SafeCloud

Elastic privacy-preserving storage and computation

D3.6

Project reference no. 653884

August 2017

European Commission

Horizon 2020
European Union funding
for Research & Innovation

Document information

Scheduled delivery	31.08.2017
Actual delivery	31.08.2017
Version	1
Responsible Partner	INESC TEC

Dissemination level

Public

Revision history

Date	Editor	Status	Version	Changes
05.07.2017	F. Maia	Draft	0.1	ToC
19.07.2017	J. Paulo	Draft	0.2	INESC TEC input
28.07.2017	K. Tarbe	Draft	0.3	Cyber input
07.08.2017	H. Niedermayer	Review	0.4	TUM review
20.08.2017	J. Paulo	Review	0.5	TUM review included
16.08.2017	K. Tarbe	Review	0.6	CYBER review
29.08.2017	J.Paulo	Final	1.0	Final version

Contributors

Karl Tarbe (CYBER)
João Paulo (INESC TEC)
Francisco Maia (INESC TEC)

Internal reviewers

Heiko Niedermayer (TUM)

Acknowledgements

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

More information

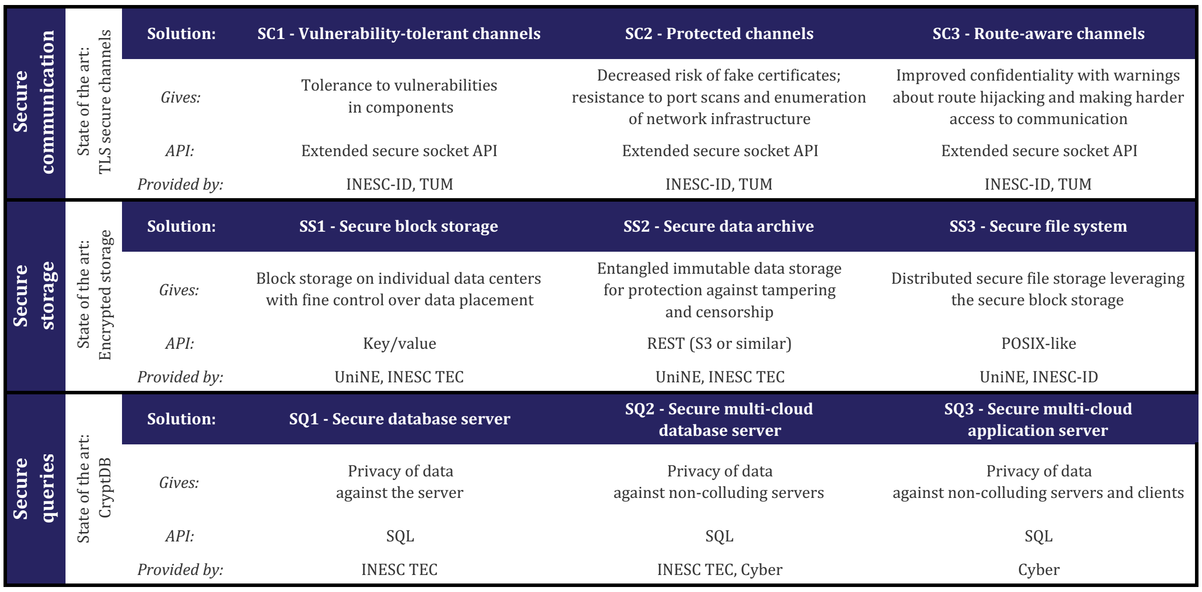
Additional information and public deliverables of SafeCloud can be found at http://www.safecloud-project.eu

Table of contents

List of Figures

Executive summary

The framework proposed by SafeCloud consists of three layers: secure communication, secure storage, and secure queries. Secure communication provides schemes for the establishment of channels amongst protocol participants employing technologies for tamper-resistant channels, ensuring confidentiality and availability. Secure storage provides techniques for reliable storage, such as long-term confidentiality, protection against file corruption or data deletion. Finally, secure queries provide cryptographic constructions from the database storage layer to the end-user processing requests. The overarching idea is to allow system developers to use the techniques provided by these three layers in order to achieve application-specific deployments. These deployments should surpass the state-of-the-art of existing tools with respect to functionality, performance and security. We recall Figure 1, from the general SafeCloud framework description.



Secure communication	State of the art: TLS secure channels	**Solution:**	**SC1 - Vulnerability-tolerant channels**	**SC2 - Protected channels**	**SC3 - 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:**	**SS1 - Secure block storage**	**SS2 - Secure data archive**	**SS3 - Secure file system**
		Gives:	Block storage on individual data centers with fine control over data placement	Entangled immutable data storage for protection against tampering and censorship	Distributed secure file storage leveraging the secure block storage
		API:	Key/value	REST (S3 or similar)	POSIX-like
		Provided by:	UniNE, INESC TEC	UniNE, INESC TEC	UniNE, INESC-ID
Secure queries	State of the art: CryptDB	**Solution:**	**SQ1 - Secure database server**	**SQ2 - Secure multi-cloud database server**	**SQ3 - Secure multi-cloud application server**
		Gives:	Privacy of data against the server	Privacy of data against non-colluding servers	Privacy of data against non-colluding servers and clients
		API:	SQL	SQL	SQL
		Provided by:	INESC TEC	INESC TEC, Cyber	Cyber

Figure 1: The SafeCloud framework.

The Secure queries layer contemplates three different solutions, each assuming different trust models, deployment models, privacy-preserving techniques, and consequently, different tradeoffs in terms of performance, dependability and security.
This deliverable is concerned with the elasticity of the three distinct solutions. In other words, the ability to scale-up or down the resources used by each of the solutions, according to the performance and availability requirements of the applications using these, is a crucial property that is nowadays present in state-of-the art commercial cloud services.
The document presents how elasticity is achieved, for each of the three solutions, by recapping the main components present in each architecture and specifying in a per-component basis the needed steps for achieving an elastic and fault-tolerant deployment. This discussion takes into account the impact in fault-tolerance and elasticity for the different privacy-preserving techniques that each solution supports and how the overall architectures are able to scale-up or down according to demand.

