Sealed computation: a mechanism to support privacy-aware trustworthy cloud service

Lamya Abdullah and Juan Quintero

Department of Computer Science, University of Erlangen-Nuremberg, Erlangen, Germany and Department of Research and Development, Uniscon GmbH, München, Germany

Abstract

Purpose – The purpose of this study is to propose an approach to avoid having to trust a single entity in cloud-based applications. In cloud computing, data processing is delegated to a remote party for efficiency and flexibility reasons. A practical user requirement usually is data privacy; hence, the confidentiality and integrity of data processing needs to be protected. In the common scenarios of cloud computing today, this can only be achieved by assuming that the remote party does not in any form act maliciously.

Design/methodology/approach – An approach that avoids having to trust a single entity is proposed. This approach is based on two concepts: the technical abstraction of sealed computation, i.e. a technical mechanism to confine a privacy-aware processing of data within a tamper-proof hardware container, and the role of an auditing party that itself cannot add functionality to the system but is able to check whether the system (including the mechanism for sealed computation) works as expected.

Findings – Discussion and analysis of the abstract, technical and procedural requirements of these concepts and how they can be applied in practice are explained.

Originality/value – A preliminary version of this paper was published in the proceedings of the second International Workshop on SECurity and Privacy Requirements Engineering (SECPRE, 2018).

Keywords Data privacy, Auditor, Cloud service, Privacy by design, Sealed computation, Trustworthy cloud requirements

Paper type Research paper

1. Introduction

Cloud computing has become widespread because it allows for supplying and utilizing computation resources in an on-demand fashion. This reduces cost, increases flexibility and improves infrastructure scalability (Mell and Grance, 2009). Cloud computing is increasingly being adapted for several kinds of services, including services provided by networks of small devices, commonly referred to as the Internet of Things (IoT). IoT Cloud (Alam et al., 2010) or “Cloud of Things” (CoT) (Aazam et al., 2014) provides

The authors would like to thank Felix Freiling for his comments and suggestions on previous versions of this article and Zinaida Benenson for her collaboration on the previous version of this work published at SECPRE 2018 (Abdullah et al. (2018)). The authors would also like to thank Nico Dötting, Johannes Götzfried, Tilo Müller and Hubert Jäger for useful comments and discussions.

This research was supported by the “Privacy&Us” Innovative Training Network (EU H2020 MSCA ITN, grant agreement No.675730).
resources, such as storage, analytics tools and shared configurable computing resources, to reduce the cost and complexity associated with the IoT systems.

When data processing and storage are delegated to a cloud provider (CP), one of the primary concerns is users’ data privacy. Recently, the European General Data Protection Regulation (EU GDPR, 2016) was implemented. It emphasizes the importance of taking Privacy by Design into account (Art. 25 GDPR). Privacy by design is based on the idea that data processing environment is designed such that user’s privacy is considered at the very beginning of the development stage of the system (Cavoukian, 2011). Different privacy enhancing technologies (PETs) have been integrated into application designs. However, the cloud computing model is complex; consequently, various PETs are needed to be applied to the design of different layers so that the overall system is trusted for end users’ data privacy. Although protecting data at rest can be achieved by encryption and access control mechanisms, protecting data while in use, i.e. during processing, requires guarantees that the execution infrastructure is designed to be privacy-aware and does not allow access or modification of the data and code inside the execution environment.

The users of a cloud service usually have to trust the cloud provider to act as expected. However, in common cloud deployments, there is no technical guarantee that a single malicious insider, such as a system administrator, operator or a person with physical access to the infrastructure, does not tamper with code and data. Cloud clients, in the current practice, establish a sense of trust in the cloud provider via contracts over service level agreements (SLAs), auditing certificates and reputation. SLAs are used to define the expected service level, the objectives to be fulfilled, in addition to other aspects such as legal and financial issues. Unfortunately, even with the most refined SLAs, the necessity to place trust in the cloud provider remains. Hence, cloud clients should provide some technical guarantees and indications that the cloud service is trustworthy for privacy-aware data processing.

1.1 Example scenario
For example, consider the scenario of an IoT Cloud implementation for usage-based insurance (UBI) (Karapiperis et al., 2015), a car insurance business model, where the insurance company calculates premiums based on driver’s behavior using actual driving data. In UBI, participating cars are equipped with a telematics device to collect driving data such as location, speed, acceleration, cornering and other details. Driving data are processed to get a ranking based on personal driving behavior. Using the driver ranking, the insurance company calculates a customized premium to the driver using a more accurate risk estimate, thus reducing incurred losses (Derikx et al., 2016; Soleymanian et al., 2017) and offering a bonus in the case of good driving behavior. UBI promises many benefits such as reducing incurred losses through accurate risk estimates for the insurance companies (Soleymanian et al., 2017; Derikx et al., 2016) and improving the driving style through feedback and decreasing the premiums for the drivers. But, obviously, UBI also raises concerns such as user discrimination (Karapiperis et al., 2015) and privacy (Soleymanian et al., 2017; Derikx et al., 2016).

