SafeCloud

SafeCloud Architecture

D4.1

Project reference no. 653884

August 2016

European Commission

Horizon 2020
European Union funding
for Research & Innovation

Document information

Scheduled delivery	01.09.2016
Actual delivery	18.09.2016
Version	1.8
Responsible Partner	CYBER

Dissemination level

Public

Revision history

Date	Editor	Status	Version	Changes
04.07.2016	K. Tarbe	Draft	0.1	Initial TOC
18.08.2016	K. Tarbe	Draft	0.2	Integrated inputs from others
29.08.2016	K. Tarbe	Draft	0.5	Address the reviews
31.08.2016	M. Barbosa	Revision	0.7	Additional reviews
01.09.2016	J. Paulo and F. Maia	Final	1	Final version
18.09.2016	K. Tarbe and R. Rebane	Final	1.8	New release with the changes discussed during the SafeCloud Fall 2016 meeting in Munich.

Contributors

K. Tarbe (CYBER)
R. Rebane (CYBER)
H. Mercier (UniNE)
M. Correia (INESC-ID)
J. Paulo (INESC TEC)
F. Maia (INESC TEC)
B. Portela (INESC TEC)
M. Barbosa (INESC TEC)

Internal reviewers

P. Sousa (Maxdata)
V. Schiavoni (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
AES	Advanced Encryption Standard
API	Application Programming Interface
CA	Certificate Authority
FUSE	Filesystem in Userspace
IP	Internet Protocol
JDBC	Java Database Connectivity
MPC	Multi-Party Computation
MPTCP	Multi-Path TCP
NoSQL	Not Only Structured Query Language
ODBC	Open Database Connectivity
PKI	Public Key Infrastructure
REST	Representational State Transfer
SOCKS	Socket Secure (Network proxy protocol)
SQL	Structured Query Language
TCP	Transport Control Protocol
TLS	Transport Layer Security
TTL	Time To Live
UDP	User Datagram Protocol

Table of contents

Executive summary

SafeCloud architecture/framework is composed by three main layers that target secure communication, secure storage, and secure data processing. These layers consist of multiple solutions that each solve a problem in their area. In this document we provide the reader with an overview of every solution in the SafeCloud architecture, how to deploy and integrate them to applications and a brief description of the security guarantees they provide. We also give concrete examples of how the solutions are deployed for the Maxdata and Cloud&Heat use cases.

1 Introduction

The components of the general SafeCloud framework are depicted in Figure 1. The framework consists of three separate layers, each providing solutions in their own technological domain. The solutions provide different security guarantees at different costs. Generally, stricter security guarantees impose greater functional limitations or performance costs on the applications using the solution. The three layers of the SafeCloud framework are secure communications, secure storage and secure query processing.

The secure communications layer provides solutions that improve the security aspects of communication channels over some untrusted environment. We tackle three important challenges in this layer. First, the vulnerability-tolerant channels solution gives communication channels that are built on multiple redundant security mechanisms to ensure that failure of any one mechanism does not cause a security failure in the channel. Second, the protected channels solution introduces multiple methods to reduce the risk of fake certificates used by the parties. This solution also makes it more difficult to run port scans and do enumeration of the network infrastructure. Third, the route-aware channels solution deploys methods to improve confidentiality and detect route hijacking. All of the solutions are built on top of the Java Secure Socket API.[1] Further details for this layer are discussed in WP1 deliverable D1.1.

The secure storage layer consists of solutions that provide confidentiality and integrity guarantees for data stored in an untrusted environment. The secure block storage and the secure file system give similar secure storage benefits but with APIs at different levels of the stack. The long-term distributed encrypted archival solution provides an *entangled* immutable data store for protection against tampering and censorship. Further details for this layer are discussed in WP2 deliverables D2.1 and D2.2.

The secure query processing layer makes it possible to store encrypted data in untrusted environments while still being able to process it in useful ways. All the techniques in this layer provide SQL query processing in untrusted environments but differ in the degree on which data is kept undisclosed to the different parties and on the type of queries that can be performed. The deployment model is also different. In the first approach data is stored and processed in a single untrusted domain, while in the second data is stored and processed across multiple untrusted domains. Additionally, the third solution contemplates several data owners that wish to perform cross computation across their data without revealing any sensitive information to each other. Further details for this layer are discussed in WP3 deliverables D3.1 and D3.2.