1 Introduction

The SafeCloud secure queries architecture has three solutions as shown in Figure 2:

Secure queries		
SQ1 **Secure database server**	**SQ2** **Secure multi-cloud database server**	**SQ3** **Secure multi-cloud application server**

Figure 2: Secure Queries Solutions.

For each processing solution, a detailed architecture, the prototypes and the different sets of supported privacy-preserving techniques have been described in previous deliverables D3.1 to D3.5. The different architectures and privacy-preserving techniques used in each solution assume different trust models and, as a consequence, different privacy guarantees emerge for each solution.

All processing solutions are designed to be deployable in distributed infrastructures and to support thousands to millions of clients. For this, each design must be able to scale up in terms of resources (servers, disks, etc) as the number of clients increases. This property is crucial to maintain the desired quality of service (performance, dependability).

Another important property to be contemplated in the SafeCloud processing stack is the ability to decrease the computational resources as the load in the system decreases. For most applications, there are off-peak periods where the load in the system is expected to be smaller. Reducing the computational resources being used while maintaining the desired quality of service allows a more cost-efficient approach.

This ability to scale-up and down computational resources according to the load of applications is a fundamental characteristic of cloud services that will be contemplated in the SafeCloud project[AFG+10]. In this deliverable, we provide a thorough description on how elasticity is provided for each SafeCloud processing solution. We provide an independent analysis for each architecture and design, both in terms of how the different components handle elasticity and on the impact of the supported privacy-preserving techniques.

In the following sections, we start by presenting an overview of how elasticity is handled in state-of-the-art SQL and NoSQL databases and then, for each of the three SafeCloud processing solutions, the following technical details are discussed. First, we recap the architecture of the solution, which has already been detailed in previous deliverables, to present the main components that will need to handle elasticity. Then, for each component, we explain how data partitioning and resource allocation can be done to support a fully elastic solution. This analysis also contemplates the impact in elasticity for the privacy-preserving techniques supported by that specific solution.

2 Context

Cloud services are now widely adopted due to their pay-as-you-go model which allows enterprises to only allocate and pay for the resources needed by their applications as the number of clients increases or decreases [AFG+10, GB12]. For achieving an efficient pay-as-you-go model, the elasticity of cloud services is fundamental. By elasticity, we mean the ability to grow and shrink resources according to the needs of the applications using these distributed services. In a cloud scenario, allocating more resources for an application requires launching more cloud server instances and balance the application load between both the new servers and the previous ones. On the one hand, when applications are using more resources than needed, with respect to the expected quality-of-service, some resources can be released. For this case, the workload being served by the allocated cloud servers must now be migrated to the remaining servers seamlessly.

On the other hand, the removal of servers from the cloud service may also happen due to failures. In this case, if a specific server fails, data belonging to the applications must not be lost and the application workload must still be served with the desired quality-of-service guarantees. Then, if needed, additional servers can be launched to replace the failed ones.

Current SQL solutions provide cluster-based approaches for dealing with the addition and removal of servers [WPS+00,MYSQL17]. Servers in the same cluster are replicated either with synchronous or asynchronous replication to ensure fault-tolerance, while sharding can be applied to partition the database tables and improve the scalability of SQL deployments. However, the sharding process in traditional SQL databases is a complex task as the relational model is heavily based on establishing relations across tables, which makes it harder to split tables in different servers while still guaranteeing the desired Availability, Consistency, Isolation and Durability (ACID) requirements. Similarly, database replication requires complex and costly synchronization mechanisms. Thus, impairing the cluster ability to scale up from setups with tens to hundreds of servers.

On the other hand, NoSQL databases provide a simpler schema and API that allows improving the ability to scale up and down by relaxing the strong data consistency provided in SQL solutions [HHL+11]. For these NoSQL solutions, tables can easily be sharded and replicated across different servers, making the scale up and scale down operations easier to implement and maintain.

The HBase system, which is the NoSQL database used in SafeCloud secure queries solutions 1 and 2, provides a good example on how elasticity is handled in NoSQL solutions. Next, we detail how this system handles the partitioning of data and the addition and removal of nodes. This description is also important to understand how SafeCloud secure queries solutions 1 and 2 will provide these desired guarantees.

2.1 HBase

Apache HBase is a distributed, scalable and open-source non-relational database [APACHE17a]. Inspired by Google's BigTable [CDG+08], it can be thought of as a multi-dimensional sorted map or table. The map is indexed by a tuple composed by row key, column name and timestamp, which is used as a key for a given value, as illustrated in Figure 3.

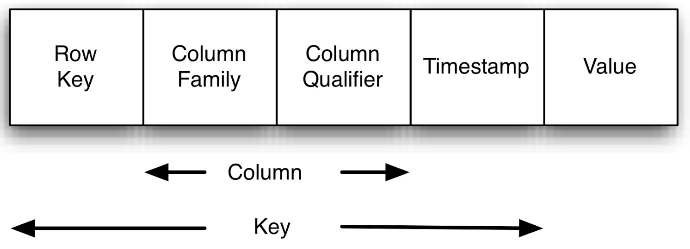


Figure 3: HBase map structure.

Row keys might be associated with an unbounded and dynamic number of qualifiers (columns) grouped into column families (groups of columns). As an example, in a HBase table storing information of a given company's employees, "employee" may be a column family and the employee's "name", "age", "salary" can be distinct column qualifiers grouped by that column family. Each qualifier is then identified by concatenating its column family's name and qualifier byte array, *i.e.*, family:qualifier. A number of rows form a table, and each row may specify a distinct number of column families. Both the row key and the associated values are arbitrary not-interpreted arrays of bytes. Data is maintained in a lexicographic order first by row key, second by column's family name followed by qualifier and, in descendent order, by timestamp.