Privacy requirements of such an application are more than data confidentiality. For example, anonymity and data minimization require that users are not identified via the set of records and only the minimum amount of personal data is exposed to third parties. Various PETs can be applied to the UBI scenario to achieve such requirements. However, in this work, we are focusing on protecting data during processing because there is a need for a
definition of a privacy-aware infrastructure that protects data at use and not only at rest or in transit.

Figure 1 depicts an abstract view of UBI. The service provider may actually be the same entity as the insurance company, but in many business implementations [Bonus Drive by Allianz Deutschland AG (2017), Smart Driver by HUK-Coburg (2017)], it is a different company. One reason for separation is that insurance companies do not have the corresponding know-how to compute the driving ranking. Another reason is that the insurance companies want to mitigate privacy concerns by stating that they have no access to the behavioral data because it is processed by a third party (Allianz press release, 2017).

To provide a privacy-aware infrastructure for a sensitive data processing in the cloud, the below requirements must be fulfilled:

- **Confidentiality of data**: Drivers agree that their ranking is computed, but they want their individual usage data to remain confidential toward the insurance company and the cloud provider. While technically this could be achieved by processing data locally (i.e. in the car), local computation has disadvantages to the service provider who has the following requirement.

- **Confidentiality of code**: Service providers want to protect their intellectual property from other parties, particularly the insurance company and the cloud provider. So, while the driver ranking may be public, the software that is deployed in the cloud should be protected.

- **Integrity of data and code**: Insurance company, service provider and the drivers should have a guarantee that the cloud provider does not change data or code in any unauthorized way.

Notes: The data is processed by the service provider on behalf of the insurance company. Processing is performed by a cloud provider running the service provider’s software. The drivers receive feedback their driving habits.
A preliminary version of this work was published in the proceedings of the second International Workshop on Security and Privacy Requirements Engineering (SECPRE, 2018) Abdullah et al. (2018). In this study, a general approach that ensures generic confidentiality and integrity of cloud services and that avoids the necessity of having to trust a single entity is proposed. Our approach is based on the combination of two concepts:

1. sealed computation, i.e. an abstract mechanism to confine the processing of data within a tamper-proof hardware container; and
2. a procedural mechanism of mutual checking by applying the additional role of an auditing party, which is necessary to check whether the system works as expected but cannot modify the system.

We describe the abstract technical and procedural requirements of both concepts and argue that they are sufficient to achieve the abovementioned generic security properties. In the spirit of work by Morris (1973), our work is conceptual, avoiding over-formalization but still providing clear definitions and evaluating statements. The main insight is to show how an abstract hardware mechanism (sealed computation, which is solely defined by its requirements) must be utilized in the design of cloud service as a PET such that the necessity to trust in a single entity is avoided. Similar to other work (Schuster et al., 2015; Baumann et al., 2014), this study focuses on integrity and confidentiality properties and does not consider availability.

1.3 Outlook
In Section 2, related studies are discussed. In Section 3, the concept of sealed computation is defined. Then, in Section 4, the system and attacker model is presented. In Section 5, the procedural mechanism applying the role of an auditor is described. In Section 6, in addition to the analysis, it is argued that general security and privacy requirements are satisfied unless two parties act maliciously. Finally, in Section 7 the conclusion of this study is provided.

2. Related work
Privacy is a major factor in trusting data and computation outsourcing such as in a cloud-based applications (Pearson, 2013; Takabi et al., 2010). Hence, trust establishment has been discussed in the context of cloud from different perspectives in the literature. We distinguish them into technical and non-technical approaches. Georgiopoulou and Lambrinoudakis (2016) reviewed a number of trust models for cloud computing, trying to provide a gap analysis in the literature. However, the review considered only a very limited set of models. Pavlidis et al. (2014) studied the trust from requirements engineering perspective and suggested a methodology that enables developers to represent and reason about trust relationships in the system. Moreover, a framework that, in addition to the security and privacy requirements criteria, enables the analysis of trust assumptions by identifying direct and indirect trust relationships was proposed by Pavlidis et al. (2013). Supported by a CASE tool, the framework allows calculating a score for each cloud provider based on the analyzed trust relationship, which is defined as “a relationship that exists between the trustor and the trustee and resolves a dependency based on trust.” Then, the cloud provider with the highest score is selected.

Non-technical approaches, ranging from SLAs and recommendations for security architecture, risk management and operational teams, have been developed and used. For
example, Alhanahnah et al. (2017) studied a trust valuation framework to allow cloud clients to choose among a set of cloud providers based on trust levels. The authors distinguished trust factors into two sets: SLA-based and non-SLA factors based on the provider’s reputation and even financial status.

Rizvi et al. (2014) utilized the auditor role to provide an objective trust baseline assessment to enable clients to decide between CP candidates. The proposed model delegates the trust assessment to an auditor to calculate trust values. Therefore, clients who need to choose between CPs request the trust values from the auditor based on required service. The auditor role in our model is different from the one described by Rizvi et al. (2014) in which the auditor is actively involved in the deployment process of the cloud service and allows to establish stronger security guarantees (see Section 5).

