auditing and monitoring.

Two platforms, OpenStack and Kubernetes, are the cornerstones of the SecureCloud infrastructure services and

[4] https://www.musl-libc.org/
[5] https://speed.pypy.org/about/
[6] http://pyspeed.lsd.ufcg.edu.br
[7] https://pypy.org/

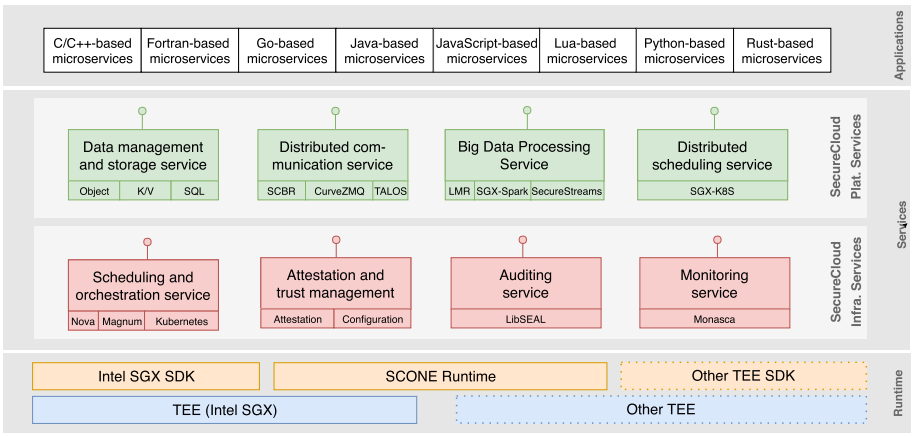

Fig. 2 Overview of the SecureCloud platform and its components

have been selected because of their popularity and the provided features. OpenStack is an open-source IaaS platform that can be used to deploy private, public, and hybrid clouds. Recent user surveys among OpenStack [9] and Kubernetes users [10] report that Kubernetes is the most popular tool for using containers in the cloud, used by 50% of deployments. The surveys still highlight that Kubernetes leads the usage for orchestration in production environments with 32%, followed by Docker Swarm (10%) and Apache Mesos (6%).

In Fig. 2, three services are relevant for the applications described in this paper. The *Monitoring Service* is based on the upstream OpenStack Monasca system.[8] In our case, we simply add a few metric collection agents that expose metrics related to the usage of secure resources, for example, SGX Enclave Page Cache (EPC) usage. This metric is useful because over-commitment of EPC memory causes considerable performance degradation. In addition, other relevant metrics for the monitoring of secure applications have been added and open sourced[9].

Once applications are running inside an enclave, the next issue is to provide them sensitive configuration, such as secrets. This is a known hassle of Intel SGX as it requires the development and operation of a different piece of software to remotely attest the original applications before passing the secrets. Both our demonstration applications require secrets, especially certificates for authenticating itself and the other microservices used. The *Configuration and Attestation Service* (CAS) is then responsible for the attestation and trust management, abstracting the problem of verifying if applications that are actually running have the code hashes registered during development and validation. These code hashes are computed upon application instantiation and serve as a signature for that application or microservice. If the signature of the running application match with a previously registered signature, the application is granted access to secrets, such as database credentials, TLS certificates and configuration parameters.

Lastly, the *Scheduling and Orchestration Service* provides functionalities spread over a set of low-level services. For example, we modify OpenStack Nova so that metadata on the flavor is passed to the special hypervisor that is able to instantiate VMs with SGX access. This special hypervisor is a fork from the popular KVM hypervisor and is maintained by Intel[10]. For cloud environments without KVM, an alternative is to consider bare-metal instances (e.g., using OpenStack Ironic[11] or infrastructure containers (e.g., LXD[12]), which could directly access the non-virtualized SGX device. In addition, the OpenStack Magnum component instantiates Kubernetes clusters on demand, clusters which now need to be aware of SGX.

Platform services

The higher level services are named *SecureCloud platform services*. These services offer functionality that can be used by applications. More specifically, the SecureCloud platform services offer different types of storage services, such a secure key-value and object store, and an SQL adapter on top of it. For communication, the SecureCloud platform services offer point-to-point communication as well as many-to-many communication. The latter, named *Secure Content Based Routing* (SCBR) [11], is based on the publish/subscribe model.

Additionally, data processing services are also offered. One example is the usage of MapReduce paradigm for secure data processing [12]. Another example is a set of modifications made on Apache Spark that enable

[8] http://monasca.io/
[9] https://github.com/christoffetzer/linux-sgx-driver
[10] https://github.com/intel/kvm-sgx
[11] https://wiki.openstack.org/wiki/Ironic
[12] https://linuxcontainers.org/lxd/getting-started-openstack/

