Architectural and API proposal for the secure processing stack

D3.1

Project reference no. 653884

February 2016

European Commission

Horizon 2020
European Union funding
for Research & Innovation

Document information

Scheduled delivery	29.02.2016
Actual delivery	01.03.2016
Version	1.0
Responsible Partner	INESC-TEC

Dissemination level

Public

Revision history

Date	Editor	Status	Version	Changes
07.02.2016	Francisco Maia, João Paulo	Draft	0.1	First Draft
10.02.2016	Bernardo Portela, Dan Bogdanov	Draft	0.2	Architecture definition
17.02.2016	João Paulo, Francisco Maia, Bernado Portela, Dan Bogdanov	Submission	0.3	First Submission
18.02.2016	Miguel Pardal, Miguel Correia	Revision	0.4	INESC Revision
19.02.2016	João Paulo, Francisco Maia, Bernado Portela, Dan Bogdanov	Revision	0.5	Changes according to revision.
26.02.2016	Valerio Schiavoni and Hugues Mercier	Revision	0.6	UniNE Revision
29.02.2016	João Paulo, Francisco Maia, Karl Tarbe	Final version	1	Final version of the deliverable
31.08.2016	João Paulo, Hugues Mercier	Final version	1.1	Comments clean-up

Contributors

Bernado Portela (INESC-TEC)
Dan Bogdanov (CYBERNETICA)
Karl Tarbe **(**CYBERNETICA)
Reimo Rebane **(**CYBERNETICA)
Francisco Maia (INESC-TEC)
João Paulo (INESC-TEC)
Rogério Pontes (INESC-TEC)

Internal reviewers

Miguel Pardal (INESC-ID)
Miguel Correia (INESC-ID)
Valerio Schiavoni (UniNE)
Hugues Mercier (UniNE)

Acknowledgements

This project is partially funded by the European Commission Horizon 2020 work programme under grant agreement no. 653884.

More information

Additional information and public deliverables of SafeCloud can be found at http://www.safecloud-project.eu

Glossary of acronyms

Acronym	Definition
D	Deliverable
DoA	Description of Work
EC	European Commission
PM	Project Manager
PO	Project Officer
WP	Work Package
SQL	Structured Query Language
API	Application Interface
ICT	Information and Communications Technology
IT	Information Technology
ACID	Atomicity, Consistency, Isolation, Durability

Table of contents

1 Executive summary

The SafeCloud framework has three layers, as depicted in Figure 1 presents the secure communication technology that ensures that parties can form secure channels that are even harder to eavesdrop or block; SafeCloud secure storage technologies go beyond encryption and offer protection against deletion and destruction; SafeCloud secure query technology extends cryptographic protection from database storage to data processing. An ICT system developer can combine solutions from the three layers to provide beyond-state-of-the-art protection for personal data or business secrets.

Secure communication	State of the art: TLS secure channels	**Solution:**	**Vulnerability-tolerant channels**	**Protected channels**	**Route-aware channels**
		Gives:	Tolerance to vulnerabilities in components	Decreased risk of fake certificates; resistance to port scans and enumeration of network infrastructure	Improved confidentiality with warnings about route hijacking and making harder access to communication
		API:	Extended secure socket API	Extended secure socket API	Extended secure socket API
		Provided by:	INESC-ID, TUM	INESC-ID, TUM	INESC-ID, TUM
Secure storage	State of the art: Encrypted storage	**Solution:**	**Distributed encrypted filesystem**	**Long-term distributed encrypted data storage**	**Secure block storage**
		Gives:	Encrypted file storage	Entangled immutable data storage for protection against tampering and censorship	Block storage on individual data centers
		API:	POSIX	REST (S3 or similar)	Key/value
		Provided by:	UniNE, INESC-ID	UniNE, INESC-TEC	UniNE, INESC-TEC
Secure queries	State of the art: CryptDB	**Solution:**	**Secure processing in a single untrusted domain**	**Secure processing in multiple untrusted domains**	**Secure processing in multiple untrusted domains with untrusted clients**
		Gives:	Privacy of data values against the server, optional key privacy	Privacy of key and data values against the servers	Privacy of key and data values against the servers and clients
		API:	SQL	SQL	SQL
		Provided by:	INESC-TEC	INESC-TEC, Cyber	Cyber

Figure 1:The SafeCloud framework

The objective of work package 3 (WP3) is centred on the practical and theoretical investigation of mechanisms for data distribution and processing, in a way that allows for security assurances to be provided whilst maintaining computation capabilities. It is, therefore, focused on the secure query or secure processing layer (bottom of the table).

This deliverable (D3.1) is the first of WP3, to be delivered on month 6. We describe the general architecture and three different solutions for protecting private data during SQL query execution. The three solutions provide varying levels of security and performance. For example, solutions 1 and 2 can achieve higher performance levels, but assume that the query source can be trusted with all the data. However, solution 3 no longer assumes the query source can be trusted and uses a stronger encryption scheme. This last solution is thus very suitable for securely processing data collected from multiple sources.

This document is structured as follows. Section 2 will describe the high level architecture that is common to the three solutions proposed in this WP. Sections 3, 4 and 5 match solutions 1, 2 and 3 respectively, detailing what are the proposed approaches, the associated motivation, and how they fit the general architecture. Finally, Section 5 concludes the deliverable with a general overview of its contents.

The secure processing layer of SafeCloud is focused in investigating, evaluating and implementing solutions that provide adequate trade-offs between performance and security levels, leveraging trust models and computational capacities required from the trusted and untrusted deployments. However, real-world scenarios and their practical deployments can widely vary, so it is highly unlikely to expect a single solution to fit every particular case. With that in mind, alternatively to what could be achieved with a single and more focused approach, we propose three solutions for privacy-preserving storage that are suited for applications with distinct security and performance requirements. This decision allows extending the practicability and applicability of the secure processing layer. The compromises in designing our proposed solutions are depicted in Figure 2.