Several technical means were studied and presented to enforce/enhance trust in the cloud. On the one hand, solutions that use cryptographic mechanisms, such as attribute-based encryption and policy enforcement and access control, are used to protect data at rest and to ensure that data are accessed and retrieved only for authorized users (Lacroix and Boucelma, 2014). Furthermore, Santos et al. (2012) employed attribute-based encryption to provide a policy enforcement protocol based on trusted platform modules. Similarly, Li et al. (2010) proposed a model to support security duty separation in multi-tenant Infrastructure-as-a-Service (IaaS) cloud between CP and customers based on trusted platform modules, and they optionally added the auditor role. Moreover, Ge and Ohousson (2010) proposed to build an architecture for the IaaS model that provides sealed storage and relies on remote attestation such that the clients trust to deploy their virtual machines. These models were designed for Platform as a Service (PaaS) and IaaS cloud models that require less security responsibilities on the CP; hence, responsibilities are shared with the customers. However, more responsibilities are required from CP in the SaaS model (Rizvi et al., 2014).

On the other hand, numerous approaches (Maene et al., 2017) have been proposed to achieve security requirements relying on trusted hardware. For example, hardware security modules (HSMs) (Dyer et al., 2001; Utimaco IS GmbH, 2018), i.e. tamper-resistant physical computing devices, can perform secure and confidential computation of data. Using HSMs, it is possible to deploy specific software modules, create cryptographic keys and process data purely within the hardware device. Returning to our UBI scenario, the HSM can be used to effectively protect the service provider’s data and code from the cloud provider. However, in this case, the necessity to trust a single entity is not avoided; it is merely shifted from the cloud provider to the trusted hardware provider. This observation is not specific to HSMs but holds also for other such technologies such as Intel SGX (Baumann et al., 2014; Schuster et al., 2015).

A trustworthy and privacy-aware cloud may be addressed by the use of cryptographic techniques such as fully homomorphic encryption (Gentry, 2009). However, it is still inefficient for most computations (Schuster et al., 2015). Similarly, verifiable computing (Parno et al., 2013) was designed to enable result correctness verification but has not shown support for general purpose cloud computing yet. Until these solutions are available for high level data processing there is a need for models that allow application providers and users to trust the software running on cloud service infrastructure for the privacy of data processing.

3. Sealed computation
While data at rest can typically be protected by encryption, data are commonly unprotected during processing against attackers that have physical access to the execution environment. We now introduce the abstraction of sealed computation that describes a well-defined level of protection against such attackers. Intuitively, this is done by encapsulating the software
execution within a physical piece of hardware. We introduce the definition of sealed computation using abstract roles to keep it general. Later, these roles are mapped to the parties of cloud computing (see Section 4) where we utilize the notion of sealed computation to maintain the integrity and confidentiality requirements of the system.

3.1 Definition

In sealed computation, a party $A$ provides a physical execution container $C$ into which a party $B$ may “seal” its software. The container $C$ ensures that the software is running in an unmodified fashion. Furthermore, $C$ also guarantees that only a restricted set of interactions with the software are possible through a well-defined interface. Moreover, no information is leaked from within $C$ to the outside, not even to $A$, the provider of the container, nor the software provider $B$.

More formally, let a party $A$ provide a physical execution container $C$ and party $B$ provide a software $M$ that implements some input/output specification via a well-defined interface. The interface can be thought of as a description of input/output signals over wires or the format of incoming or outgoing protocol messages.

To map the following sealed computation definition to the UBI scenario of Figure 1, it may help to think of the execution container $C$ being a specific HSM provided by party $A$ (the Cloud Provider), while party $B$ is the service provider (SP) who wrote software $M$ on behalf of the insurance company.

Definition 1 (Sealed Computation).

We say that $B$ seals $M$ within $C$ provided by $A$ if the following technical requirements are met:

- **Sealing**: $A$ and $B$ cannot access the code and data of $M$ after it has been sealed within $C$, apart from changes allowed by the interface.

- **Attestation**: As long as $M$ has not terminated and as long as $A$ acts honestly, $C$ can provide evidence, which proves that $C$ is running software provided by $B$ in a manner that is unique to the sealing instance, i.e. any change of $M$, $C$ or any subsequent sealing using the same combination will result in different evidence.

- **Black-box**: Information flow between $M$ and any other party (including $A$ and $B$) is restricted by the interface specification of $M$, i.e. nothing about the internal state of $M$ (code and data) can be learned apart from what is given away via the interface.

- **Tamper-resistance**: Any usage of $M$ that does not satisfy the interface specification results in the termination of $M$ and the destruction of $C$ such that neither code nor data from within $C$ can be retrieved.

Intuitively, the sealing requirement of sealed computation binds the execution of a program to a particular hardware environment. The requirements of black-box and tamper-resistance limit access to data and code only to interactions given in the functional specification of $M$. Black-box restricts information flow for expected interactions, while tamper-resistance does this for unexpected interactions.