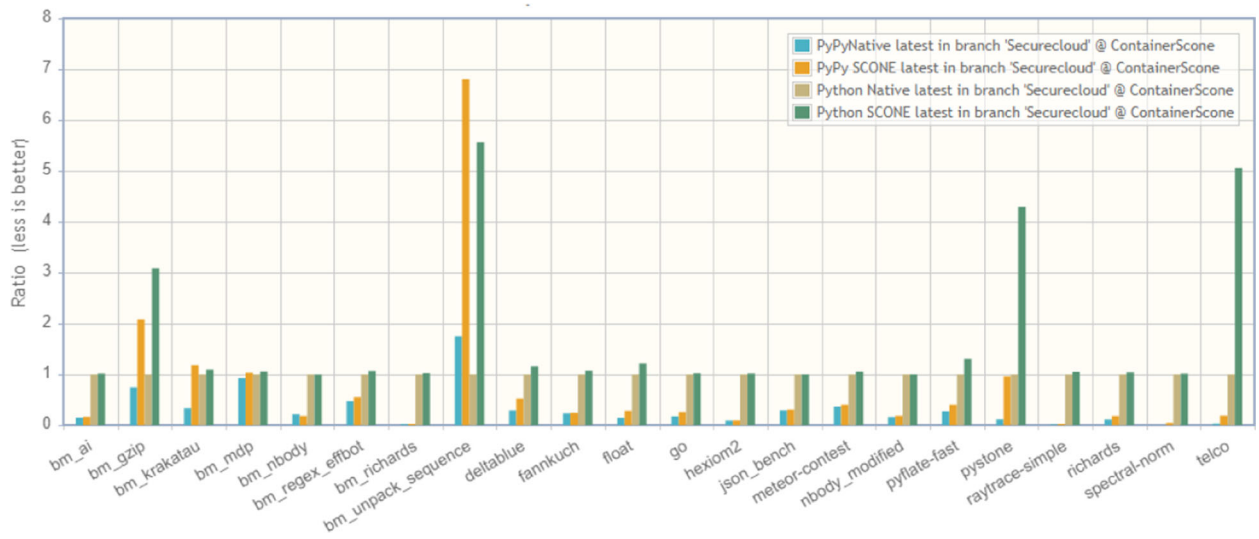

Fig. 3 Performance overhead of common Python benchmarks running with SCONE

encrypted data to be processed in a way that decryption is done only inside enclaves.

Three platform services are specially important for the data validation and the fraud detection applications presented in this paper: the secure key-value store (KVS), the SQL-to-KVS converter (CascaDB) and the batch job processor (Asperathos). These services are detailed next.

Secure KVS

We have implemented a distributed Secure Key-Value Store (KVS) that exhibits a trade-off between the features required by big data applications and the complexity, performance, and security impact of individual features. The rationale for our feature set is described in [13]. As a result, we managed to implement a reliable, scalable, and secure system that satisfies our use case requirements. For example, both Billing Validation and Fraud Detection applications have a phase for data aggregation where large blobs of measurement sets for individual consumers and retrieved and analysed, while being kept confidential in all steps from storage to processing, even on top of standard storage volumes provided by the cloud infrastructure.

The KVS provides scalability by being itself a set of microservices that run on top of Kubernetes. As seen in Fig. 4, client applications communicate with the KVS via HTTPS connections, using their unique API keys to authenticate themselves. In order to protect the confidentiality and authenticity of objects stored in the KVS, a microservice was added in front of our REST API. We refer to it as the TLS interceptor which is based on TaLoS [14], our library for building applications that support TLS connection termination inside enclaves. This interception service has the role of intermediating the connection between applications and storage components, handling the HTTPS requests inside the enclaves and encrypting the data that is written to the KVS. Likewise, it intercepts and decrypts HTTPS responses containing object data before they are returned to the client. The termination of TLS connections and encryption/decryption operations are performed inside an SGX enclave, thus cleartext object data is never exposed outside of an

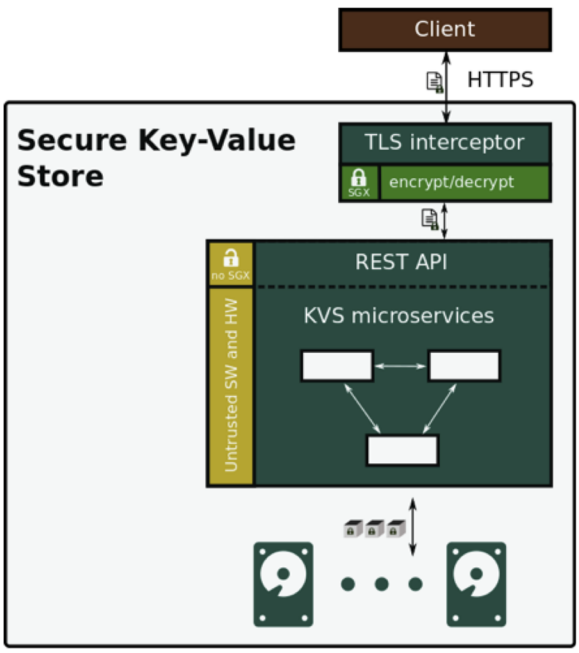

Fig. 4 Secure KVS:architecture overview

enclave. The other microservices that make up the KVS handle only encrypted object data and have no access to the decryption keys. After the TLS interceptor component, the request is forwarded to a well-defined REST API. The request is handled in the KVS microservices according to the target storage policy, i.e. redundancy is added before it gets stored on a variable number of distributed persistent storage nodes. Positioning the TLS interceptor as the gateway of the KVS removes the need to provide trusted computation capabilities in the other components of our system. Thus, the other KVS microservices do not need to run on SGX hardware.

This secure KVS provides reliability and redundancy by distributing data onto multiple storage nodes using replication, erasure coding, or a combination of the two. Any combination of replication and erasure coding can be specified to achieve the optimal trade-off between storage cost, throughput, and reliability. A frequently used object (hot data) can be stored with multiple replicas. An infrequently used object (warm data) could have one exact replica and eight erasure coded fragments, of which only six are necessary for recovering a copy of the object. An archival object (cold data) can be stored simply by using erasure coding. Additionally, the policy can be changed as object access patterns change.