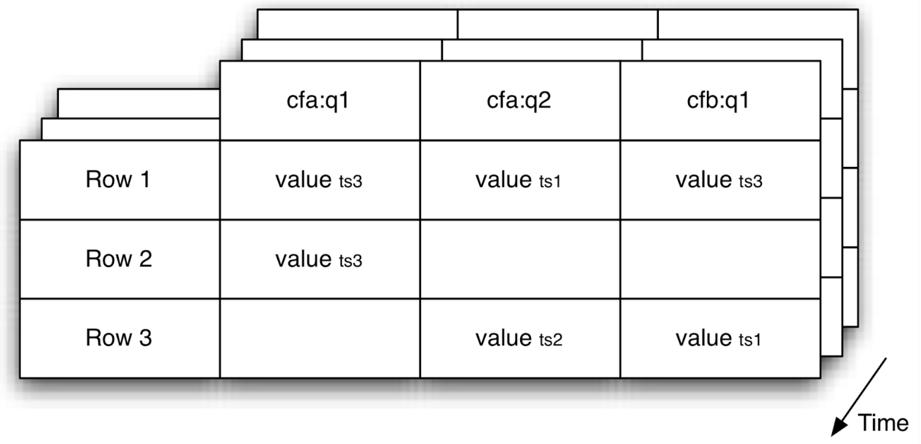


Figure 4: HBase logical table view.

A logical view of an HBase table is presented in Figure 4. In this view, each cell (value) is the intersection of a row key and a column qualifier (cq). Additionally, timestamps (ts) may be used to have a multi-dimensional table, since it means that several versions may exist simultaneously. From this point onwards, we will use the term "row" to denote a single row according to this logical view. Typically, column families (cf) are well-defined and must be created before data can be stored. In contrast, qualifiers are created in runtime by inserting new key-value pairs.

2.1.1 HBase API

HBase exposes a set of operations for data access that is quite similar to the one used in other key-value data stores. This interface is provided by the HBase client component and encompasses the following operations:

- GET - Get key-value pairs of a given row, identified by row key;
- PUT - Insert or update a key-value pair for an existing or a new row;
- SCAN - Get all key-value pairs for a specific range of rows;
- DELETE - Remove one or more key-value pairs belonging to one or more rows;

Note that, for GET and PUT requests, it is only possible to retrieve or update specific column qualifiers if the requests specify both the row key, column family and column qualifier being targeted. Additionally, HBase provides filter operations for both GET and SCAN requests. Namely, it is possible to request several key-value pairs with a scan request and then filter only the key-values where a specific column has a certain value. As an example, if a company stores in HBase information about its employees, it is possible to query all employees with identification numbers (row key) between number 100 and 1000, and then filter the request to only retrieve the entries for employees that were born in 1986 (considering that age is a column qualifier).

2.1.2 HBase Architecture

Figure 5 depicts the HBase architecture and main components.

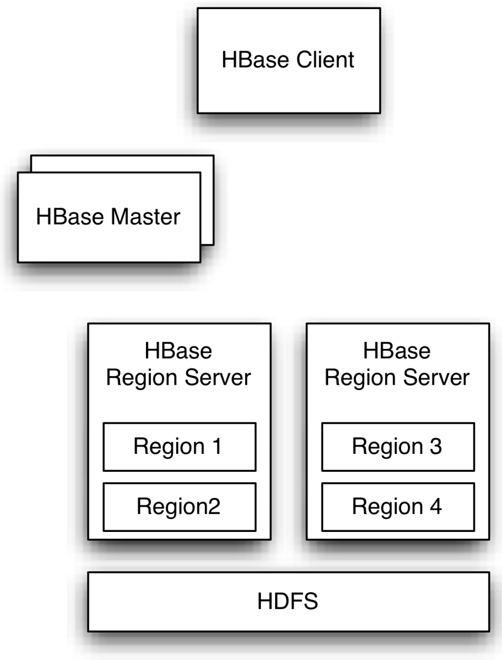


Figure 5: HBase Architecture.

An **HBase client** component is provided, so that applications using HBase on the client-side are able to perform queries, following the HBase API, to the HBase backend (combination of HBase Masters and Region Servers). Succinctly explained, the HBase client contacts the Master component to know what Region Servers are responsible for storing the rows for a specific request, and then the client issues the request to the appropriate Region Servers, which, in turn, reply to the client with the query results.

The **HBase Master** is responsible for redirecting HBase client requests to the appropriate Region Servers, where the keys being stored/retrieved are kept. The HBase Master may be deployed in a primary/secondary replication mode to ensure a fault-tolerant design via redundancy.
Rows in a table are partitioned horizontally and each partition is called a Region. After partition, resulting regions can be distributed across several nodes named **RegionServers** that are responsible for serving one or more regions. All columns and values of a given key (row) are available in the same Region Server. Regions Servers store and retrieve Regions' data from the Hadoop Distributed File System (HDFS) [APACHE17b]. This is a distributed file system deployable on top of multiple servers in order to provide a fault-tolerant design. Ideally, each Region Server is hosted in the same machine with the HDFS data node serving the data for the Regions belonging to that Region Server. This option promotes data locality and allows Regions Servers to have a more efficient access to their data.

2.1.3 HBase Elasticity

The horizontal partition of rows in HBase, each partition being an independent Region, allows to easily migrate a region across different RegionServers (usually located in different servers). This way, the HBase system can easily scale up by adding new RegionServers and moving regions of existing RegionServers to the new machines.
Furthermore, if the size of a Region (i.e, number of rows) grows above a given threshold, the Region can be split in two, while one of the Regions can be assigned to other RegionServer to improve load balancing and improve the database performance. This is an automatic mechanism supported in current versions of HBase.

In the case of server failures or intentional node removal, to reduce the number of resources (servers) for a specific HBase deployment, the regions being served by the removed servers will be migrated to other available RegionServers. This assignment is done by the HBase Master component, which can be replicated for fault-tolerance purposes. In fact, the HBase Master component is aware of this migration of Regions and redirects the client requests for the keys in those regions to the new RegionServer(s).