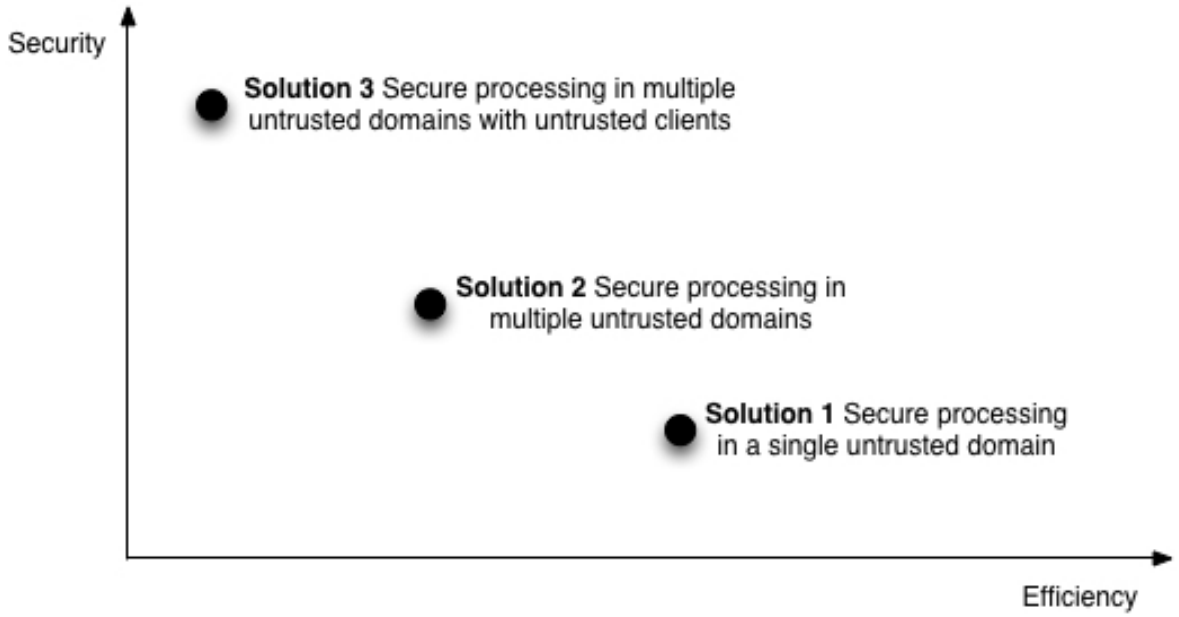

Figure 2: SafeCloud solution range and trade-offs

The first solution describes a simple cloud setting, in which clients have access to a local secure SQL engine, and want to outsource data storage and computation to a remote NoSQL engine. When this deployment is considered to be untrusted (as is the case when the one holding the data is responsible for handling sensitive information), the issue of security and privacy should be taken into consideration for protocol development, while still ensuring that the solution remains practical and applicable in real world scenarios. For instance, the choice of using standard cryptographic techniques on all data allows for achieving reasonably strong security guarantees, however it disables computation over the data, which in many cases defeats the purpose of employing a cloud provider. As such, our solution proposes the usage of different techniques for different levels of data sensitivity, aiming to provide "good enough" security assurances while enabling efficient computation over stored data.

The second solution aims to leverage multiple untrusted non-colluding cloud providers to enhance security guarantees without losing computational functionalities. It assumes the same client with a secure SQL engine, but now it communicates with several

untrusted NoSQL cloud providers. By employing cryptographic techniques that rely on distributing information over these non-colluding entities, state-of-the-art protocols for multi-party computation can enable secure function execution over data that is kept confidential from every individual cloud provider. The performance values are expected to be lower than the ones from the previous approach, however this setting allows for computation over data that is kept information-theoretically secure at all times.

The third and final solution is different from the previous ones in two major points. First, it reduces the computational power required from the client, by allowing the SQL engine to be run in the untrusted environment. Second, it allows for multiple clients to process over the stored data without assuming that all of them are trusted (i.e. that they trust each other). The latter opens the possibility for several users with sensitive data to jointly store data with security guarantees, while allowing computations (such as data analytics) to be executed over the collected information. This is particularly impactful for clients such as hospitals and governmental institutions, which manage large amounts sensitive data but also would highly benefit from joint statistical analysis over it.

2 General architecture for privacy-preserving computation

Cloud Computing has become a key paradigm for application and service deployment. Companies benefit from the pay-as-you-go model and avoid high upfront costs with IT infrastructures. The move to the cloud paradigm also targets performance, since the cloud resources can adapt and scale according to the demand. Such benefits come at the expense of the company's control over the infrastructure and require the company to trust on a third party - a cloud provider - with the company's software stack and data. Therefore, taking advantage of cloud capabilities is non-trivial as this raises several security and privacy concerns, as well as architectural challenges [AFG+10, MSL+11].

The lack of secure and private solutions for data storage and computation in current cloud environments demands different deployment compromises, which in turn lead to suboptimal systems. In particular, even if companies can largely benefit from a move to cloud services, they avoid doing so due to security concerns [DROPBOX12]. As a consequence, the solution is to continue relying on-premises' infrastructures to hold the most critical parts of the company's system. The resulting system, therefore, consists of a hybrid infrastructure composed by on-premises storage and computation and by offloaded storage and computing capabilities to the cloud. Considering this setting, different compromises can be obtained according to the amount of offloaded responsibilities, which are related to the trust level each company attributes to the cloud provider.

In this section, we present a general architecture for SafeCloud solutions to privacy-preserving storage and computation1. Heavily rooted in the cloud computing paradigm, such architecture comprises a set of comprehensive solutions to private and secure data storage and computation considering the project use cases. Each solution offers practical trade-offs between performance and security as well as between processing power and infrastructure trustworthiness.

We begin by describing the assumptions considered in the design of SafeCloud's privacy-preserving processing solutions by presenting the planned deployment scenario. We proceed presenting the general architecture that abstracts each one of the different solutions and covers their key components. Finally, each SafeCloud solution is presented detailing the security and trust models it considers, and a representative use case of its applicability.

2.1 Deployment Scenario

Traditionally, companies and organisations internally develop and maintain their IT infrastructure to support their activities. Such practice has, often, very high costs associated to the infrastructure itself as well as deriving from problems such as resource over-provisioning or under-provisioning. More recently, the cloud computing paradigm changed how applications are being deployed and how IT infrastructures are managed.

[1] Data computation will also be referred as data processing in this document. In the context of SafeCloud, data computation or processing is related to the knowledge that can be obtained from processing a given set of data. In particular, this encompasses data aggregation, data filtering and data analytics to mention a few.

The core idea of this new paradigm is to leverage large infrastructure deployments in a pay-as-you-go model where clients can dynamically allocate and release resources from such deployments and pay only for what they use. A direct consequence of this model is that, now, a company can have access to large infrastructures and set-up large deployments without incurring in significant up-front costs [AFG+10].

Modern application and service deployments can fall into one of three categories. The first one includes in-house deployments where everything from IT infrastructure to software is designed, developed, and maintained within the company or organization facilities. The second one is the opposite situation where everything is handed over to a cloud or a group of cloud providers and the organization or company simply manages its infrastructure remotely. The third category includes any compromise between the two previous ones. This last category comprehends, for instance, situations where critical services are kept on premises and complementary services are deployed in the cloud for economical benefits.

Considering the third category of services, different trade-offs are possible between what to deploy on premises and on the cloud. These are security trade-offs but also performance trade-offs. For instance, if the necessary computational and storage resources exist on premises, doing all the computation there may result in better performance due to the fact that security mechanisms benefit from having a powerful trusted environment, and do not have a heavy impact over the processing mechanisms themselves. On the other hand, the cloud computing model can be more attractive in terms of cost since the pay-as-you-go model allows the allocation of computational resources only when they are required and, for large-scale applications, the cloud's elasticity characteristics make it an ideal choice.

However, offloading data storage and processing to an untrusted environment while keeping data private, requires complex security mechanisms that may lead to a decrease in applications performance [CK08].

Figure 3 depicts an overview of such a deployment. Typically, a proxy is required to integrate both the local and the remote deployments. This proxy is still considered to be located on premises.