[1] https://docs.oracle.com/javase/8/docs/api/javax/net/ssl/package-summary.html

SafeCloud architecture

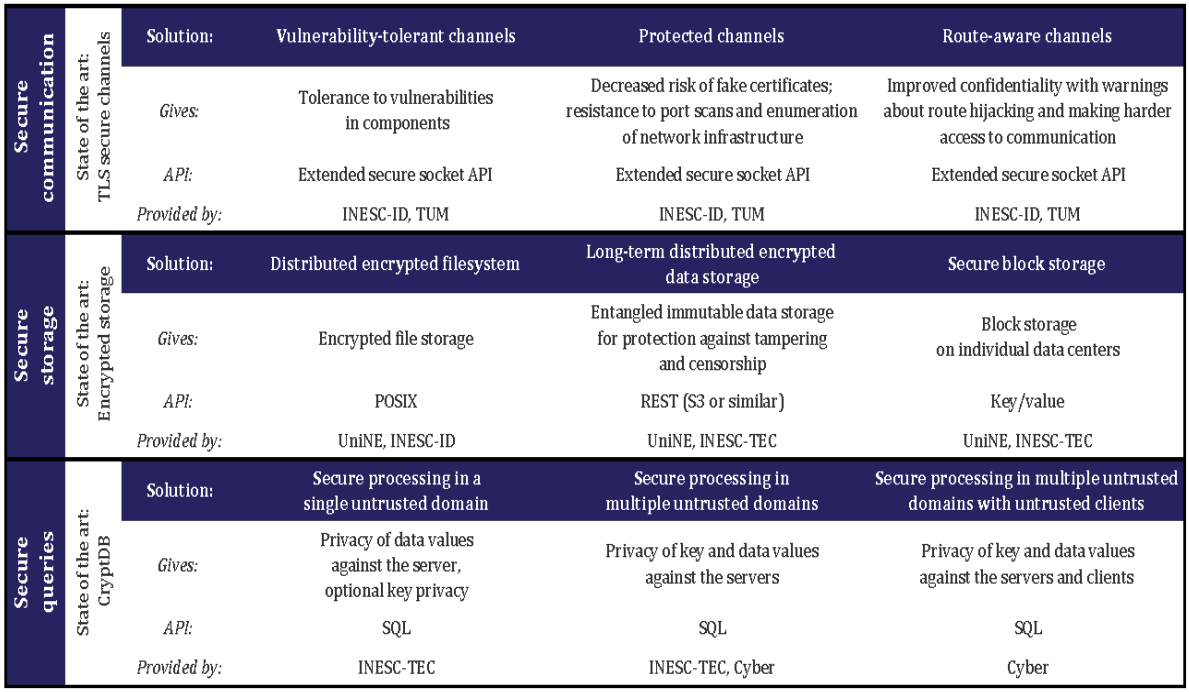

Figure 1: SafeCloud architecture components.

This deliverable provides an overview of the whole SafeCloud architecture and how it suits the requirements of the project's use cases. In the following sections we describe the features, the deployment and integration views of the architecture and security guarantees for each solution in every layer.

2 Secure communication

The secure communications layer provides protocols that improve the security aspects of communication channels over untrusted environments. We briefly describe three kinds of novel security-enhanced communication channels in this layer. Further details can be found in deliverable D1.1.

2.1 Vulnerability-tolerant channels

Vulnerability-tolerant channels (Secure Communication 1 or SC1) is a solution that is similar to Transport Layer Security(TLS) protocol. It exceeds TLS by utilizing redundant cryptographical mechanism to provide communication channels with data confidentiality, integrity and authenticity even in the case of zero day vulnerabilities found in a subset of the used mechanisms.

2.1.1 Features

Diversity and redundancy. SC1 uses k different cipher suites together. Up to $k - 1$ compromised suites the channel remains secure. SC1 should be considered for mission-critical applications, because it provides additional protection against vulnerabilities that may be found in each of the underlying cypher suites.

Java Secure Socket API. SC1 leverages the indirection provided by Java Secure Sockets. The creation of a socket is done through a factory object that abstracts the concrete class that is instantiated. Therefore, SC1 remains API compatible with the existing Java Secure Socket API.