Note that the actual data being served by the HBase cluster is stored in HDFS, a distributed and fault-tolerant file system. By choosing the appropriate degree of data replication, one ensures that when servers fail, the data of all regions is still available and persisted. However, when Regions are migrated due to scale up or down concerns, the locality of data between the Data Node and Region Server running in the same server is lost. For ensuring that data locality is restored, HBase supports a compaction operation that allows rewriting the content of a Region to HDFS in order to ensure that the DataNode co-located with the RegionServer responsible for that Region is the one holding a copy of the data. This is a costly operation that may only run periodically due to its noticeable overhead in the performance of requests to the HBase cluster.

2.1.4 HBase and SafeCloud's Secure Queries Solutions

Solution 1 and 2 depend on a HBase NoSQL backend and will use the elasticity mechanisms already provided by HBase to support the elasticity and system dynamism (entrance and leaving of nodes) required by these components. As detailed in Section 4, Solution 2 resorts to multi-party protocols, depending on three different HBase clusters, which requires additional mechanisms to allow the desired elasticity and dynamism properties.

The other main component present in the architecture of these two solutions is the SQL query processing. As we will show in the next sections, this is a stateless component that can easily be launched or removed from servers without affecting the proper behaviour of the corresponding secure SQL stack. Solution 3 does not use HBase and therefore cannot use the elasticity and dynamism provided by HBase.

Next, we describe in more detail, for each solution, how elasticity is achieved and the impact that each architecture and the chosen privacy-preserving techniques have in achieving this goal.

3 Secure Database Server

3.1 Architecture and Workflow

Any application that wants to integrate SafeCloud secure queries solutions has two distinct APIs available. It can use either a SQL interface or a NoSQL one. For solution 1 in particular (secure database server), SafeCloud will provide full SQL compatibility, for the processing use-cases of Work Package 5, and a HBase-like NoSQL interface.

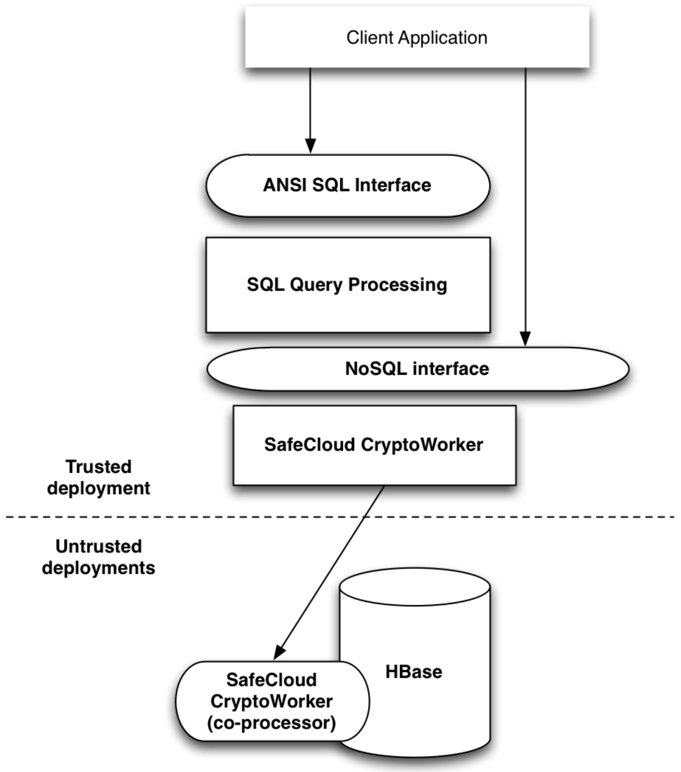


Figure 6: Concrete SafeCloud deployment for Solution 1.

To offer a SQL and NoSQL integration for the client application, SafeCloud solutions are deployed across two main sites (one trusted site and one untrusted) as explained in previous deliverables. Figure 6 depicts a high-level overview of such deployment scheme.

Concretely, the client application has access to the trusted deployment site where it can issue requests to the desired API - SQL or NoSQL. For the case the application is using the SQL interface, a query processing component translates the application queries into NoSQL ones.

In SafeCloud Secure Queries solutions 1 and 2, transactional support can be achieved by integrating the existing open-source project OMID [APACHE17c, BHK+17] with HBase and the query processing component. In fact, the OMID system is already tightly coupled with HBase, extending it with support for transactions. This means that the query processing component must translate SQL requests, with transactional semantic, to the appropriate transactional NoSQL requests. The first requirement (transactional SQL translation) is already met by the current version of the query processing component and the second requirement (transactional NoSQL support) will be supported by integrating OMID with solution 1.

After the translation step, upon the reception of a NoSQL request a SafeCloud CryptoWorker will be called to handle it [MPP+17]. These CryptoWorker components are installed in the same trusted site as the query processing component. The main job of the CryptoWorker is, according to its configuration, translate the received NoSQL operations into privacy-preserving requests that will be processed on a single untrusted site, where an HBase cluster is deployed. As some cryptographic techniques require both computation at the trusted and untrusted site (e.g. searchable encryption schemes) an untrusted site CryptoWorker will translate messages sent by the CryptoWorker at the trusted site into NoSQL operations that the HBase backend can process. In particular, in order to improve the performance of the system and to leverage specific characteristics of HBase, SafeCloud CryptoWorkers are deployed at the untrusted site as HBase co-processors. Thus, they are running collocated with HBase Region Servers [APACHE12]. For other techniques, such as Standard, Deterministic and Order-preserving encryption, computation is only required at the trusted site CryptoWorker. These techniques preserve the computational requirements of the data being protected which allows the HBase backend to perform the needed computation over private data.

More details regarding this architecture refinement including cryptographic components for NoSQL translation, and how these are instantiated in the current prototype are available in deliverables D3.3 and D3.5.