As seen on Fig. 5, replication has much higher storage requirements than erasure coding. However, it has a higher throughput for storing or accessing data as it does not require encoding or decoding steps, respectively. Hybrid policies are in between the two approaches in storage and throughput, thus, they provide a natural trade-off: data can be accessed from the replica as often as possible, while storage costs are reduced.

Lastly, we have deployed the KVS to an OpenStack cluster, using the infrastructure services discussed earlier, and measured the read and write operations per second for various data sizes. The results can be seen Fig. 6. We used Kubernetes to orchestrate the microservices. The benchmarks used 32 distinct Python client processes to execute the read and write operations, ensuring that enough load is placed on the system. There were 8 instances of the TLS interceptor component running on two different nodes, each with its own 8-core SGX-enabled virtual CPU (vCPU) and 16 *GB* or RAM. Incoming requests were load-balanced between the two. These were the only nodes in the deployment to utilize SGX hardware. The microservices in charge of accessing the underlying storage subsystem ran on 6 nodes with a single vCPU core and 2 *GB* of RAM each. These storage nodes used Kubernetes' persistent volumes feature to ensure they always accessed the correct storage volume. The volumes were mapped to the underlying physical storage devices using OpenStack Cinder with a CEPH backend[13], both required no modification as data was already encrypted. All other KVS microservices were scheduled to run on a single node with an 8-core vCPU and 16 *GB* of RAM. Additional details on the experiments can also be found in [13].

SQL-to-KVS converter – cascaDB

Many applications need to handle high volumes of data, be it a simple information system that needs to persist customer data or a complex application that needs to persist low-level data for posterior aggregation. Using a database factors out much of the complexity to manage the data. The two applications discussed in this paper are no exception.

When choosing a database, one initial decision a developer needs to make is regarding the trade-off between scalability and feature set. On the one hand, key-value storage systems are easier to scale both in performance and capacity. On the other hand, relational systems offer a much richer set of features and count on the experience of developers with SQL semantics.

Not surprisingly, power grids, as other critical systems, have many legacy applications. Rewriting applications is a major obstacle that needs to be overcome when migrating the applications to the SecureCloud platform. In the SecureCloud project, we take a hybrid, layered approach. First, the core of existing applications does not need to be modified to run in enclaves. As detailed in previous sections, the SCONE toolset enables compiling applications written in languages such as C/C++ and Fortran to be run inside enclaves and be remotely attested.

Next, we understand that managing big data requires systems that scale massively. In addition, and maybe even more importantly, making these systems fault-tolerant and secure requires carefully thinking on which features will be implemented. We used the distributed secure KVS described in "Secure KVS" section and built a SQL mapping engine that runs inside enclaves, supporting the basic queries needed by our applications.

Thus, on top of the KVS, the *Customizable Adapter for Secure Cloud Applications* (CascaDB) converts SQL statements into key-value accesses transparently. It offers TLS connections for communication and runs inside SGX enclaves to enable protection of sensitive data (including the SQL statements themselves). Figure 7 depicts the approach adopted for CascaDB's implementation. On the top left-hand side, the querying service (e.g., MDC) accesses CascaDB. On the top right-hand side, the KVS implements the actual storage. Using such an approach, legacy systems used to insert measurements in the databases require no modifications,

[13] https://ceph.io

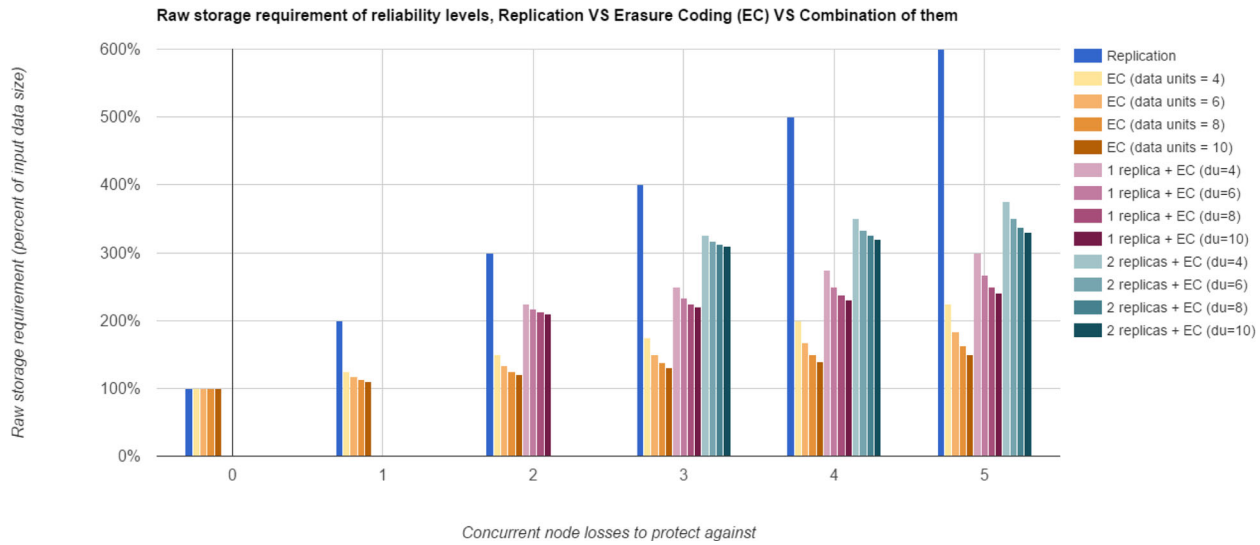

Fig. 5 Secure KVS: Comparing the raw storage requirements of different data protection policies, namely, replication, erasure coding and hybrid policies

while new analysis applications such as new implementations of the Data Validation and Fraud Detection applications can be written to access data directly in the Secure KVS to perform big data analysis more efficiently.