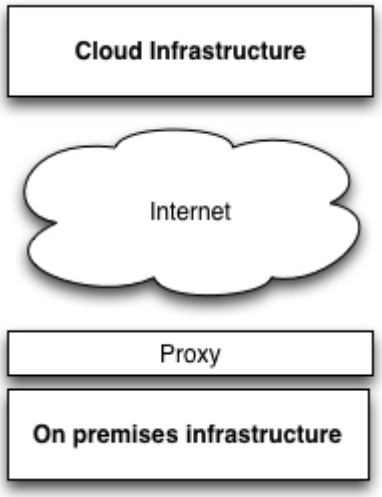

Figure 3: Deployment scenario

Given the distinct advantages of deploying applications in an on premises infrastructure and on a cloud infrastructure, in SafeCloud we assume the following trust considerations. The on premises infrastructure is assumed to be completely trusted, which implies that it is allowed to receive, handle and process sensitive data in the clear (this does not mean that it cannot be attacked or have malicious insiders, only that these are not the problems SafeCloud aims to solve). On the other hand, cloud providers are not allowed to deviate from the system's specified behavior, but may attempt to compute over collected data to ascertain additional information. This assumption that cloud providers can compromise the privacy of data stored and computed at their infrastructures is a core concern of the SafeCloud project and of this work package. As a consequence, data processing on premises can be done over in the clear data and without access restrictions. In the cloud, however, sensitive data needs to be protected and processing may be impaired in order to achieve such protection.

As an example, take a clinical analysis laboratory that processes and stores sensible data from patients, and where an on premises cluster infrastructure, with the necessary security mechanisms to be safe against external attackers, is available. If the number of patients is very large or grows significantly, and the on premises cluster storage and processing capabilities are no longer sufficient, it may be desirable to off-load the data and some of the processing capabilities to a third-party infrastructure, such as a cloud infrastructure. However, this infrastructure is no longer controlled by the applications and data rightful owners, which may lead to security breaches and undesired information disclosure. These concerns were very recently confirmed to be well justified after the recent media hype surrounding the revelations made by former NSA contractor Edward Snowden and high-profile security vulnerabilities such as the Heartbleed[D+14] bug in OpenSSL[L13]. In SafeCloud, cloud infrastructures are considered untrusted environments.

2.2 General solution API

Having described the deployment scenario considered by SafeCloud's secure processing solutions, we now focus on the interfaces that will be offered to the applications.

A major goal for any component in the SafeCloud's stack is applicability and practicality. SafeCloud aims at developing software components that can have immediate and widespread adoption from the industry. Accordingly, it becomes inevitable to pursue the definition of system interfaces that can accommodate current practices, that relieve current systems and applications of any kind of refactoring, while remaining compatible with SafeCloud's goals.

Looking at the current landscape in terms of data management systems it is possible to categorize systems into two major groups according to their interface (API). On one hand, we have traditional relational database systems that rely on a query language, typically SQL, as their interface. SQL is a widely-used programming language designed for managing data stored at relational database systems [LCW93]. The relational model used in these traditional SQL databases is well-studied and understood, and SQL is supported as the main query language of the most successful commercial database systems.

On the other hand, we have the so-called NoSQL databases [HHL+11]. NoSQL databases are becoming increasingly popular among applications that must scale for large amounts of data and clients. A major advantage of NoSQL over traditional relational databases is that scalability can be achieved in a distributed environment with commodity hardware, thus not requiring expensive dedicated hardware, which allows to lower data storage and processing costs. On the other hand, NoSQL databases have a limited set of operations when compared to relational databases. Even though NoSQL databases are now focused on offering richer query languages and that under the NoSQL designation we can find different data models (document based, key-value, graphs, etc.), the most successful and widespread NoSQL databases, in their core can be seen as an dictionary composed by several unique keys that are associated with one or more values. It is possible to insert new keys and values, to update the value of a specific key (put), to get the value for a specific key (get), to get the values for a determined range of keys (scan) and to delete existing keys (delete).

Notwithstanding the fact that NoSQL databases are successfully deployed in many contexts and applications and have undeniable popularity, a significant slice of the world's systems are, and arguably will continue to be, heavily rooted on SQL interfaces and the relational model. This situation can be partially explained by the abundance of legacy applications or simply because SQL is a significantly richer API in terms of expressiveness. Additionally, many applications require ACID guarantees and transactional support, which are key to enable support for multiple and concurrent client access to the same database. This kind of guarantees are also typical of traditional relational databases and lacking in NoSQL databases. Due to the distinct advantages of SQL and NoSQL approaches, this work package will support both types of APIs, which are further detailed next.

2.2.1 SQL API

In the design of the secure data storage and processing solutions we will consider the SQL language specification as it is defined by the American National Standards Institute (ANSI), which adopted it as a standard in 1986 [DD+93,ANSI16]. Although this standard is still being improved, most relational database solutions try to follow the same specification of the SQL language. Since this is a very rich language, having a standard specification is key for the interoperability of distinct database products. The SQL

language allows the expression of statements to change the schema of relational databases, to insert, update and delete data from the databases and also to perform a vast range of queries over the data stored at these databases.

As mentioned previously, transactional support is another important characteristic of SQL databases that, for many applications, is key for maintaining data integrity, thus ensuring that multiple clients accessing the same database see a consistent state of the database at all times.

2.2.2 NoSQL API

Under the NoSQL term it is possible to find a diverse set of scalable and widespread database technologies. In SafeCloud, we will focus on Apache HBase, one of the most successful and widely used NoSQL databases [APACHE16]. Not only it is considered one of the most mature NoSQL databases in the market but it is also the NoSQL database with which the SafeCloud partners have a longer, continued and extensive knowledge. This is a clear advantage for the success of the project. Consequently, the NoSQL interface offered by SafeCloud's solution will be heavily inspired by the HBase API which we now describe in detail.

Apache HBase is a distributed, scalable and open-source non-relational database. Inspired by Google's BigTable, it can be thought as a multi-dimensional sorted map or table. Each row is identified by a unique key and may be composed of several columns (values). HBase exposes a set of operations for data access that are quite similar to operations used in other key-value data stores:

GET - Get key-value pairs of a given row;

PUT - Insert a key-value pair for existing or new row;

SCAN - Get all key-value pairs of a row range;

DELETE - Remove one or more key-value pairs of rows;

This set of operations will constitute the SafeCloud secure processing NoSQL API.

2.2.3 Discussion

All the solutions described in this work package will provide a SQL interface, in its ANSI definition, along with transactional support. Nevertheless, it is important to notice that, in order to provide certain security and privacy guarantees while maintaining acceptable performance levels, some of the solutions may not offer complete coverage of the SQL language features. Whenever this is the case, we will detail the actual language coverage of the solution and the motivations behind those limitations.

While offering a SQL interface allows SafeCloud solutions to be attractive and highly applicable for nowadays systems and applications, specific classes of applications are left out. In particular, applications to which NoSQL databases completely fulfill their requirements. Accordingly, it should also be a concern for SafeCloud to offer solutions with this type of interface, as it expands its scope to a larger set of applications. By leveraging previous research developed on INESC TEC in the context of several EU research projects, we provide solutions capable of fulfilling both demands: a SQL and a NoSQL interface. More precisely, we aim to provide a system that offers a full ANSI SQL

API along with transactional support, built on top of a NoSQL database in an ultra-scalable design [CNIMBO09,CPAAS14].