2.1.2 Deployment and integration

Stakeholders

1) **Client** - establishes a connection, wants to make sure that the channel provides authenticity, confidentiality and integrity.
2) **Server** - awaits connection requests, wants to make sure that the channel provides authenticity, confidentiality and integrity.

Security-wise both Client and Server are symmetrical.

SC1 components

1) **SC1 Client** - the component to be deployed by the Client, provides replacement for the Java Secure Socket API and vulnerability tolerant channels.
2) **SC1 Server** - the component to be deployed by the Server, provides replacement for the Java Secure Socket API and vulnerability tolerant channels.

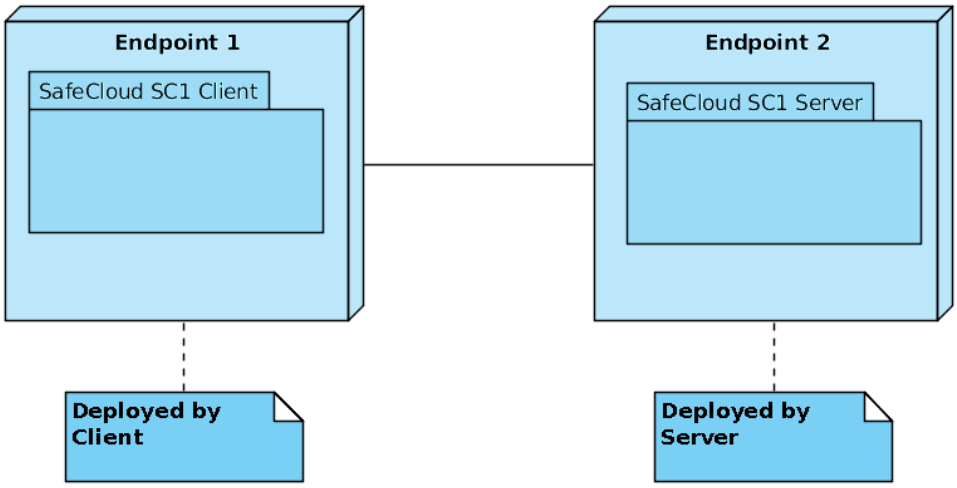

Figure 2: SafeCloud SC1 deployment diagram

The deployment of SC1 is straightforward: SC1 Client component is deployed by the Client and SC1 Server component is deployed by the Server as is depicted on Figure 2.

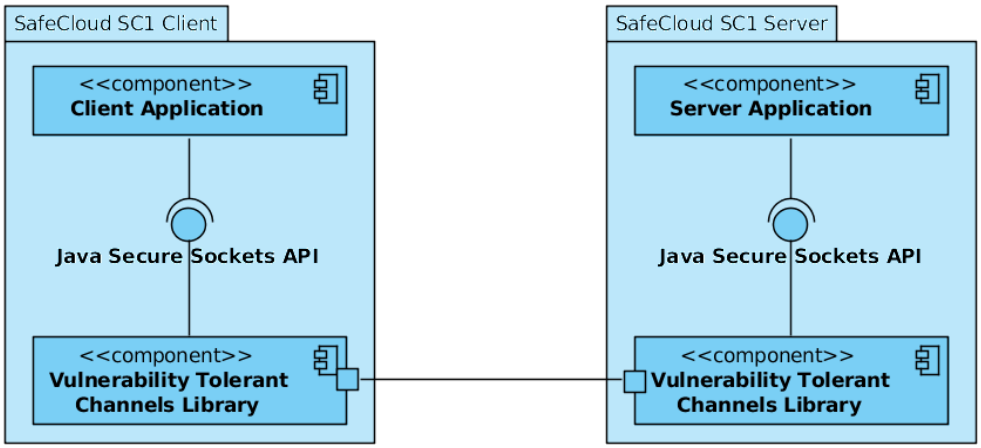

Figure 3: SafeCloud SC1 component diagram

Figure 3 describes the components in more detail. SC1 Client and Server Components are mostly identical. Client / server application uses Java Secure Sockets API to interface with the Vulnerability Tolerant Channels Library, which in turn provides the implementation for Java Secure Sockets.