CascaDB receives SQL statements and returns JSON results after transforming these data requests into key-value ones. It was built in a modular architecture. Independent modules are easier to fit in the enclave's limited memory and can be replicated for scalability.

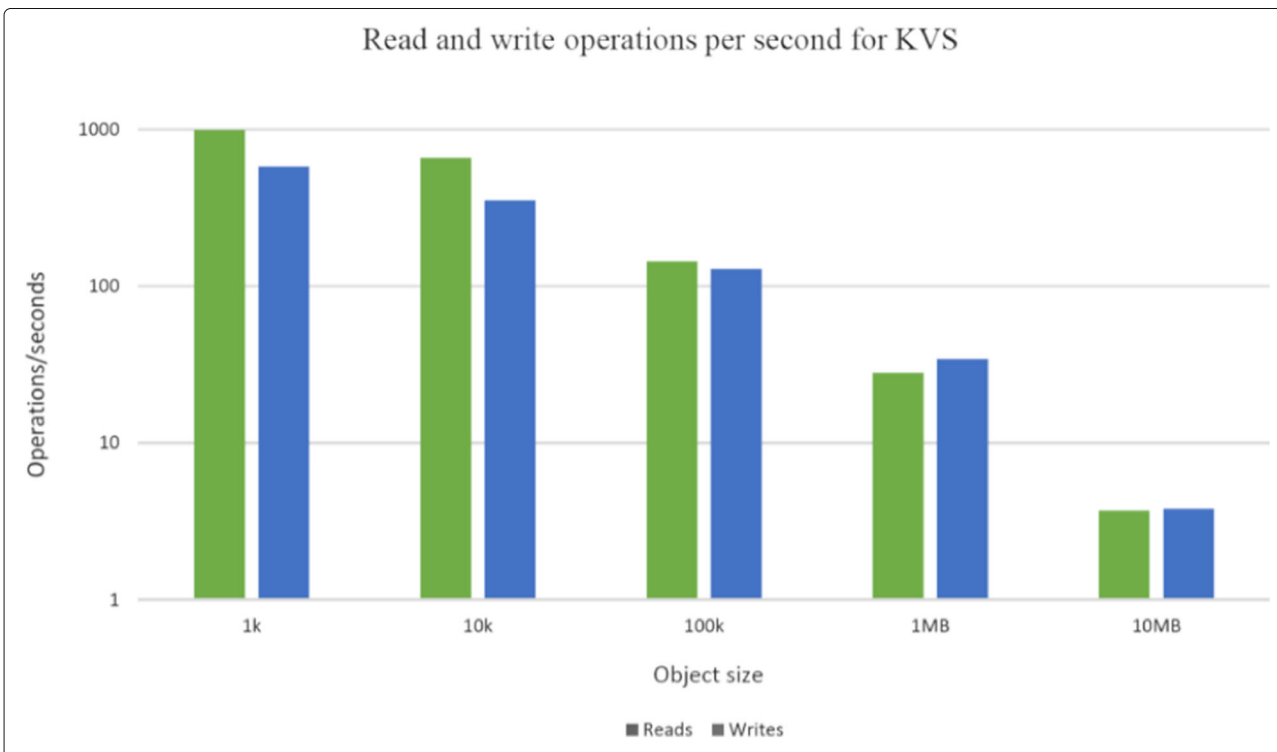

Fig. 6 Secure KVS: read and write operations for different object sizes

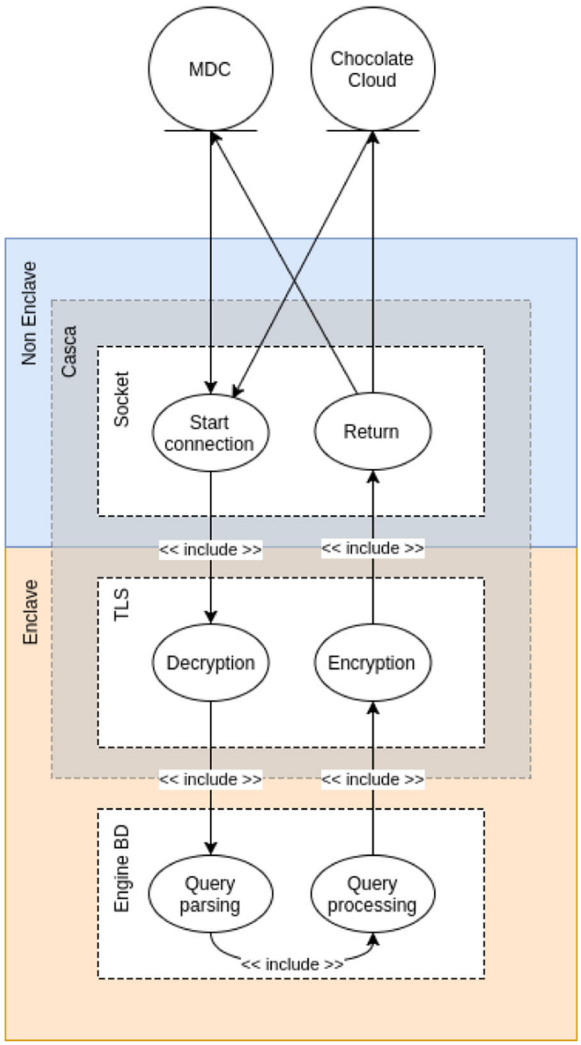

Fig. 7 CascaDB approach

The *SQL translation engine* handles SQL queries and interacts with the underlying KVS module to answer requests. Figure 8 depicts the internal parts of the SQL translation module and its interactions with the KVS. Data definition (DDL) queries are handled by the parser and have their settings stored in the data dictionary. The schema and keys define how each tuple will later be mapped to the key-value model.

When a data manipulation (DML) query is received, a dictionary lookup is performed to determine the mappings between the models. For insertion queries, a key is composed by a concatenation of the table name and the primary key(s) (a predefined separator is used for readability). Tuple attributes are also concatenated to be stored as the value. Figure 9a depicts an insert query for a table named `reading` that has the first two attributes as primary keys. Figure 9b depicts the resulting key-value pair generated. The key-value pair is then submitted to the KVS. For selection queries the key is assembled with the table name and the keys presented in the query, potentially with a wildcard character (*) to retrieve all keys matching the prefix. Currently only queries with conditions over the primary keys are supported. More flexible selects and joins are upcoming developments.

For example, a wider coverage of the SQL language can be implemented with extra indexes for non-key attributes – stored in the key-value store. Joins on the primary keys can be implemented efficiently since the KVS returns keys in alphabetical order, enabling the use of a merge-join algorithm. Other joins can be implemented using the aforementioned extra indexes or full table scans when indexes are not available.