2.3 General Solution Architecture

In this section we describe the high level architecture for SafeCloud's privacy-preserving processing solutions. The goal of providing this overview is to allow for the establishment of clear relationships between the different solutions, highlighting their differences and similarities. We consider this exercise essential for the rigorous analysis of each individual solution, and for the evaluation of its feasibility with respect to the associated application requirements.

From a very high level perspective and considering the deployment scenario of Section 2.1, we can identify two major system environments: an on-premises infrastructure and an external infrastructure, typically a Cloud provider. In both environments, the choice of system components deployed and tasks performed leads to different compromises among cost, performance, security and privacy.

To the set of components running on-premises we will call trusted deployment, and to the set of components offloaded to third-party premises we will call untrusted deployment. It is intuitive to visualize the systems that lie at the extremes of this notation scheme: on one side, we have a trusted deployment that encompasses all the desired functionalities, thus not requiring any offloading of storage or computation to the untrusted deployment; on the other side, a system where all computation is done in the untrusted deployment, possibly due to lack of system resources at the trusted side. The SafeCloud solutions represent compromises lying in between these extreme scenarios.

An important aspect to notice is that both the trusted and the untrusted deployment can be composed by one or more infrastructures. For instance, we might consider a scenario in which trust is distributed among several cloud providers, so as to avoid offloading data storage and computation to a single entity's superintendence. Having this in mind, we propose a SafeCloud general architecture for privacy-preserving data storage and processing depicted in Figure 4. We make use of squares to represent processing components, rounded edge rectangles to represent interfaces and cylinders to represent storage components. Storage components can also offer processing capabilities.

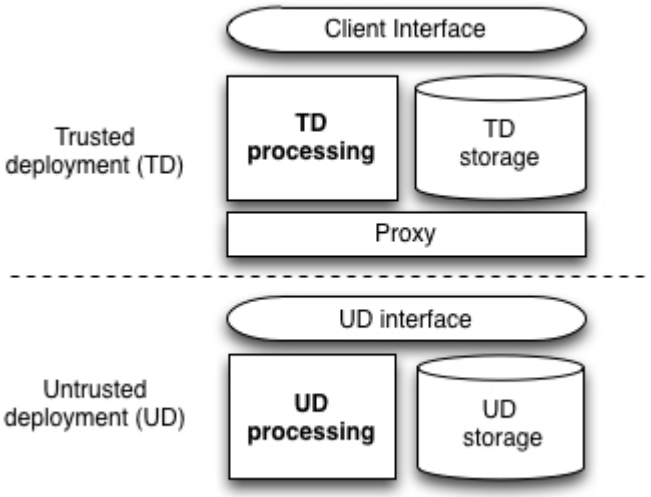

Figure 4: General Architecture

We consider a set of components in each one of the deployments. Each solution will offer a specific interface to the application employing it, the Client Interface component. As described in previous sections, this interface will typically expect SQL queries, although a NoSQL interface will also be available for applications. At the same time, an interface to the untrusted deployment is also defined for each solution. Note that this can also correspond to a SQL interface, to a NoSQL interface, or to a subset of either one of those. Additionally, we consider that some processing and storage capabilities (namely the processing and storage components) can be deployed in both the trusted and untrusted sides. Finally, we consider a proxy component that is responsible for the communication between the trusted and untrusted deployment.

In the next sections we will describe in detail the three SafeCloud solutions for privacy-preserving data storage and processing. For each of these proposed solutions, we define the processing responsibilities assigned to the trusted deployment and to the untrusted one. We will also instantiate the components of this general architecture with concrete software protocols, and detail the interfaces required for the solution deployment.

3 Solution 1: Secure processing in a single untrusted domain

3.1 Introduction

The first solution we consider is expected to achieve the highest performance when compared to the alternatives described in this document. At the same time, it is the solution where the trust model is the strongest and where the amount of storage and processing capabilities delegated to the untrusted deployment is the lowest. Briefly, solution 1 assumes a deployment setting where the trusted deployment has some computational power but scarce storage resources, while the untrusted deployment is assumed to have both complementary computational power and the necessary storage capabilities.

As an application example for this solution, consider a laboratory that owns or completely trusts a private cloud with some computational power, but is interested in offloading most of the storage effort into an untrusted cloud domain. The laboratory manages sensitive data, so it is not acceptable to have it stored/processed it in the clear outside of what the laboratory considers to be its trusted domain.

As mentioned above, we aim at offering both a SQL and a NoSQL interface to SafeCloud clients. Leveraging work from previous projects it is possible to offer a fully ANSI SQL compliant database system on top of a NoSQL database, in particular, on top of HBase. This is achieved by decomposing the database functionality into several components that address self-contained challenges and allow the whole system to scale. In detail, SQL queries are issued to an engine component that translates them to NoSQL statements handled by HBase. At the same time, transactional support is guaranteed by a separate component. Building on such architecture we will be able to offer a secure NoSQL interface to SafeCloud client applications and, on top of such interface, offer an ANSI SQL one.

In this first solution we assume that the SQL query processing and the transactional support processing can be done entirely on the trusted deployment. The cryptographic techniques are applied over the underlying NoSQL engine, in the sense that the trusted deployment will be responsible for the encryption of data before sending it to the cloud provider, to be handled by the untrusted deployment. Note that it is imperative for these techniques to allow the NoSQL system to provide the necessary functionality to support the aforementioned ANSI SQL database system.

In the next section, we describe in detail the design and architecture of Solution 1, and discuss the tradeoffs between security and performance that it entails.

3.2 Architecture and API

In Figure 5 we depict the instantiated architecture for Solution 1. It is observable that most of the processing is done in the trusted deployment. Almost all storage, on the other hand, is left under the responsibility of the cloud provider. In detail, responsibilities are assigned as follows.