3.2 Data Partitioning

In order to have an elastic design our solution must support efficient data partitioning. The first important aspect of this design is that data is only being persisted at the untrusted NoSQL (HBase) backend. Both the query processing and trusted CryptoWorker components are stateless (with the exception of some configuration files that can easily be replicated across different servers running these components). This means that data partitioning is only needed at the HBase database. For this solution, the supported privacy-preserving techniques (further discussed in D3.5) do not affect the partitioning scheme used in the original HBase system. This way, data partitioning (i.e., Region splitting and merging) is managed by the HBase component without requiring any modification.

3.3 Elasticity

The same way data partitioning is managed by HBase without any modifications, providing elasticity in this solution also solely depends on the elasticity capabilities of the HBase deployment itself. In more detail, since the entire trusted deployment is composed of stateless servers (Distributed Query Engine, Transactional Components and HBase client components), elasticity at this level is easily handled without any need for online external coordination software, data migration or system reconfiguration. Different instances of these components can be launched or removed without any impact in the correct functioning of Solution 1. However, three bootstrapping mechanisms must be taken into account. First, static configuration files needed by these components must be made available across the machines where these are deployed. For this, either the configurations are replicated in each server or stored in a service such as Zookeeper [APACHE17d]. Second, client applications must have a discovery service available in order to locate Query processing nodes available to process requests. This can easily be achieved with a load balancing component. Third, different CryptoWorkers must have access to a cryptographic key repository where encryption keys for protecting data are available. This requirement is important so that different CryptoWorkers are able to encode and decode common data. This key repository may be made available by the applications using the database as, in many cases, the management of cryptographic keys is application-specific. The SafeCloud platform's modular design allows the integration with several key management services.

The database's dataset is entirely stored on the untrusted site. This requires additional care in order to provide elasticity due to the need to ensure data persistency and availability. However, as mentioned before, we can leverage elasticity mechanisms that HBase already provides. In fact, adding or removing nodes in an HBase system not only is possible but it also guarantees that data is automatically partitioned and migrated to accommodate such change. This automation is also available for fault tolerance. Whenever a server fails, HBase ensures the necessary migration of data Regions in order to continue serving requests. Adding nodes to replace the failed ones must be done manually but automatic external systems exist (e.g. MeT) [CMM+13]. These mechanisms of HBase resort to HDFS data replication thus making the adequate configuration of this component important.

Finally, the OMID transactional component resorts to the HBase cluster to store metadata, that does not have any sensitive information, which allows this system to scale up and down by leveraging the original HBase capabilities. The OMID component issuing the certification of transactions is the only centralized component but, as shown in previous work, it is able to scale for a large number of transactions and, in this project, to provide the needed transactional throughput[BHK+17].

3.4 Discussion

As mentioned previously, Solution 1 relies on the elasticity of the different components to provide a fully elastic solution. The privacy-preserving techniques do not have any impact in the elasticity of any of these components, which is not true for Solution 2, and again shows the advantage of providing different solutions with different tradeoffs in terms of performance, security and now, elasticity.

While the query processing and CryptoWorker components are stateless and can easily scale up or down, the HBase database already has in-place mechanisms to be scalable. Furthermore, much research work has been done lately to improve the elasticity of HBase and allow it to scale-up and down automatically [CMM+13]. Such work could be incorporated in this solution as future work.

Finally, it is important to note that dividing the architecture into all these components is important to have a design where it is possible to launch, independently, more instances of the components that are a bottleneck for the performance of client applications. Such is not possible in monolithic database designs where all components are tightly coupled and cannot scale-up or down independently.

4 Secure Multi-Cloud Database Server

4.1 Architecture

Similar to Solution 1, this solution also relies on a secure NoSQL backend (HBase). However, as pointed in Figure 7, Solution 2 considers multiple untrusted domains. Consequently, a single HBase deployment is no longer sufficient, as each untrusted domain must be treated as a completely independent entity. Nevertheless, the high-level approach taken is similar to the one previously described for Solution 1, i.e., it has a SQL query processing component with transactional support, which can be provided through the integration with OMID, and we do not modify HBase itself but rather externally add the necessary behavior via the appropriate embedded mechanisms (coprocessors).