Confidential batch job processor – asperathos

The Asperathos framework[14] provides tools to facilitate the deployment and control of applications running in cloud environments. For example, Asperathos can provide quality of service (QoS) by controlling resources allocated during runtime. Nevertheless, in contrast to other orchestration tools, such as Kubernetes itself or OpenStack Heat, it can be configured to consider application specific metrics and to actuate in a customized fashion. In the case of batch jobs using enclaves for data protection, as in the case in the two applications considered here, this is even more relevant as the operator may want to add the usage of enclave memory into the control logic [15].

The architecture of Asperathos is depicted in Fig. 10. It is composed of three main modules: (*i*) the *Manager* is the entry point for the user and is responsible for receiving an application submission, triggering all other steps; (*ii*) the *Monitor* is responsible for gathering, transforming and publishing metrics collected from applications (e.g., the application progress) or environment resources (e.g., CPU usage); (*iii*) the *Controller* is the component that adjusts the amount of allocated resources dedicated to an application.

Each of the components above can be customized with plugins. For example, in a deployment with the Secure KVS serving several applications and the Data Validation application executing analysis of the user billing data, the user could could want to have custom plugins in the Monitor and in the Controller so that whenever the Secure KVS scales up its TLS interceptors, the Data Validation batch workers are reduced to free up SGX resources.

While both applications require some batch processing in addition to the storage and communication services, diversifying the number of infrastructure services can increase the operation burden. Therefore, we