The attestation requirement enables external parties to validate the fact that $M$ has been sealed. It implies that $C$ contains some known unique characteristic that can be validated by checking the provided evidence. This validation, however, depends on the correctness of $A$. A common realization of this is for $A$ to embed a secret key within $C$ and allow external parties to validate its existence by providing the corresponding public key. The existence of such a unique characteristic indicates that it is possible to establish an authentic and confidential communication channel to $M$ once sealing has started.
Similarly, note that \( B \) or any user of \( M \) still has to rely on \( A \) to act honestly because it is not verifiable whether \( C \) actually implements sealed computation. However, if \( B \) correctly seals \( M \) within \( C \) provided by an honest \( A \), even \( A \) cannot change \( M \) afterwards and the tamper-resistance requirement of \( C \) protects all secrets within \( M \) that are not accessible via its interface or before sealing.

3.2 Confidential software deployment
The notion of sealed computation is a powerful abstraction that can be used to describe techniques that protect software during deployment. We now argue that the technical requirements of sealed computation allow to ensure the confidentiality of the code that is sealed.

Intuitively, the idea of confidential software deployment is for \( B \) to initially install within the sealed computation a loader stub that is able to load the final user program specified by \( B \) into \( C \). Within the sealed computation, this software is decrypted, installed and then takes over the final interface operations expected by the users. This loader stub can be part of the sealed computation mechanism from the start. Because it can be easily added to any mechanism that satisfies Definition 1, we did not include it as an additional requirement in that definition.

We may observe that \( M \) cannot be assumed to remain confidential if \( A \) is untrustworthy. However, if \( A \) is trustworthy, sealed computation can be used to run code that remains confidential even toward \( A \).

3.3 Sealed computation implementation examples
The execution container \( C \) can be thought of as being a standalone physical machine protected and equipped with special active tamper-resistance mechanisms. Several technologies exist in practice that focus on protecting data and code within the execution environment. The following list (sorted chronologically) arguably shows that there exist real-world implementations that at least were intended to achieve the properties of sealed computation.

3.3.1 Hardware security modules. Certainly, HSMs, most closely, resemble what Definition 1 expresses. HSM is a physical computing device to manage and safeguard keys for strong authentication and perform protected computation. Based on the design of the HSM, it ensures tamper resistance by hardware-based sensor mechanisms that delete internal data upon detection of unusual environmental conditions. While it is not easily possible to run general-purpose applications within HSMs (Baumann et al., 2014), it is possible to install specific software modules on HSMs and create cryptographic keys with the hardware device that never leaves it, thereby supporting attestation. Common fabrication examples of HSMs are the IBM 4758 (Dyer et al., 2001) and the Utimaco Cryptoserver (Utimaco IS GmbH, 2018). HSMs are commonly used to protect long-term secret cryptographic keys in banking applications and are used for protecting secrets by Amazon’s Cloud HSM (Amazon Web Services, 2013). In comparison to the other solutions, HSMs are a rather expensive technology.

3.3.2 Sealed cloud. Also close in spirit to our definition of sealed computation is Sealed Cloud provided by Uniscon GmbH (2017). In that technology, a sealed segment is a computer within a physical container (usually a server rack or an entire hall of server cages) that is protected by means of state-of-the-art perimeter security (including Firewalls, IDS and monitoring) (Jaeger et al., 2014). This includes various types of sensors and detectors that capture unauthorized physical access (similar to burglary protection systems in cars). In case of attack detection, the system sets off an alarm and triggers internal protection...
procedures that include erasing sensitive data and wiping encryption keys. The server itself uses secure boot and a sealed key service distribution to provide software integrity.

3.3.3 Intel Software Guard eXtension. Software Guard eXtension (SGX) is an Intel technology that allows application developers to protect code and data from modification and disclosure via the use of protected execution areas in memory (enclaves). SGX technology supports data sealing and attestation mechanisms. Moreover, it allows lower trust in the operator and/or environment that runs the application (Anati et al., 2013). Although SGX is a relatively new technology, it has gained considerable attention to be used in clouds. Research prototypes of isolated execution environments based on SGX have been presented by Baumann et al. (2014) and Schuster et al. (2015). A commercial example of the use of SGX is the Fortanix key management service (Kumar et al., 2017) because it leverages Intel SGX to provide runtime encryption to protect data and applications. However, several side channel attacks on SGX enclaves have been published (Xu et al., 2015; Shinde et al., 2016; Götzfried et al., 2017; Hähnel et al., 2017) such that it is not clear how well SGX can realize the Black-box property of Definition 1.

3.3.4 AMD secure encrypted virtualization. Secure encrypted virtualization (SEV) is a security technology designed by AMD for virtualized environments (AMD, 2018). Similar to Intel SGX, it aims at isolating execution between low privileged code and high privileged code; however, at the level of virtualization hypervisor, to protect guest machines' execution. It integrates Secure Memory Encryption within the virtualization extensions of AMD-V; therefore, it also protects read/write data from/to memory (Kaplan et al., 2016). Moreover, SEV firmware provides three main properties: platform authenticity, attestation of launched guest's virtual machine and data confidentiality. Because AMD SEV is not readily available for testing, it is unclear how well it fits into the definition of sealed computation. It is to be expected that, similar to Intel SGX, many side channels remain in the hypervisor; therefore, it is also not clear, how well SEV realizes the Black-box property of Definition 1.