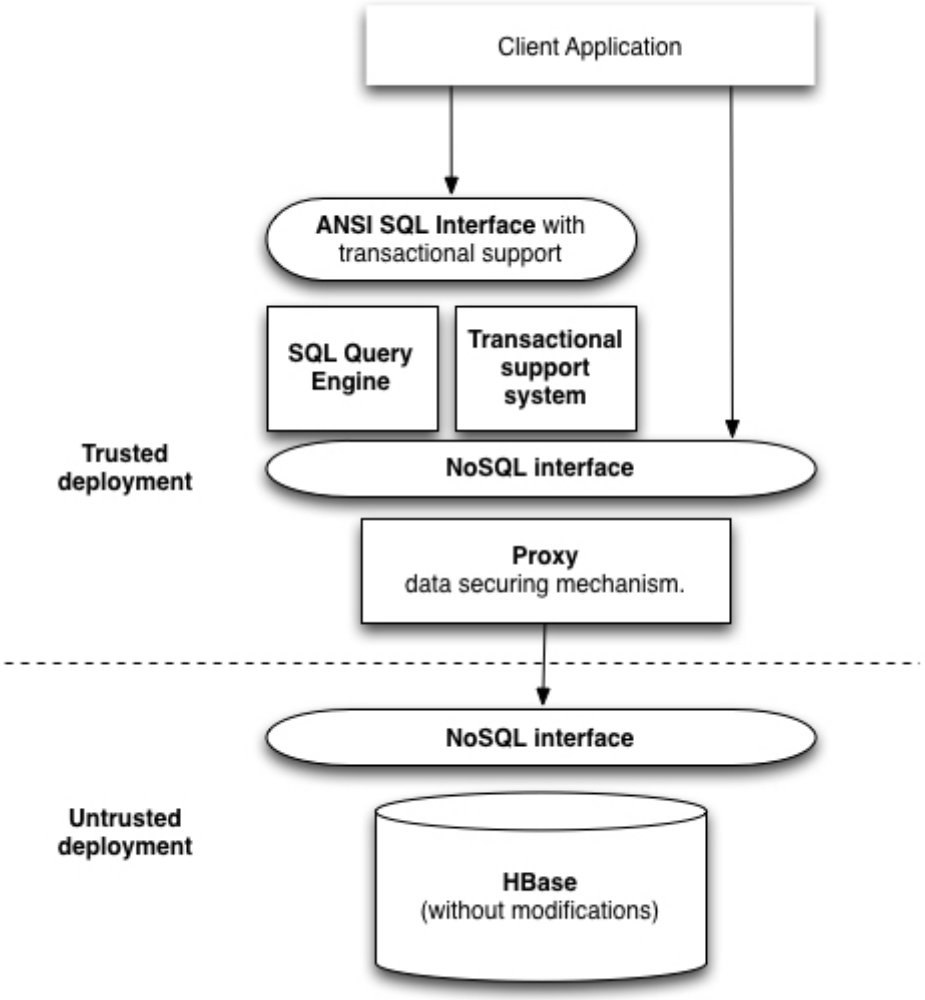

Figure 5: Solution 1 architecture

3.2.1 Trusted deployment

In the trusted domain, the SafeCloud SQL engine will process full ANSI SQL with transactional support and translate SQL queries into NoSQL statements. These statements are handed over to a proxy component that is responsible for applying the required security mechanisms to the data.

A straightforward solution regarding security could involve simply encrypting all data with standard cryptographic techniques, and therefore achieve a very strong level of security. However, this prevents the NoSQL database to be able to perform any meaningful computation over the data, implying the unfeasible scenario in which any query over the encrypted information requires the proxy to retrieve the whole encrypted database, decrypt it and process over it.

One approach would be to apply cryptographic techniques only over sensitive information, in a way that allows for the NoSQL database to perform the basic computations required. For instance, the client might consider the values of its database

to be sensitive, but allow identifiers to be left in the clear. Obviously this simple scenario is not feasible in all cases, as the data that the user considers to be non-sensitive might not suffice for our underlying NoSQL, e.g. identifiers can be left in the clear, but most of the required system queries are made over the sensitive data. One way to extend the amount of non-sensitive information could be to apply anonymization techniques over some parts of the sensitive data. Assuming that the client considers these measures to be sufficient for allowing parts of its sensitive data to be revealed to the cloud provider, this would allow for a greater portion of information to be stored in the clear.

Alternatively, we can also apply different cryptographic techniques over different levels of data sensitivity. As an example, consider order-preserving encryption (OPE) [BCL+09]. OPE is a deterministic encryption scheme whose encryption function preserves the numerical order of the plaintexts. In brief, if plaintexts m_1 and m_2 are encrypted into ciphertexts c_1 and c_2 (respectively) with OPE, and $m_1 > m_2$, then $c_1 > c_2$. This means that a database holding OPE ciphertexts can efficiently (logarithmic in the size of the database) perform range queries over its data. Techniques such as OPE reveal some properties of the data they are encrypting, but conceal the actual values. This technique is suitable when the client does not want to disclose its data but is not concerned about disclosing some of its properties (the order, in our example).

3.2.2 Untrusted deployment

The untrusted deployment mainly consists of a NoSQL database, in particular an HBase store. Data stored in this HBase deployment can be accessed via the previously described put, get and scan interface. Note that in order to achieve the desired performance, the database system takes advantage of HBase processing capabilities. Namely, HBase allows querying its data according to a set of criteria that is translated into data filters running close to the data itself. The ability to perform these processing steps is directly impacted by the type of security mechanisms deployed by the trusted proxy. Depending on the security technique used, more or less data processing can be delegated to the HBase layer. The chosen technique will, naturally, depend not only on the desired performance but also on the trust model required by the application and its data.

3.3 Discussion

Briefly, the solution described in this section offloads some of the data processing and storage done at the trusted domain to the untrusted domain by deploying a NoSQL database in the remote site. Depending on the security mechanisms deployed the amount of processing done on the untrusted site varies. Considering that doing such processing closer to the data allows for better performance, it naturally follows that the stronger the security required, the less processing can be done at the untrusted site, and the lowest the performance is expected to be. By considering different cryptographic techniques for the different levels of sensitive data, we are maximizing our performance possibilities, which could lead to solutions that are a step forward with respect to state of the art private data processing solutions.

An application example for this solution is a laboratory that owns a private cluster with some computational power, but is interested in offloading most of the storage effort into an untrusted cloud domain. The laboratory storage system deals with a certain amount of confidential information for each patient (a set of columns and respective values, for

instance), which is identified by a unique identifier (key). Publicly disclosing this identifier is not an issue because either it is already public, or the information required to disclose the actual person referred by that identifier is safely stored at the trusted on-premises deployment. Considering this scenario, offloading the data to the untrusted deployment demands for mechanisms that ensure the privacy of the patient data. In other words, mechanisms that render all values to become opaque for the cloud provider. This can be achieved by storing data in the NoSQL database using keys in the clear and values that completely hidden from the cloud provider using, for instance, standard cryptographic techniques. As a consequence, in this example, security mechanisms are quite inexpensive and, because data keys are stored in the clear, all the processing mechanisms of HBase are preserved. In particular, the ability to scan data according to user defined criteria is preserved. Moreover, as HBase is deployed in a cloud environment, this solution also allows the user to take advantage of its scalability and elasticity potential. In more detail, by allowing data keys to be stored in the clear at the untrusted NoSQL database, it is possible to retrieve specific keys and to perform filters over these keys in an efficient way and using the untrusted deployment computational power. Additionally, since NoSQL allows storing the data for each patient in a distinct column, and since these columns may be grouped in families, it is possible to have different encryption keys for each column family. This distinction allows us to establish access control over users with different permission levels (roles). For instance, in the clinical analysis laboratory example, suppose that a patient has a set of analysis that can only be seen by its personal doctor, but also has another set of documents that are visible to all the clinical staff. These documents may be stored in different column families and with different encryption keys, so that one key is available to all the clinic and allows access to one set of documents, while the other key is only available for the patient's personal doctor, that has the permission to access the other analysis set. On the other hand, if we consider an identical example with the exception that keys now may leak sensitive information and cannot be stored in the clear at the NoSQL untrusted deployment, this solution also allows applying cryptographic techniques to the keys, such as OPE, while still preserving the computation capabilities of HBase. To sum up, this solution will allow testing several cryptographic techniques in order to perform secure and efficient processing on a single untrusted NoSQL deployment. Moreover, the solution will leverage the processing capabilities available in a trusted deployment to provide a full ANSI SQL along with transactional support to applications.