[14] https://github.com/ufcg-lsd/asperathos

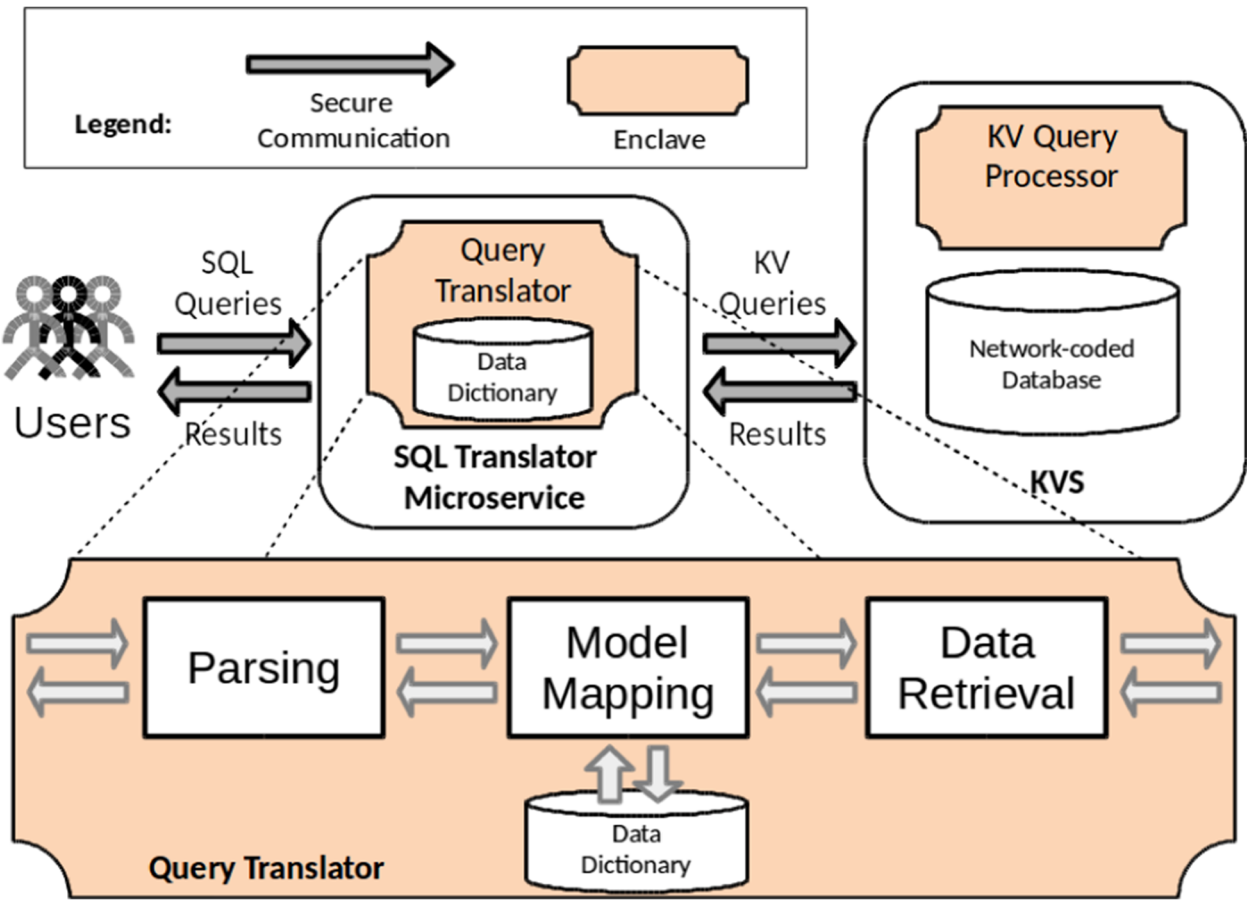

Fig. 8 Service architecture

leveraged Kubernetes also for the orchestration of legacy batch jobs.

As discussed in "Smart metering applications" section, both the Data Validation and Fraud Detection applications discussed in this paper work as a set of independent tasks. This pattern is easily mapped into Kubernetes with the help of the Kubernetes `Job` object[15]. The job abstraction provides the ability to run finite workloads by creating a set of containers (i.e., *Pods*, in Kubernetes terminology) and guaranteeing that a specified number of them successfully terminate. It is also possible to specify parallelism, time-out deadlines, and basic retrial policies, turning it into an option for running batch workloads while benefiting from other Kubernetes features and tools. We have then implemented three plugins to enable the execution of batch processing tasks using Kubernetes jobs, and the logic to monitor and actuate (e.g., scaling the cluster) through Kubernetes APIs.

To properly orchestrate secure containers on standard cloud clusters, Kubernetes needs to deal with the infrastructure's SGX capabilities. As discussed above, containers requiring SGX will contend on the availability of enclave memory. The monitoring infrastructure that feeds Kubernetes' scheduler with resource metrics must keep track of enclave memory requests and allocate the containers accordingly.

We then developed a vertical implementation of an SGX-aware architecture for orchestrating containers inside Kubernetes clusters. Using a slightly customized SGX driver, we have Linux report on enclave memory usage per container and to enforce limits to their allocation, providing Kubernetes with a new device plugin. We also implemented a new Kubernetes scheduler

```
a) INSERT INTO reading
       (   <clientID>, <timestamp>, <read 1> ... <read 30>);
b) key: reading—<clientId>—<timestamp>
   value: [<read 1> ... <read 30>]
```

Fig. 9 Example of an insert query (**a**) mapping into a key-value pair (**b**). clientID and timestamp compose the primary key of the table and become part of the mapped key. The table has 30 attributes that were omitted for clarity

[15] https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/

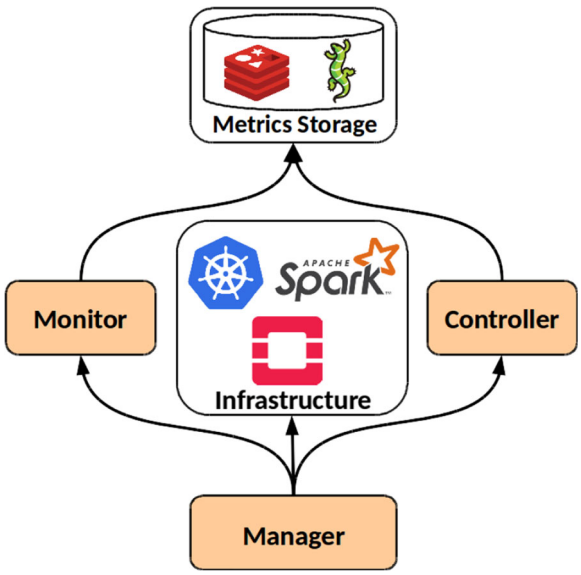

Fig. 10 Asperathos architecture

that takes into account the amount of enclave memory required by the containers and the amount of enclave memory available in the cluster nodes, preventing over-allocations, which may abruptly decrease performance in orders of magnitude [4]. Our implementation has been used to map container jobs with security requirements with priority on SGX machines [16] and to demonstrate that appropriate management of enclave memory management can reduce their overall turnaround time.

To illustrate the Asperathos framework in execution, Fig. 11 present a view of a Grafana dashboard.[16] Three different executions are shown, each with a different tolerance value regarding to how late an execution could be before actuation is triggered. As expected, less tolerance to deviations implies on more actuation events on the infrastructure.