4. System and attacker model
We now describe the system and attacker model in more detail.

4.1 Participants
For a general cloud-based application system model, our approach assumes the following main participants (referred to as entities or parties interchangeably):

- **Data Prosumer (DP):** The DP is a producer and/or consumer of data at the same time, i.e. it produces input data and/or has an interest to consume the computed results. The way in which data is processed by the application is described by the DP in the form of a functional specification.

- **Application Software Provider (ASP):** The ASP develops and maintains the analytics software that processes the data in the cloud and computes desired results according to the functional specification.

- **Cloud Provider (CP):** The CP provides the cloud service that includes the hardware infrastructure, the software, and all associated configuration, administration and deployment tasks. The CP is also responsible for the security of the system as well as its availability towards the DP.

- **Auditing Party (AP):** The AP is an independent party that helps to ensure the integrity of the hardware and software before the system becomes operational. We sometimes simply refer to the AP as the auditor.
4.2 User security and privacy requirements

The desired security and privacy requirements desired for the parties are described in more detail here. They were summarized via brain storming and based on previous experience in security and privacy requirements[1] for IT systems. Every requirement has a name that is prefixed by the corresponding participant role.

Definition 2 (User Security and Privacy Requirements).

The participants have the following security requirements:

- **DP-Privacy**: The DP requires that data remains confidential to any other party, i.e., neither CP nor ASP, AP or SCP can learn anything about the data[2].

- **DP-Integrity**: Results that are obtained from the system by the DP are correctly computed on data as provided according to the functional specification. DP-Integrity covers both data storage and processing integrity.

- **ASP-Integrity**: The analytics software provided by the ASP is executed in an unmodified form within the system. Note that ASP-Integrity does not imply DP-Integrity since the latter refers also to data.

- **ASP-Confidentiality**: No other party except the AP is able to learn about the analytics software developed by the ASP apart from what is described in the functional specification.

Revisiting the security and privacy requirements in the example of UBI above (Figure 1), the drivers agree that their ranking is computed, but they want their individual usage data to remain confidential toward the insurance company and the cloud provider (i.e. DP-Privacy).

Regarding DP-Integrity and ASP-Integrity, the insurance company, service provider and the drivers should have a guarantee that the CP does not change data or code in any unauthorized way. Concerning ASP-Confidentiality, the software providers want to protect their intellectual property from other parties; therefore, the software that is deployed in the cloud should be protected.

4.3 Attacker model

In this section, we formulate the attacker model. First, the ways in which individual participants may maliciously misbehave are described (the local attacker assumption). Then, we define a condition that restricts the number of parties that may act maliciously (the global attacker assumption). The participants may act as follows:

- **Application Software Provider**: The ASP could provide an analytics software that leaks information about the processed data, thus violating DP-Privacy. Also, the ASP could violate DP-Integrity by providing software that incorrectly computes the results, i.e. computes the results not according to the functional specification provided by the DP.

- **Auditing Party**: During verification, the AP could try to add functionality to the system to leak information about the processed data and/or the software, thereby violating DP-Privacy or ASP-Confidentiality directly.

- **Cloud Provider**: The CP could leak any software to which it has access to a malicious party, thereby violating ASP-Confidentiality. The CP has physical access to the...
mechanism provided by the SCP; therefore, it may attempt to access and/or modify data that is stored/processed, thus violating ASP-Integrity, DP-Integrity or DP-Privacy. We assume, however, that the CP protects its systems from unwanted interference and misuse by external attackers that are not specific to our scenario. Therefore, these attacks are excluded from consideration in this work.

- **Sealed Computation Provider.** The SCP could provide an incorrect sealed computation mechanism, i.e. a mechanism that has back-doors or vulnerabilities that enable changing code and data within the sealed execution container (thus violating ASP-Integrity or DP-Integrity), or a system that leaks code or data that violates ASP-Confidentiality or DP-Privacy.

If any party acts in ways described above, we say that this party acts maliciously. A party that does not act maliciously is considered honest. For reasons of simplicity, the DP is excluded from our attacker model. Typical misbehavior of the DP can be giving a wrong functional specification, providing false data or revealing the received results to any other party. Correct behavior in this respect cannot be enforced using a trustworthy cloud service. Therefore, the DP is assumed to always be honest.

The global attacker assumption, i.e. a restriction on the number of parties that may act maliciously is formulated as follows: either the AP or both SCP and ASP are honest. More precisely, if the identifiers are taken as Boolean predicates of whether they are acting honestly or not, then the global attacker assumption is satisfied if the following condition holds:

$$AP \vee (SCP \land ASP)$$

Note that the condition is independent of the actions of the CP and that it does not state which party exactly acts maliciously (AP, SCP or ASP).

4.4 Availability of remote attestation
To establish trust, it is often necessary to use mechanisms for remote attestation. Following the terminology of Coker et al. (2011), attestation is the activity of making a claim to an appraiser about the properties of a target by supplying evidence supporting that claim. An attester is a party performing this activity. The result of an attestation depends on a mixture of facts that the appraiser can directly check on the evidence provided by the attester (e.g. cryptographic signatures) and trust in the attester itself (the mechanism by which the evidence was generated). Any party being part of a remote attestation has the requirement that the directly verifiable part of the attestation works as expected. In practice, this means that the used cryptography (e.g. digital signatures) is secure and that honest parties protect their cryptographic secrets.