The major drawback of Solution 1 is the dependence on a single cloud provider (single untrusted deployment), which limits the security techniques that can be applied. In Solution 2, SafeCloud addresses this issue by considering that data will now be stored in multiple untrusted domains.

4 Solution 2: Secure processing in multiple untrusted domains

4.1 Introduction

The major difference from the previous solution is that this approach assumes the existence of multiple untrusted non-colluding deployments. Leveraging the trust model in this way allows for the usage of other security mechanisms incompatible with solution 1, such as secret sharing schemes. Spreading data and computation across several domains so that a single domain cannot compromise the privacy of the data held by it is one of the core concepts of SafeCloud that will be investigated in this solution. As

such, solution 2 is based on the assumption that the trusted deployment continues to have scarce storage resources, while still having computational capabilities. The untrusted deployments continue to complement the trusted domain with the necessary storage capabilities and additional computational capabilities. In summary, this solution aims at providing similar processing capabilities for the trusted and untrusted deployments while reducing the information leakage that a single untrusted infrastructure could exploit.

The untrusted deployment will be composed of more than one NoSQL database, while each database will be deployed in different cloud providers. Moreover, each database will only contain a part of the original information, which will avoid disclosing private information to the untrusted provider. In particular, the information is partitioned in a way that prevents a single provider to retrieve any meaningful information from the stored data. Additionally, special secure multiparty computation protocols can run over this data, enabling for NoSQL operations to be performed in a secure fashion at the untrusted deployments. This is key to support the desired functionality of delegating computation to the untrusted domain.

The architecture for this solution presents several challenges, both in terms of distributed systems and cryptography. Nonetheless, similarly to solution 1, the objective is to offer both a SQL and NoSQL interface to the SafeCloud client applications. To achieve such goal, the untrusted deployment will offer a virtual NoSQL interface on top of three HBase databases. Applications using this interface will be abstracted from the underlying data partitioning and from the number of HBase replicas being used, thus seeing a regular single node Hbase API. On top of the NoSQL API, and by recurring to the trusted deployment processing capabilities, this solution will be able to offer a fully ANSI SQL compliant database system with transactional support. Following the same structure of the previous section, we now describe in more detail the design and architecture of solution 2.

4.2 Architecture and API

Figure 6 depicts the architecture for solution 2. The major architectural difference from solution 1 is that now there are several HBase instances hosted in different untrusted infrastructures, while the trusted proxy is now responsible for collecting and processing responses from all these instances.

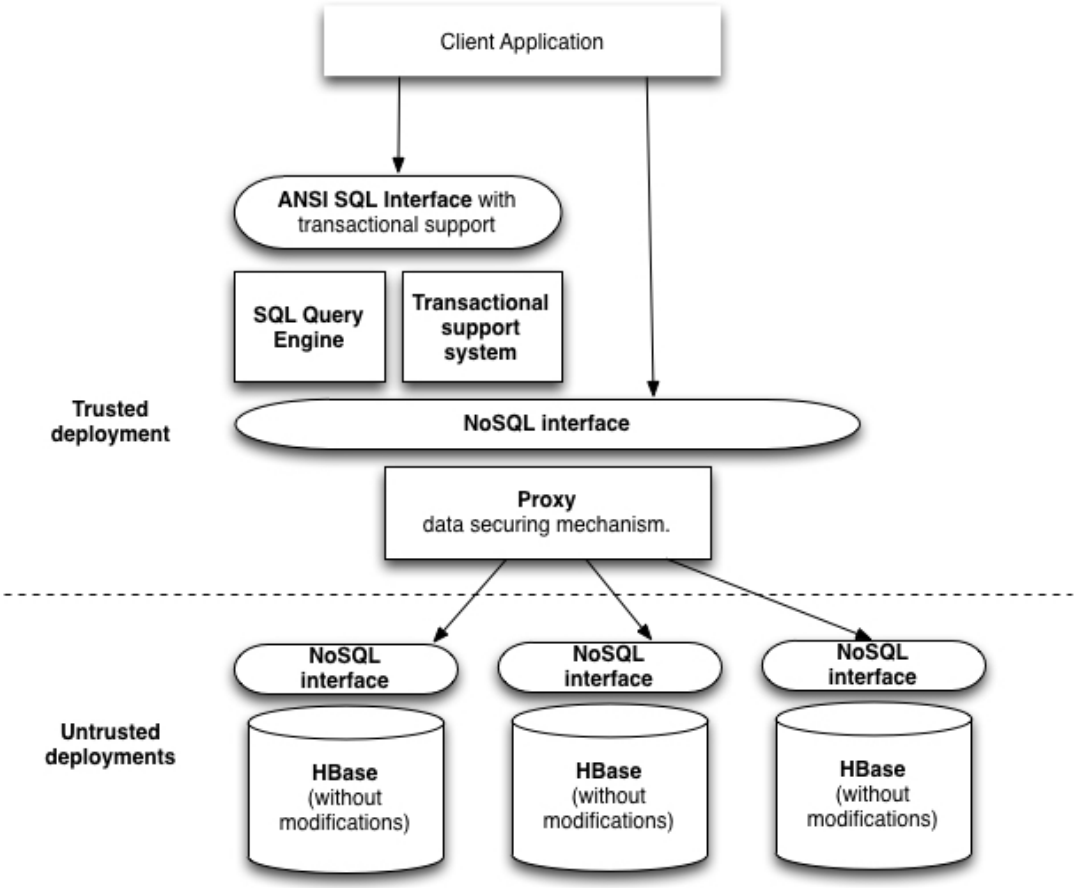

Figure 6: Solution 2 architecture

4.2.1 Trusted deployment

The trusted deployment remains very similar to the one described in solution 1, with the SafeCloud SQL engine processing full ANSI SQL with transactional support and translating SQL queries into NoSQL statements. The greatest change in this layer is in the trusted proxy component. The proxy mechanism is responsible for not only applying the required security mechanisms to the data but also to create a virtual and transparent NoSQL interface over the NoSQL database instances.

This architecture with several database instances responsible for performing secure computation expands the range of cryptographic techniques that can be applied. One way to benefit from this would be to use secret sharing and secure multi-party computation protocols over the stored data. Loosely explaining, secret sharing is a method for distributing a secret among several participants: each one will receive a share of this secret, each share does not reveal any information by itself, but a sufficient number of shares allow for the secret to be reconstructed [DPS+08]. In this specific implementation, the proxy is responsible for generating and distributing the shares associated with sensitive data, as well as gathering and interpreting the shares it receives from the untrusted domains.

From the client perspective, we expect the usability experience (modulo the performance toll these measures may take on the system) to be the same as if it was

interacting with a regular SQL/NoSQL database, depending on the API provided in the client interface.