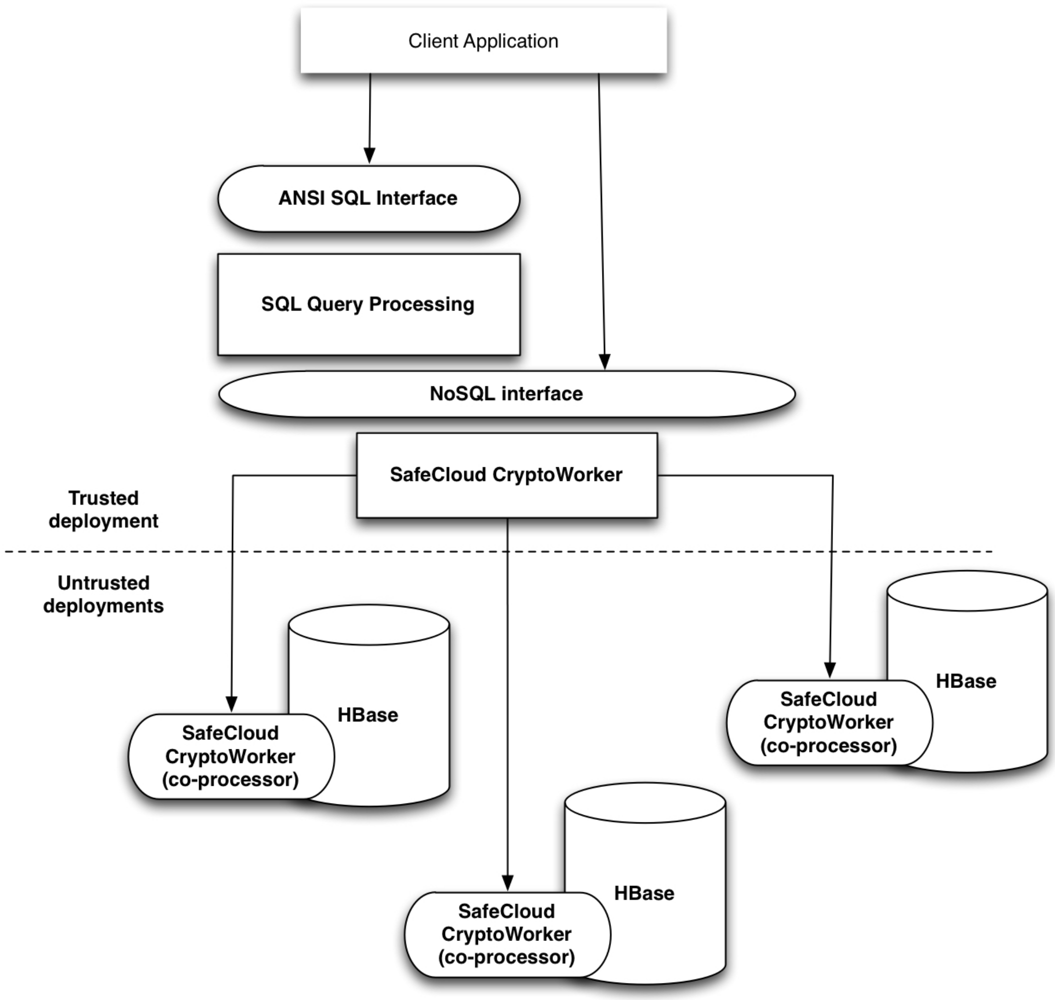


Figure 7: Concrete SafeCloud deployment for Solution 2.

For the particular case of Solution 2, we consider three independent untrusted sites (infrastructures), each with a separate HBase deployment (cluster). The added behaviour empowers each HBase cluster to provide secure operations following multi-party computation algorithms [PMP+16]. Moreover, this entails the requirement for communication capabilities between the different entities since this is a fundamental necessity for general multi-party protocols.

A client-side CryptoWorker, at the trusted site, is instantiated and configured with the appropriate multi-party computation parameters. In this solution, untrusted site CryptoWorkers (one per HBase RegionServer) take part in the process, and are now responsible for request processing and engage in a specific workflow to be able to address the data processing requests from the NoSQL client. In particular, they follow multi-party computation protocols that require communication between the three CryptoWorkers deployed in the three different untrusted sites.

Briefly, a PUT operation requires dividing the data to be stored, at the trusted-site CryptoWorker, into three secrets. Each secret is stored on a different HBase cluster and, independently it does not reveal any sensitive information from the original data. Then a GET operation for getting a specific value is encoded into three different operations to be issued to the three independent backends. When a backend-side CryptoWorker

identifies such request as a multi-party computation request, it starts the specific set of procedures to process it. This typically includes querying the local HBase instance for data, communicating with other backend-side CryptoWorkers and, finally, replying to the trusted site. The client-side CryptoWorker processes the answers and translates them into a plain data reply to the client application.
The computation and communication steps necessary for this workflow are thorough and subtle, since they depend on the underlying complex cryptographic mechanisms that allow for general secure computation of functionalities over several participants. For a more in-depth overview of MPC protocol details, please refer to deliverable D3.5.

4.2 Data Partitioning

As in Solution 1, the query processing and trusted-site CryptoWorkers are stateless so, data partitioning is handled at the untrusted HBase sites. Internally, each HBase cluster is able to partition data into different Regions that are then assigned to specific Region Servers. However, this data partitioning is done independently and in a non-mirrored fashion across the different HBase clusters (parties).

In order to run the multi-party protocols for a secret (row) spread in three different HBase clusters, it is necessary that the three RegionServers responsible for the region with that specific row are aware and can communicate with each other. As data partition in the different clusters is not deterministic, the information regarding the regions and the range of rows stored in the RegionServers of each HBase cluster must be made available in a discovery service. This way, when the untrusted site CryptoWorker (coprocessor of a given RegionServer) performs the needed multi-party computation protocol for its secret, it must resort to this discovery service to know which are the CryptoWorkers (coprocessors of other RegionServers in the other HBase clusters) that it must contact to exchange the results of the computation. In the context of SafeCloud, such a component was implemented.

Yet another challenge that arises with multi-party protocols is that operations are now sent to three different HBase clusters that may perform them in different orders. For a single client, requests can easily be ordered but when supporting multiple clients, this may be an issue. As an example, if a PUT request and a SCAN request reach the three clusters in different timings, the result of the computation may not be coherent as some HBase clusters have access to secrets that are still not stored in the other clusters.

We solve this challenge by leveraging the transactional properties provided by OMID. As a consequence, and in contrast with solution 1, any access to the NoSQL API in solution 2 must be transactional. Additionally, the HBase client was modified in order to consider an operation completed only when completed in the three sites.

4.3 Elasticity

As in Solution 1, the query processing and CriptoWorkers are stateless and can easily be deployed to new servers or removed from existing servers. Again, these components should ideally be collocated in the same server and the clients of the SQL solution must be aware of the available query processing endpoints. The OMID component is deployed in a similar fashion to Solution 1. The main difference, in the trusted site, between this and the previous solution is that OMID's metadata can be written to any of the HBase clusters as it does not disclose any sensitive information. With this integration, it is possible to have an elastic version of the components deployed at the trusted site similarly to what was previously described for Solution 1.

For the untrusted site, each HBase cluster handles elasticity independently while the discovery service mentioned in the previous section is used to enable the communication between untrusted CryptoWorkers (HBase coprocessors). This discovery system can be deployed entirely in-memory to achieve high-performance since it stores a small amount of information (in the order of MBs). Also, failures of this component can be tolerated since stored information is not critical and can be easily recovered from HBase's Master node.

Finally, it is important to note that the number of parties, in this case 3, is a restriction of the multi-party protocol itself. This means that in our current design it is not possible to increase or decrease the number of HBase clusters (parties).