5. Combining sealed computation with an auditor
One application of sealed computation in cloud computing would be for the CP to offer a mechanism to its “clients,” DP and ASP, to run applications in a sealed container. In this case, SCP and CP would be the same party. However, note that utilizing sealed computation alone is not sufficient to ensure the participants' security and privacy requirements because:
sealed computation does not guarantee anything before sealing takes place; and
the mechanism of sealed computation cannot be trusted without means to verify its function.

Therefore, in the following discussion, we consider CP and SCP as independent parties.

5.1 The role of an auditor
The sealed computation is combined with the role of an auditing party AP to establish the requirements described in Definition 2. In general, auditors are known to usually perform independent checks and assess other entities in terms of service, performance, security, data privacy and system operations (Habib Mahbub et al., 2012). We use the AP to both guarantee the functionality of the sealing mechanism provided by the SCP and to verify the functionality of the analytical software provided by the ASP. Once sealing takes place, the mechanism of sealed computation ensures continued trust in the system without having to interact with the AP anymore.

The auditor is not allowed to add or modify functionality in the system. This is ensured by a mutual checking procedure described below. The AP, however, has to enable a possibility of attestation that is independent of the SCP. This can be realized by either providing an independent mechanism or (better) by adequately configuring an attestation technique that is already presented in the sealed computation technology (e.g. by embedding a secret within the physical container of sealed computing).

Figure 2 illustrates the structural model with the roles and responsibilities of each participant. The idea is to base the functioning of the system on the assumption that either the auditor or all parties checked by the auditor are honest during critical phases of system operation. While commonly the DP has to exclusively trust the CP, it now must rely on trust either in the SCP and ASP or the AP (a condition expressed in our global attacker assumption above).

To illustrate the different roles using our introductory UBI scenario, the drivers and the insurance company share the role of the DP. The insurance company defines the functional specification of the driver ranking based on which the ASP develops the analytical software. The SCP could be a provider of the sealed computation container (such as an HSM) and the

Notes: The ASP provides software run within a sealed computation, a mechanism provided by the SCP and hosted by the CP. The AP performs an independent verification of the analytic software and the sealed computation container and enables mechanisms for the DP to remotely check its integrity.
AP would be a company such as a certified public accountant, which is able to perform code and security audits on hardware and software. The SCP is assumed to have appropriate security mechanisms in place against attacks by parties not considered above (e.g. hackers and cybercriminals). Regarding remote attestation, the HSM provides certificates using which attestation evidence generated by the HSM can be verified (Wagner et al., 2013).

5.2 Trust establishment procedure

For simplicity and comprehension of discussion, we distinguish the execution lifetime of the system model into mutually exclusive phases: the checking phase and the running phase. During the checking phase, the trust establishment procedure takes place, while the running phase begins with the service start-up. During the running phase, the DP can upload data and get results and the CP operates the cloud system.

The exact actions and obligations of the participants and interplay among each other are described as trust establishment procedure below. This procedure can be regarded as a form of procedural requirement, which in combination with the technical requirements of sealed computation, allows to fulfill the user requirements.

Definition 3 (Trust establishment procedure with mutual checking)

The participants undergo the following procedure:

1. Trust establishment in analytics software:
   - The ASP prepares the analytics software ready to be deployed.
   - The AP verifies whether the analytics software satisfies the functional specification and does not leak any information about the processed data.
   - At the same time, the ASP ensures that the AP does not change any functionality of the analytics software.
   - As a result of this procedure, ASP and AP generate public evidence to be produced by an attestation mechanism by which it can be verified that the checked version of the software is running in the sealed computation (e.g., a hash of the binary code that can be attested).

2. Trust establishment in sealed computation mechanism:
   - Before the sealed computation system is shipped and deployed, regardless of the deployment model, the SCP prepares the sealed computation mechanism (hardware and software, including the possibility for confidential software deployment).
   - The AP verifies (off-line) the integrity of the sealed computation mechanism, i.e. the entire hardware and software system. This includes a physical check for the security measures, policy compliance, data security and data privacy, as well as functional checks of the confidential software deployment mechanism.
   - At the same time, the SCP ensures that the AP is not adding new functionality during these checks, i.e., the AP is behaving according to the auditing procedure specifications.
   - The AP and the SCP generate public evidence that enables attestation of the sealed computation mechanism, e.g., by embedding independent private keys within the sealed computation container to which they possess the corresponding public keys.

3. The sealing mechanism is started in the presence of AP and SCP. At this time, the auditing procedure ends and both SCP and AP can leave the deployment site that is run by the CP.
(4) Using the confidential deployment procedure, ASP loads the code that was checked by the AP in Step 1 above.

(5) The AP and the SCP must be present any time when the system and/or the sealed computation mechanism is reset/restarted, is under maintenance or shall be changed. In such cases, the AP and the SCP must re-check the system and both must re-enable the attestation mechanism as described in the above procedure.