Related work

The use of Trusted Execution Environments (TEE) in cloud computing scenarios has been growing recently as it can help address confidentiality and integrity challenges when delegating critical processing and sensitive data to the cloud. As an example of other initiatives, SERECA [17] has investigated the usage of SGX enclaves to harden microservices following a reactive paradigm based on the Vert.x toolkit[17]. Fetzer et al. then illustrate the usage of this framework with two use cases: a water management system, which has an integrity requirement, and a cloud-based service for performance analysis of applications, which has a confidentiality requirement.

Another research initiative to use TEE in cloud computing is the ATMOSPHERE project [18]. In this case, the consortium addresses the more general problem of trustworthiness in the health domain. In their case, in addition to confidentiality and integrity, the TEEs are used to support other trustworthiness features such as privacy (by executing anonymization algorithms in enclaves) and transparency (by using enclaves to enforce that accesses to sensitive data are logged and can be tracked).

Finally, on the industry side, the recent Confidential Computing Consortium, announced by the Linux Foundation[18], is a cross industry initiative aiming to speed up the adoption of confidential computing mechanisms. The initiative includes producers of alternatives to the SCONE toolkit used in this paper to port applications to run on SGX enclaves: Microsoft's Open Enclave SDK[19] and Google's Asylo[20].

Conclusion

This paper presents the SecureCloud approach for implementing secure data processing for big data applications. The SecureCloud approach is divided into runtime, infrastructure, and platform services. The infrastructure services provide the mechanisms for orchestrating basic resources such as VMs, containers, clusters, and secrets (including both the attestation of applications and the provisioning of the secrets). On top of the infrastructure services, platform services have been provided to support the application developer. Examples of platform services are the secure content-based publish-subscribe service (SCBR) and the secure key-value store (Secure KVS). For the applications, end users with limited security knowledge are advised to build applications on top of the SCONE runtime, having the option of using compilers (e.g., for C or Fortran) or interpreters (e.g., Python). Such an approach relieves the developer from handling non-trivial tasks, such as having to split code between secure and insecure portions, and to build ad hoc helpers tools for attestation and secret provisioning. Leveraging a richer runtime also enables the user to inherit benefits from approaches that enhance security, such as side-channel protection mechanisms [19].

For the development of the infrastructure and platform services themselves, we used a mixed approach. Some services were built on top of Intel SGX SDK to enable more flexibility in splitting the code into secure and insecure parts, leading to less resource usage at the cost of much higher complexity. Some other services reuse existing code and therefore are simply adapted and recompiled using the SCONE toolset. Details on individual

[16] https://grafana.com/
[17] http://vertx.io/
[18] https://confidentialcomputing.io
[19] https://openenclave.io
[20] https://asylo.dev/

Fig. 11 Selected executions in the Grafana dashboard with tolerances set to 0%, 5%, and 10%

components, including performance evaluations, can be found on the project's website[21].

Finally, two applications in the context of Smart Grids were used to validate the proposed tools: a data validation and a fraud detection application. These applications have been used to illustrate the need for some of the infrastructure and platform services, for example: the need for bootstrapping application requires trust management and distribution of secrets, achieved through the Configuration and Attestation Service (CAS); the need for support for both legacy SQL queries and blob accesses for big data processing jobs, achieved through the CascaDB and Secure KVS; and the need for orchestrating applications using popular tools, but still enabling the efficient usage of the currently very limited enclave memory, achieved using OpenStack, Kubernetes and Asperathos.

Acknowledgements
The SecureCloud Project was funded by the Brazilian Ministry of Science Technology and Communications, the European Commission and the Swiss State Secretariat for Education, Research and Innovation through the Horizon 2020 Program, in the 3*rd* Brazil-Europe coordinated call. We also thank the UCC Cloud Challenge 2018 organizers and the anonymous reviewers for their feedback. Open Access Funding by the Publication Fund of the TU Dresden.

Authors' contributions
As the paper describes an end-to-end approach for a complex problem, the author's contributions encompass several different areas, as mentioned below. AB is the lead architect on the OpenStack integration and of the implementation Asperathos framework. CF is the lead architect for the SCONE toolset. SK was responsible for the overall architecture definition and integration. PP is the lead architect in the TaLos toolset. MP is the lead architect on the integration of the SGX scheduler into Kubernetes. PF is the lead architect for the SCBR message bus. KF was responsible for the integration of the applications. MR was the main responsible for the evaluation of the applications. LG-Jr is the lead architect of CascaDB. RR represented the power grid client companies and contributed with the load generator and application requirements. CP was responsible for the development of the data validation application. LFR was the lead architect for the data validation application. DEL was the lead architect for the Secure KVS service. MS was responsible for the development of the secure KVS service. LN was responsible for the operation and evaluation of the secure KVS service. MF was responsible for the initial implementation of the secure KVS service. All authors read and approved the final manuscript.

Availability of data and materials
The data used to feed the experiments in the project contain personal information and cannot be disclosed. Nevertheless, the actual data used in the processing does not affect the results of the experiments presented. More information about tools used in the paper can be found in the project webpage: www.securecloudproject.eu.