4.4 Discussion

Multi-party protocols have intrinsic details that affect the elasticity of solution 2. Namely, a discovery service for enabling the communication between parties, and transactional or, at least, ordering support for NoSQL queries from distributed clients or components (trusted site CryptoWorkers) to ensure that concurrent write/update/delete and get/scan queries are executed consistently across the different parties and the results also consistently returned to the client. However, in the context of SafeCloud these challenges were tackled resorting to the implementation of specific software components or already existing software such as OMID. The resulting system is expected to scale graciously and achieve the performance levels required by the Work Package 5 use-cases. Furthermore, these components also contemplate a fault-tolerant design that does not compromise system's correctness when they fail.

Similar to what was described in solution 1, extensive research work has been focusing on improving the elasticity of HBase and allow it to scale-up and down automatically [CMM+16]. Such work could be incorporated in this solution, for each independent HBase cluster, as future work. As in Solution 1, dividing the architecture into all these components is important to have a design where it is possible to launch, independently, more instances of the components that are a bottleneck for the performance of client applications. Such is not possible in monolithic database designs where all components are tightly coupled and cannot scale-up or down independently.

5 Secure Multi-Cloud Application Server

5.1 Architecture

Secure Multi-Cloud Application Server provides a limited SQL interface, that protects different data owners from each other, while still allowing to have their data in a shared database, where analytic queries can be run.

It consists of a Sharemind deployment on untrusted cloud service providers and a number of SQL frontends, which are deployed in the trusted premises of the data owner or the analytic. SQL frontend handles plaintext data and it has to be on trusted infrastructure.

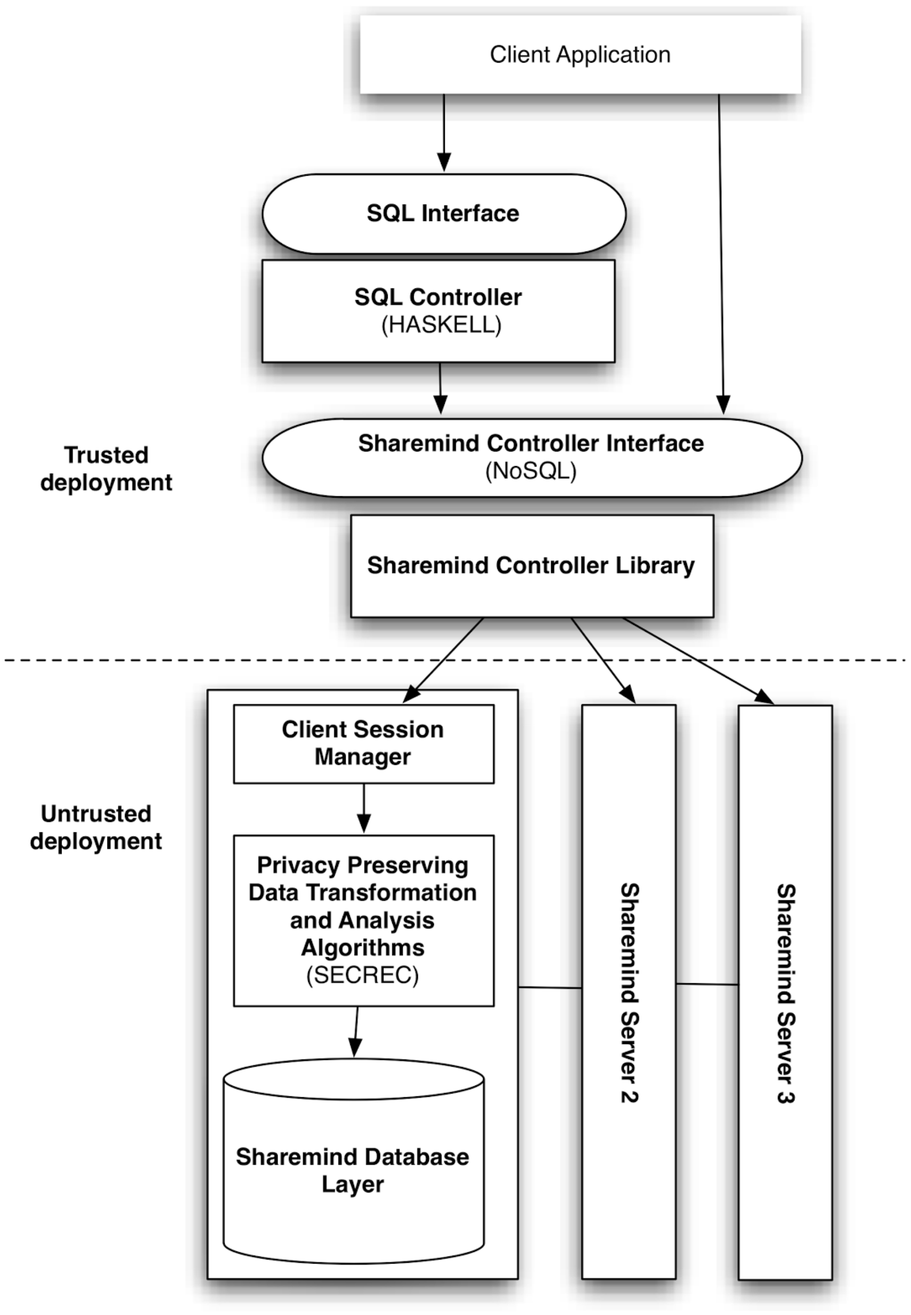


Figure 8: Secure Multi-Cloud Application Server architecture with deployment info.

As can be seen from Figure 8, Sharemind deployment consists of three Sharemind Application Servers hosted on non-colluding service providers. Sharemind Application Server is modular, and it can have several database backends for storing the secret-shared data. For an example, we could implement an HBase backend to leverage the elasticity of HBase.

On the other hand, as in Solution 2, the elasticity of the backend database (for storing secret shares) would not transfer directly into the elasticity of the whole solution.