The trust establishment procedure is illustrated as an UML diagram in Figure 3. The result of this procedure is two pieces of public evidence that all parties can use to verify their requirements:

1. Public evidence provided by AP and SCP that DP, CP and ASP can use to verify that an instance of sealed computation is running.
2. Public evidence provided by AP and ASP that can be used to verify that a particular software is running within the sealed computation.

6. Security analysis and discussion

6.1 Preliminary observation

To argue that the security and privacy requirements from Definition 2 are met, we make the following introductory observation. The sealed computation mechanism defined in
Definition 1 will not be in the running phase if the ASP software or the sealed computation mechanism is not correct.

To examine this, we make a case distinction based on the global attacker assumption, which states that all parties can act maliciously as long as the global attacker assumption is satisfied, i.e. either the AP or both the ASP and the SCP behave honestly. There are three possible cases for parties to act maliciously during the checking phase when the trust establishment procedure (Definition 3) takes place:

1. **The ASP is malicious**: If the ASP is malicious, then the AP must be honest. So, if the ASP acts maliciously and implements an incorrect software then the checking procedure (Step 1b) mandates that the AP checks the software correctness. Because the AP is honest, it will detect the incorrectness of software, the check will fail and the running phase will not take place.

2. **The SCP is malicious**: If the SCP is malicious, then the AP must be honest. So, if the SCP is not honest, the sealing container may not be implemented correctly. However, the checking procedure (Step 2b) requires the AP to check whether the sealed computation requirements are met. Because the AP is honest, it will detect incorrectness and the running phase will not be entered.

3. **The AP is malicious**: If the AP is malicious, then the ASP and the SCP are both honest. In this case, the analytics software and the sealed computation mechanism are correct from the beginning. Furthermore, the mutual checking procedure (Steps 1c and 2c) requires that both ASP and SCP ensure that the AP does not manipulate the functionality of the analytics software or the sealed computation mechanism. Thus, if the running phase is entered, both the sealed computation mechanism and the analytics software are correct.

Therefore, under the attacker assumption, the establishment procedure guarantees that the system will not enter the running phase unless it is working properly as defined in the specification. Subsequently, during the running phase, the sealed computation mechanism takes over to guarantee the desired requirements. To argue for the fulfillment of ASP-Integrity and DP-Integrity, the sealing and tamper-resistance requirements of the sealed computation ensure that content (data and code) in the sealed container cannot be improperly modified. Furthermore, the black-box requirement restricts information flow such that DP-Privacy and (assuming confidential deployment) ASP-Confidentiality are maintained. The requirements are discussed in more details below.

### 6.2 Detailed analysis

The following detailed analysis determines what things can go wrong and discusses the possible consequences of participants’ malicious acts on the general requirements (so-called “what-if security analysis”). Table I summarizes those parties that can potentially affect the general requirements.

#### 6.2.1 Integrity requirements

We first consider ASP-Integrity and investigate what may happen if other parties act maliciously during the checking and/or running phase. What if the AP is behaving maliciously and tries to modify the application software provided by the ASP?

The trust establishment procedure, Definition 3, requires the AP to check that the software is working according to the specifications. Assuming the global attack assumption holds, if the AP is behaving maliciously and tries to modify the software, then the mutual checking procedure will not pass correctly and consequently the service will not be started.
initially. Moreover, based on the confidential deployment that is derived from sealed computation properties, it is guaranteed that the attested version of the software, during running time, is the same version of the software deployed by the ASP, i.e. correct system setup results in software integrity during the Running phase.

What if CP or ASP behave maliciously and try to modify the application software during the running phase?

*Sealing and Tamper-resistance* properties guarantee that no party (neither the CP nor ASP or external entity) can neither read nor modify the SW after it has been sealed to the environment. Thus, during the running phase, the sealed computation properties guarantee that the software is running based on the specification and cannot be manipulated without being noticed. Thus, the proposed approach provides means to ensure that ASP-Integrity is fulfilled in the system without the need to trust single entity.

We now turn to DP-Integrity. Generally, the results may be incorrectly computed because of two primary reasons: the application software is implemented correctly but it has been manipulated in the runtime environment or the application software is not implemented correctly (regardless whether intentionally or not). Assuming the application software is implemented correctly, similar to the ASP-Integrity analysis above, it is shown that the software’s integrity is satisfied by the approach. Hence, data integrity is provided as a result of the *Sealing and Tamper-resistance* properties of sealed computation.

What if the auditor is behaving maliciously during the checking phase or the sealed computation was not implemented correctly (intentionally or not)? Following from the attacker assumption during the checking phase and based on the trust establishment procedure (Definition 3), the service would not be started initially in both cases. Therefore, the model guarantees that once the systems start (correctly), the deployed software will behave as specified during the running phase. This implies that DP-Integrity is fulfilled.

6.2.2 Confidentiality requirements. We first look at DP-Privacy. Data privacy may be violated because of a compromised infrastructure, compromised software or changes made by AP.

What if SCP sets up an infrastructure that does not fulfill the sealed computation properties or the ASP deploys software that leaks data?

During the mutual checking phase, the AP detects inappropriate implementations; therefore, it prevents the system startup. Similarly, the mutual checking prevents AP from embedding changes that might lead to data leakage.