4.2.2 Untrusted deployment

One of the SafeCloud objectives is to explore the possibilities of running SQL or NoSQL queries on data that is divided in several databases. Accordingly, in the solution presented in this section, the untrusted deployment now encompasses several NoSQL databases, in this case HBase stores, running in different clouds. Again, we cannot trust any of the cloud providers, but we assume they are not willing to collude. Each HBase instance will be able to perform computation over stored data without disclosing any meaningful information about that data. For instance, if secret sharing is being used, the HBase instances will be able to translate NoSQL queries into protocols that are aware of the underlying secret sharing schemes. This will allow each instance to process data and answer the queries while still being unable to, individually, extract any knowledge from it.

Some computation is already possible in the distributed secret shared scenario when using additive secret sharing [BNT+12]. This secret sharing scheme already has several secure protocols that can compare values and do arithmetic operations on shares. To achieve this, the databases go through several steps of information sharing, which allow them to reach consensus about the stored data without ever knowing which are the original secrets. In SafeCloud the aim is to extend these techniques to HBase databases to allow secure data processing in such systems.

It is of special relevance that these mechanisms can be included in the HBase system without modifying its core implementation. This can be achieved by using HBase co-processors that, for simplicity, can be seen as plugins that are implemented and added to HBase instances, and that allow extending the computation done when NoSQL queries are performed [APACHE12].

4.3 Discussion

In summary, the solution presented in this section no longer assumes that data is stored in a single untrusted entity. This assumption allows increasing the provided security guarantees and to employ new cryptographic techniques like secret sharing mechanisms. In particular, these multi-party computation algorithms over sensitive data don't require for significant properties of the data to be revealed, as was the case with order-preserving encryption.

On the other hand, secret sharing techniques will add extra storage and communication overhead that will negatively impact the performance of solution 2, at least when compared with the performance achievable by solution 1.

Returning to the clinical laboratory example, this time the laboratory owns a private cluster with limited computational power, and wants to offload storage and computational power to several cloud providers. Much like in the previous scenario, the laboratory is not willing to disclose patient sensitive information, but now the order of the sensitive information follows a predictable pattern, so the laboratory is not interested in order-preserving security mechanisms. Moreover, the laboratory does not trust a single cloud provider to hold and process all the data. However, it considers the

scenario of colluding competing cloud providers highly unlikely, so it is comfortable with assuming it will not occur.

By spreading the information in parts according to provably secure mechanisms of secret sharing, it has assurances that each cloud provider will not be able to retrieve the sensitive information from what is stored within. Furthermore, the homomorphic properties of these shares allow for computation over them, so the laboratory also has access to provably secure protocols for processing over the securely stored shares, in a way that the secrecy of data is kept at all times.

For a broader scope of real-world applications, we might be interested in allowing for several untrusted clients to query over potentially sensitive data. In solutions 1 and 2 there is a single trusted domain that manages data in the clear, decides what encryptions schemes to apply and that handles the translation of SQL to NoSQL. This does not consider the scenario in which several laboratories (with independent databases holding sensitive data) want to jointly compute over their data sets. This is a very common use case when considering the relevance of subjects such as big data, where these laboratories could ascertain valuable statistical information from huge quantities of data. Such use case is explored in solution 3 presented next.

5 Solution 3: Secure processing in multiple untrusted domains with untrusted clients

5.1 Introduction

In the first two solutions it is assumed that there is only one entity, who owns all the data stored in the untrusted domain. In this third solution this assumption is dropped. The motivation for getting rid of this assumption comes from the need to analyze data from multiple sources while keeping it private. A well known example, where there are many data owners and shared analytics is needed, is the Yao's Millionaires' problem [YAO+82]. In this simplified case there are two millionaires, who want to know who is richer. The problem is that they do not want to reveal their net worth to anyone. In this simplified case, we have two data owners, the millionaires and the need for analytics on the shared data of both: who is richer. One extra requirement is that the system should give the answer to only authorized users, in this case the millionaires themselves. A third party, for example the local newspaper, should not learn which of the millionaires is richer.

This means that even the authorized users should not be able to see all the data. Therefore, data is processed in the untrusted domain in encrypted form on the backend components. Users cannot make direct queries for data: only aggregation results are returned. It might be the case that for real world applications more fine-grained access policies are needed. For an example computing minimum of some table column would reveal too much and is not allowed, but computing mean of the same column is allowed.

Millionaires' problem is a classical toy example, but there are many more cases, where different entities, who own sensitive data, could leverage the results of joint data analysis. Examples can be found where personal data is stored. For an example a

hospital has medical records, which are highly sensitive. However, to track large scale viral outbreaks, different hospitals need some way to combine their data, while preserving their confidentiality.

5.2 Architecture

The architecture for Solution 3 is depicted in Figure 7. In previous solutions the trusted domain processing did some actual processing of the data. In this solution the trusted deployment is responsible for only encrypting the query parameters, decrypting the results and orchestrating the work done in the untrusted domain. The front-end still deals with processing the query and executing it on the backend, but unlike the other solutions, only the queried data is sent to the trusted domain. The only data processing that happens in the trusted domain is encryption/decryption of the data.

In the sense of the general architecture in Section 2, there are no trusted domain processing or storage involved. The proxy component does encryption, decryption and query translation in addition to communicating with the untrusted domain components. The solution will be built on top of the Sharemind framework [SHAREMIND+08]. The Sharemind framework is a programmable system for doing privacy-preserving computations. Although Sharemind supports different deployment schemes, the most evolved protocol suite is based on secret-sharing between three parties. Therefore, the untrusted deployment consists of three Sharemind servers, each holding part of the secret-shared data.

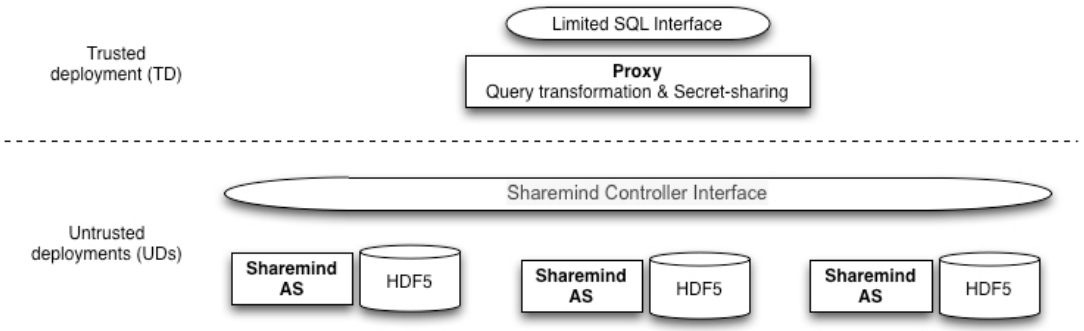

Figure 7: Solution 3 architecture.

It is required that three Sharemind Application Servers are hosted in different administrative domains. Three domains just need to be different, they all can be in the untrusted cloud. This is to avoid collaboration between entities that host the Sharemind servers. Collaboration would break the confidentiality guarantees offered by secret sharing.