2.1.3 Security

Trust assumptions

1) The Client and Server are trusted.
2) The network between Client and Server is untrusted.
3) There is an upper bound for the number of cryptographic mechanisms that can be compromised.

Client & Server. Both endpoints of a secure channel will see the data flowing through the channel. They are interested that they are communicating with each other not with an impostor, no other party can decipher the data and data is not changed while in transit. In other words, SC1 provides authenticity, confidentiality and integrity.

Network. Internet is a heterogeneous global network, where connections usually go through many different intermediate nodes. The network is prone to malicious agents,

who might want to spy on the communication between the Client and the Server or alter the communication.
Cryptography. Communication is protected with *k* cryptographic mechanisms at the same time, and up to *k – 1 compromised* mechanisms the communication is secure.

2.2 Protected channels

Protected channels (Secure Communication 2 or SC2) provide protection against service discovery. In this solution services are behind a firewall which blocks all connection attempts unless a successful port knocking is performed.

2.2.1 Features and usage

Complete coverage. SC2 can cover the whole machine, meaning that port knocking is needed for all inbound connections. Connection attempts made without a valid port knock are blocked by the firewall.
PKI based authentication. Servers and clients all have certificates. Server needs only the CA certificate to verify the validity of clients' port knocking requests. Certificate revocation lists are used to revoke access.
Proxy service. For legacy applications that cannot use the port knocking client library, there is a proxy service that handles the port knocking transparently. Client applications connect to the proxy using SOCKS protocol and ask the proxy service to open the connection to the protected server.

SC2 hides available services from a port scanner. This reduces the attack surface: an attacker cannot determine the vulnerable services and cannot access them, because without a successful port knocking, the attacker is blocked by a firewall. This solution can be used on all servers to additionally protect remote administration services.

2.2.2 Deployment and integration

Stakeholders

1) **Client** - sends the port knocking packet, then establishes connection.
2) **Server** - runs a service to listen for port knocking requests.

SC2 components

1) **SC2 Client library** - the component to be deployed by the Client. Provides a mechanism for sending the port knocking request.
2) **SC2 Server component** - the component to be deployed by the Server. Listens for port knocking requests and opens ports for specific clients in the firewall.
3) **SC2 Proxy** - the component to be deployed by the Client. Provides a proxy service for applications that cannot use the SC2 Client library.

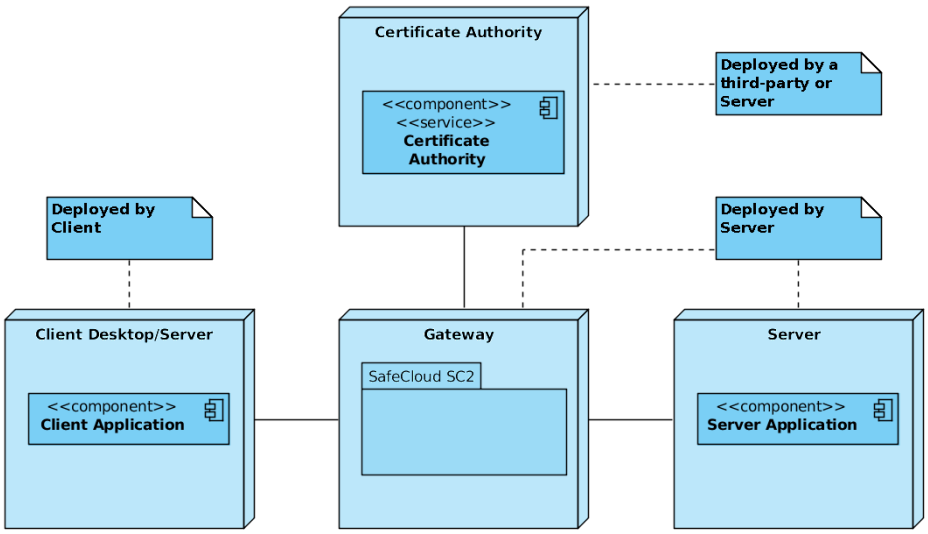

Figure 4: SafeCloud SC2 deployment diagram

Figure 4 illustrates the relations between the stakeholders and the component they deploy. We assume that there is a trusted PKI. PKI can be deployed and administered by the cloud service provider.