However, after the system started, sealed computation properties *Sealing and Tamper-resistance* ensures that, during the running phase, the ASP cannot change the software after it has been deployed; thus, no data are exposed to the ASP. Furthermore, the *Black-box* property guarantees that information flow is restricted to the interface specification. As a result, consumers shall receive results only as agreed in the specification. We now turn to
ASP-Confidentiality. ASP-Confidentiality might be breached either if the runtime container or one of the privileged participants leaks details about the application software.

What if the sealed computation mechanism is not implemented correctly?

Based on the model setup, the AP verifies the sealed computation mechanism to ensure that it is correctly implemented. Moreover, based on the properties of sealed computation, the Black-Box property ensures that nothing can be learned about the software apart from what is defined in the functional specification. Moreover, the Tamper-resistance property guarantees that code cannot be retrieved even in case of termination.

Furthermore, the devised confidential deployment procedure protects code (even if it is decrypted within the sealed computation container) so that other parties cannot recognize the code during deployment. The attestation confirms not only of ASP but also of AP guarantee that the software is running only for the intended execution container and certainly was checked by the AP; therefore, the sealed computation is trustworthy.

Consequently, a correct implementation of the approach ensures the confidentiality and data privacy in the system under the given assumptions.

6.3 Discussion

While our results are conceptual, they provide a preliminary guideline of building a trustworthy cloud computing service in which cloud customers can trust that CPs and operators cannot access their data and code. The concept of sealed computation can be regarded as extending the concept of sealed storage [defined by Morris (1973)] for storage and computation. To the best of our knowledge, there has been no attempt to more precisely describe the properties and requirements sealed computation before. Any computational implementation that satisfies the requirements defined in Definition 1 can be considered a sealed computation mechanism. However, in practice, one may argue that any assumption, such as the security of cryptography or requirements like Black-box of any hardware device, only hold with a certain probability; therefore, the guarantees in practice never hold 100 per cent.

One may also argue that many parts of the procedures described in Definition 3 are rather hypothetical and cannot be realized fully in practice. For example, the AP is assumed to perfectly verify the correctness of the software of the ASP (in Step 1b) against the functional specification. While software verification has come a long way, it still is restricted by the size and complexity of the software system.

Another example that appears far from practice is the statement that the AP can verify the correctness of the sealed computation container (hardware and software) provided by the SCP (in Step 2b). It is well known that the production of hardware is a very complex process involving lots of different technologies. The resulting chips are rather transparent and need complex validation equipment to be checked.

Still useful insights can be inferred from the proposed approach. While the AP is one party in our model, it can consist of multiple independent auditing actors correctly, e.g. different companies that check all independent parts of the system and mutually certify the results toward each other. The collection of auditors in its entirety then forms the AP, meaning also that all “sub-auditors” must behave correctly for the AP to be regarded as honest. In practice, these sub-auditors are even often part of the same company, although in different parts that are independent of each other (such as software development and testing departments).

Another insight is that it is possible to delegate security enforcement to trusted hardware without having to trust a single entity. However, during the checking phase, the AP must be continuously present until the sealed computation container runs, and it
must be possible to establish attestation evidence, which is *independently supported* by the AP and the SCP (for the sealed computation container) and by the AP and the ASP (for the analytics software). These points result from the requirement of mutual checking, i.e. not only does the AP verify the actions of ASP/SCP but also ASP/SCP need to prevent the AP from slipping in new functionality to software and hardware, a detail that is often overlooked or (unconvincingly) excluded by the assumption that the AP is always honest. Being able to embed shared attestation credentials of mutually untrusted parties in a single trusted hardware container is a feature that is, to our knowledge, not supported by any currently available trusted computing mechanism (*Maene et al.*, 2017). Thus, the proposed approach presents an idealized version of system construction and deployment processes that can serve as an orientation for practice for achieving a trustworthy service.

7. Conclusions and future work
We introduced the sealed computation concept definition and proposed a mutual checking procedure with an auditor role during setup time to provide an increased level of security and trust in cloud scenarios. The sealed computation concept abstracts from trusted hardware technologies such as HSMs. The auditor is an abstraction of policies and procedures that increase trust in a single party.

We believe that the abstract system model using the auditor as an additional role is a good approach for medium-size and large cloud deployments rather than running their own private cloud. While the existence of the role of auditor may be intuitive, on the one hand, it is not clear whether the concept is necessary, i.e. whether any technique that distributes trust can simulate the auditor as described above. On the other hand, practical methods for auditing could be investigated. Furthermore, a more rigid formalization for the attestation verification can be attempted in future.

Notes
1. Here we use "security and privacy requirements" in the general aspects of the terms. The detailed privacy requirements such as: minimized data disclosure, access control and authorization, and anonymized data, etc. are out of the scope of this paper. The current stage of this work focuses on abstract requirements for a trust- worthy cloud service. Then in a further stage the detailed requirements can be discussed at the level of SW application design.

2. Here we explicitly use the term Privacy and not Confidentiality to emphasize end users’ privacy (as individuals) against the providers and operators of the system (as organizations).

References


**Corresponding author**

Lamya Abdullah can be contacted at: lamya.abdullah@uniscon.de