5.2 Data Partitioning

Efficient data partitioning is needed for supporting elasticity. In our solution, the handling of partitioning is handed over to the distributed backend database, that stores the secret shares. Our SQL frontend and the rest of the Sharemind Application Server are mostly stateless. So, these components are not concerned with data partitioning.

5.3 Elasticity

The data should be stored on a distributed database backend. More precisely, each Sharemind Application Server will need a separate cluster for the backend database. We assume that the backend database is elastic.

Since our solution does not promise transaction safety, handling of concurrent writes and reads must be handled by the application. Therefore, we just launch multiple Sharemind deployments, that use the same distributed database backend. For insertion of data, there is no need to synchronize anything, the same for read-only workload. Therefore, if we have a scalable database backend, we can run additional Sharemind deployments on top of it.

The only component not mentioned is the SQL frontend. Many instances of the SQL frontend can connect to one Sharemind deployment. Therefore, the SQL frontend also can be scaled independently of the rest of the components. Thus, it will not cause a bottleneck in the system. The load balancing, i.e., the act of choosing which Sharemind deployment the SQL frontend connects to, is left for the application.

Handling the elasticity is not automated and in large part is up to the application that uses the Secure Multi-Cloud Application Server. Adding and removing of resources is left entirely up to the application. However, automated tools can be built on top of Secure Multi-Cloud Application Server.

5.4 Discussion

Solution 3: Secure Multi-Cloud Application Server can work in an elastic setting, but it lacks the mechanism for supporting elasticity out of the box. Implementing elasticity on top of it is dependent on the actual use case, because depending on the specific deployment scenario, various trade-offs can be made.

Cybernetica has done some experiments in a "map-reduce" setting, where multiple Sharemind deployments were used in parallel [BJS+16].

6 Conclusion

To sum up, this document shows that each of the SafeCloud Secure Queries' solutions was designed to support elastic and fault-tolerant deployments. Solutions 1 and 2 have common architectures and most components (Query processing, Transactional module, Trusted Site CryptoWorker) are stateless or are thought to inherently be elastic and resilient. The mains difference regarding these two solutions is in the privacy-preserving techniques being used and in the untrusted NoSQL (HBase) backend deployment. As discussed in the document, Solution 1 can fully leverage the elasticity and fault-tolerant characteristics of HBase. On the other hand, Solution 2 resorts to multi-party protocols that strictly require three independent NoSQL (HBase) cluster deployments and an in-place communication protocol across these. These assumptions require additional components, such as a discovery service, to ensure that the solution remains elastic while ensuring the consistency guarantees expected by applications using the database. Moreover, although each HBase cluster is elastic and tolerates failures, the multi-party protocol requires three clusters thus restricting the elasticity and increasing the complexity of having a fault-tolerant solution in a cluster-wide fashion. Finally, although it is out of the scope of the project, for both solutions, is it still possible to further improve NoSQL elasticity to be fully automatic by integrating previous work in this field.

For solution 3 the elasticity is not actually built into the system, but the solution is modular and elasticity can be built on top of solution 3. Since solution 3 uses MPC same as solution 2, similar restrictions apply: the number of clusters is fixed to 3.

7 References

[AFG+10] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Commun. ACM* 53,4 (April 2010), 50-58, 2010.

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

[APACHE17a] Apache HBase Team. "Apache HBase ™ Reference Guide". (https://hbase.apache.org/book.html), 2017

[APACHE17b] Apache Hadoop documentation. (https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html), 2017

[APACHE17c] Apache Omid (Optimistically transaction Management In Datastores) documentation. (https://omid.incubator.apache.org), 2017

[APACHE17d] Apache Zookeeper documentation. (http://zookeeper.apache.org), 2017

[BJS+16] Dan Bogdanov, Marko Jõemets, Sander Siim, Meril Vaht. "Privacy-preserving tax fraud detection in the cloud with realistic data volumes." Cybernetica research report T-4-24. 2016.

[BHK+17] Bortnikov, E., Hillel, E., Keidar, I., Kelly, I., Morel, M., Paranjpye, S., Perez-Sorrosal, F., and Shacham, O. "Omid, Reloaded: Scalable and Highly-Available Transaction Processing." In FAST, pp. 167-180. 2017.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4 (June 2008), 26 pages. DOI=http://dx.doi.org/10.1145/1365815.1365816

[CMM+13] Cruz, F., Maia, F., Matos, M., Oliveira, R., Paulo, J., Pereira, J., and Vilaça, R. 2013. MeT: workload aware elasticity for NoSQL. In *Proceedings of the 8th ACM European Conference on Computer Systems* (EuroSys '13). ACM, New York, NY, USA, 183-196.

[GB12] Galante, G., and Bona, L. "A survey on cloud computing elasticity." In Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, pp. 263-270. IEEE, 2012.

[HHL+11] Han, J., Haihong, E., Le, G., and Du, J. "Survey on NoSQL database." In Pervasive computing and applications (ICPCA), 2011 6th international conference on, pp. 363-366. IEEE, 2011.

[MPP+17] Macedo, R., Paulo, J., Pontes, R., Portela, B., Oliveira, T., Oliveira, R., Matos M. “A Practical Framework for Privacy-Preserving NoSQL Databases” (to appear) 36th IEEE International Symposium on Reliable Distributed Systems (SRDS), 2017.

[MYSQL17] ORACLE. MySQL replication documentation. (https://dev.mysql.com/doc/refman/5.7/en/replication.html), 2017.

[PMP+16] Pontes R, Maia F, Paulo J, Vilaça R. 2016. SafeRegions: Performance Evaluation of Multi-party Protocols on HBase. 2016 IEEE 35th Symposium on Reliable Distributed Systems Workshops (SRDSW), 2016.

[WPS+00] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., and Alonso, G. "Database replication techniques: A three parameter classification." In Reliable Distributed Systems, 2000. SRDS-2000. Proceedings The 19th IEEE Symposium on, pp. 206-215. IEEE, 2000.