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

Author details
[1]Universidade Federal de Campina Grande, Campina Grande, Brazil. [2]Technische Universität Dresden, Dresden, Germany. [3]Imperial College London, London, United Kingdom. [4]University of Neuchâtel, Neuchâtel, Switzerland. [5]Universidade Tecnológica Federal do Paraná, Curitiba, Brazil. [6]LACTEC, Curitiba, Brazil. [7]INMETRO, Rio de Janeiro, Brazil. [8]Chocolate Cloud ApS, Aalborg, Denmark.

Received: 6 May 2019 Accepted: 15 October 2019
Published online: 04 December 2019

[21] https://www.securecloudproject.eu

References

1. Intel (2015) Intel Software Guard Extensions. Cryptology ePrint Archive, Report 2016/086. https://software.intel.com/sites/default/
2. ZigBee Standards Organization (2012) ZigBee Specification 053474r20. Rev. 20
3. Associação Brasileira de Normas Técnicas (ABNT) (2008) Intercâmbio de informações para sistemas de medição de energia elétrica. Rev. 1
4. Arnautov S, Trach B, Gregor F, Knauth T, Martin A, Priebe C, Lind J, Muthukumaran D, O'Keeffe D, Stillwell ML, Goltzsche D, Eyers D, Kapitza R, Pietzuch P, Fetzer C (2016) Scone: Secure linux containers with intel sgx. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16. USENIX Association, Berkeley. pp 689–703. http://dl.acm.org/citation.cfm?id=3026877.3026930
5. Szefer J, Keller E, Lee RB, Rexford J (2011) Eliminating the hypervisor attack surface for a more secure cloud. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS '11. pp 401–412. https://doi.org/10.1145/2046707.2046754
6. Subashini S, Kavitha V (2011) Review: A survey on security issues in service delivery models of cloud computing. J Netw Comput Appl 34(1):1–11
7. Costan V, Devadas S (2016) Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086. http://eprint.iacr.org/2016/086
8. McKeen F, Alexandrovich I, Berenzon A, Rozas CV, Shafi H, Shanbhogue V, Savagaonkar UR (2013) Innovative instructions and software model for isolated execution. In: Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP '13. ACM, New York. pp 10–1101. https://doi.org/10.1145/2487726.2488368, http://doi.acm.org/10.1145/2487726.2488368
9. OpenStack Foundation OpenStack User Survey November 2017. https://www.openstack.org/assets/survey/OpenStack-User-Survey-Nov17.pdf. Access 1 Apr 2019
10. Cloud Native Computing Foundation (2017) Cloud Native Technologies Are Scaling Production Applications. https://www.cncf.io/blog/2017/12/06/cloud-native-technologies-scaling-production-applications/. Access 1 Apr 2019
11. Pires R, Pasin M, Felber P, Fetzer C (2016) Secure content-based routing using intel software guard extensions. In: Proceedings of the 17th International Middleware Conference, Middleware '16. ACM, New York. pp 10–11010. https://doi.org/10.1145/2988336.2988346. http://doi.acm.org/10.1145/2988336.2988346
12. Pires R, Gavril D, Felber P, Onica E, Pasin M (2017) A lightweight mapreduce framework for secure processing with sgx. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid '17. IEEE Press, Piscataway. pp 1100–1107. https://doi.org/10.1109/CCGRID.2017.129. https://doi.org/10.1109/CCGRID.2017.129
13. Lucani DE, Feher M, Fonseca K, Rosa M, Despotov B (2018) Secure and scalable key value storage for managing big data in smart cities using intel sgx. In: 2018 IEEE International Conference on Smart Cloud (SmartCloud). pp 70–76. https://doi.org/10.1109/SmartCloud.2018.00020
14. Aublin P-L, Kelbert F, O'Keeffe D, Muthukumaran D, Priebe C, Lind J, Krahn R, Fetzer C, Eyers D, Pietzuch P (2018) LibSEAL: Revealing Service Integrity Violations Using Trusted Execution. In: Proceedings of the Thirteenth EuroSys Conference. ACM, New York. pp 24:1–24:15. http://doi.acm.org/10.1145/3190508.3190547. http://doi.org/10.1145/3190508.3190547
15. Ataide I, Vinha G, Souza C, Brito A (2018) Implementing quality of service and confidentiality for batch processing applications. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). pp 258–265. https://doi.org/10.1109/UCC-Companion.2018.00065
16. Vaucher S, Pires R, Felber P, Pasin M, Schiavoni V, Fetzer C (2018) Sgx-aware container orchestration for heterogeneous clusters. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). pp 730–741. https://doi.org/10.1109/ICDCS.2018.00076
17. Fetzer C, Mazzeo G, Oliver J, Romano L, Verburg M (2017) Integrating reactive cloud applications in sereca. In: Proceedings of the 12th International Conference on Availability, Reliability and Security, ARES '17. ACM, New York. pp 39–1398. https://doi.org/10.1145/3098954.3105820. http://doi.acm.org/10.1145/3098954.3105820
18. Brasileiro F, Brito A, Blanquer I (2018) Atmosphere: Adaptive, trustworthy, manageable, orchestrated, secure, privacy-assuring, hybrid ecosystem for resilient cloud computing. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). pp 51–52. https://doi.org/10.1109/DSN-W.2018.00025
19. Oleksenko O, Trach B, Krahn R, Silberstein M, Fetzer C (2018) Varys: Protecting SGX enclaves from practical side-channel attacks. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18). USENIX Association, Boston. pp 227–240. https://www.usenix.org/conference/atc18/presentation/oleksenko

Publisher's Note