5.2.1 Proxy component

The proxy component provides a SQL interface to the secure processing system. It parses the queries and translates them to a series of operations that need to be done on the back-end. Some of these operations, like filtering, require input parameters. The front-end component will encrypt these parameters when executing the operations. The results of the operations are stored in temporary tables, which stay in the back-end. The proxy component only gets references to these tables. A special operation is used to

return the final result of a query back to the proxy component. The proxy component will then decrypt the result and outputs it to the SQL interface.

In Sharemind context the Proxy component is just a specific application that uses the Sharemind Controller library. The controller library has simple functions to run programs on Sharemind deployment. The programs are written in a domain specific language called SecreC. Programs written in SecreC have special data types for secret shared data, and operations run on the data will automatically use secure multiparty computation protocols.

5.2.2 Back-end Component

The back-end component is a suite of functions written in SecreC, which run on top of the Sharemind Application Server. The suite has functions for basic operations like filtering and grouping tables. All the operations are executed using secure multi-party computation protocols based on additive secret-sharing scheme.

Sharemind Application Server has a database component which stores encrypted data. It currently uses a simple HDF5 based backend as the storage component. HDF5 is a data model, hierarchical file format and a library used mostly for scientific experiments [HDF5]. However, other more interesting storage backends can also be implemented easily thanks to Sharemind Application Server's modular design.

5.3 Discussion

The solution will not implement the full ANSI SQL standard, because some parts of it would be inefficient or difficult to implement securely. For some operations like Equi-joins we have efficient implementations. However, the ANSI SQL standard has some obscure parts that do not fit into the secure processing model very well.

The performance of the solution will be highly dependent on the network latency between Sharemind servers. The latency and bandwidth between the proxy and Sharemind servers will not be as important factor, as is the the latency and bandwidth between Sharemind servers.

This solution covers an important use case, where something has to be computed on confidential data belonging to multiple parties, who cannot trust each other with the data. Sometimes the legal aspects prevent giving out data to other parties. Other times business secrets cannot be revealed to competitors. However useful statistical information, that all parties would benefit from, can be obtained from shared data. This solution makes it possible to obtain the benefits, while preserving the confidentiality of data.

6 Conclusion

This first deliverable of WP3 aims for two major goals. First, it presents and details the unifying general architecture of SafeCloud. Second, it proposes three main solutions for privacy-preserving storage and computation based on this architecture and focused on providing feasible deployments to meet the requirements of SafeCloud's use cases.

Our proposed architecture is heavily rooted in the cloud computing paradigm, therefore making the explicit distinction between *on premises infrastructure* and *cloud infrastructure*. This led to a general solution architecture comprised of two main components: the *trusted deployment* and the *untrusted deployment*, communicating with the client via a *client interface*, and to each other via a *proxy*. By having these clear distinctions, we can specify:

- The API our solutions provide to the client, being either SQL or NoSQL - *client interface*.
- The computational requirements delegated to the local and remote components - *trusted/untrusted deployment*.
- How the client side is to interact with the cloud provider - *proxy*.

Our privacy-preserving solutions follow the described structure, and are derived from a comprehensive analysis of security and performance trade-offs. SafeCloud avoids the problems inherent to a single-scenario focused approach by providing several solutions that meet different functional and security requirements. Our first solution assumes a single untrusted cloud provider, and proposes the usage of different techniques for various levels of data sensitivity. The second solution attempts to extend the functionalities of the first one by assuming a setting with multiple non-colluding untrusted cloud providers, employing security mechanisms that rely on this new deployment to provide higher levels of security for storage and processing. Our third and final solution extends the scope of the second by considering multiple clients with sensitive data, which allows for computations relying on huge amounts of data, such as data analytics, to be run over several datasets of data without revealing sensitive data to the party requesting the computation (other than what would be revealed by the computation result itself).

To sum up, this work package aims to go beyond the state of the art to provide a range of solutions that fit applications with different requirements in terms of security and efficiency, while allowing secure and private data processing in untrusted environments.

7 References

[MSL+11] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, and Markus Huber (2011). "Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space". USENIX Security Symposium.

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia (2010). A view of cloud computing. Commun. ACM 53.

[Dropbox12] Dropbox (2012). ”Dropbox has become "problem child" of cloud security …”,(http://venturebeat.com/2012/08/01/dropbox-has-become-problem-child-of-cloud-security/)

[CK08] Octavian Catrina, and Florian Kerschbaum (2008). "Fostering the uptake of secure multiparty computation in e-commerce”. Third International Conference on Availability, Reliability and Security.

[L13] Susan Landau(2013). "Making sense from Snowden: What's significant in the NSA surveillance revelations”. IEEE Security & Privacy.

[D+14] Zakir Durumeric et al (2014). "The matter of heartbleed”. Conference on Internet Measurement Conference.

[LCW93] Hongjun Lu, Hock Chuan Chan, and Kwok Kee Wei. 1993. “A survey on usage of SQL”. SIGMOD Rec. 22.

[HHL+11] Jing Han, E. Haihong, Guan Le, and Jian Du (2011). "Survey on NoSQL database”. Pervasive computing and applications.

[ANSI16] American National Standards Institute. ANSI Web page. (http://www.ansi.org)

[DD+93] Chris J. Date, and Hugh Darwen (1993). “A guide to the SQL Standard: a user's guide to the standard relational language SQL”. Vol. 55822. Addison-Wesley Longman.

[APACHE16] Apache HBase Team (2016). “Apache HBase ™ Reference Guide”. (https://hbase.apache.org/book.html)

[BCL+09] Alexandra Boldyreva, Nathan Chenette, Younho Lee and Adam O’Neill (2009). "Order-preserving symmetric encryption”. Advances in Cryptology-EUROCRYPT.

[DPS+08] Mahir Doganay, Thomas Pedersen, Yücel Saygın, Erkay Savaş, and Albert Levi (2008). "Distributed privacy preserving k-means clustering with additive secret sharing." International workshop on Privacy and anonymity in information society.

[APACHE12] Apache HBase Team (2012). “Coprocessor Introduction”. (https://blogs.apache.org/hbase/entry/coprocessor_introduction)

[CNIMBO09] CumuloNimbo european project partially funded by the European Commission under the 7th European Framework contract number FP7-257993 (2009). (http://www.cumulonimbo.eu/node/20)

[CPAAS14] CoherentPaaS european project funded by the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 611068 (2014). (http://coherentpaas.eu)

[YAO+82] Yao, Andrew C. "Protocols for secure computations." Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on. IEEE, 1982.

[BNT+12] Dan Bogdanov, Margus Niitsoo, Tomas Toft, Jan Willeson (2012). "High-performance secure multi-party computation for data mining application". International Journal of Information Security.

[SHAREMIND+08] Dan Bogdanov, Sven Laur, Jan Willemson. Sharemind: a framework for fast privacy-preserving computations**.** In Proceedings of 13th European Symposium on Research in Computer Security, ESORICS 2008, LNCS, vol. 5283, pp. 192-206. Springer, Heidelberg. 2008.

[HDF5] HDF5 home page. (https://www.hdfgroup.org/HDF5/), 2016