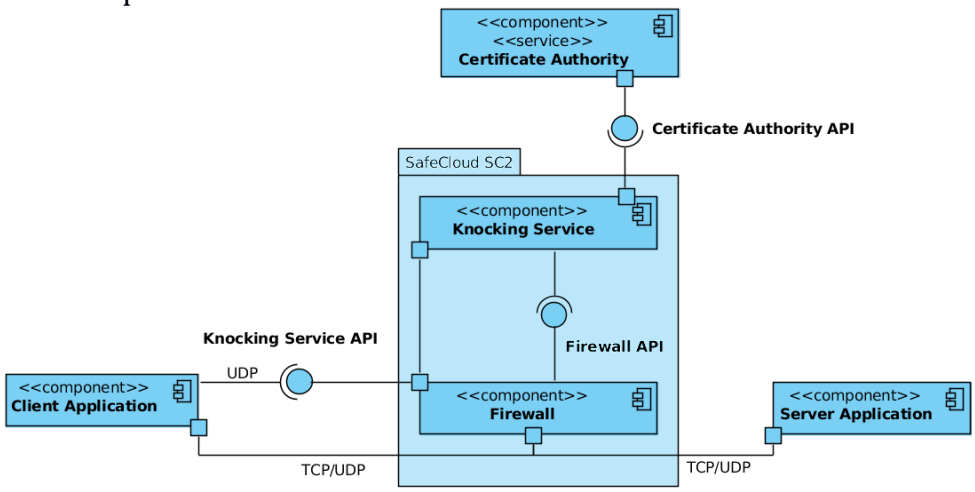

Figure 5: SafeCloud SC2 component diagram (without proxy)

The architecture of the SC2 is depicted on Figure 5. Client application uses the SC2 Client library and the PKI to construct valid port knocks sent over UDP to the server.
Knocking Service on the server is integrated with the Firewall: it rejects all connections by default, expect port knocks which are handled by the Knocking Service. In the case of a valid port knock, the Knocking Service will use the Firewall API to allow packets from the Client through the Firewall.

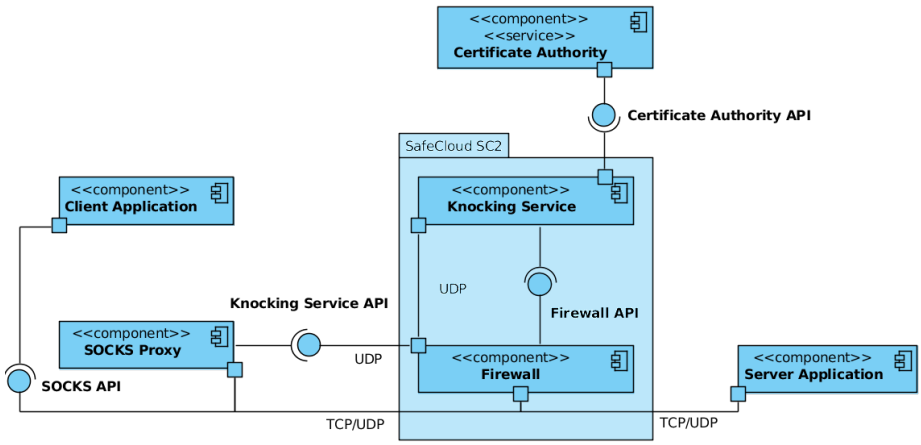

Figure 6: SafeCloud SC2 component diagram (with proxy)

The use of the optional Proxy component is depicted on Figure 6. In this scenario the port knock is sent by the SC2 Proxy. This is useful to run existing client applications without further modifications.

2.2.3 Security

Trust assumptions

1) CA and PKI are trusted.
2) Firewall component is trusted.

PKI. PKI is used for exchanging the public keys needed for verifying the identity of participants.

Firewall. Most of the security comes from the firewall component that blocks unauthorized (without a valid port knock) connections.

2.3 Route-aware channels

Route-aware channels (Secure Communication 3 or SC3) provide active and passive methods to detect anomalies in the network path. This helps to detect IP prefix hijacking. Also multiple network paths are used to make eavesdropping on the connection harder.

2.3.1 Features and usage

Route monitoring. SC3 will monitor the number of intermediate nodes between endpoints. It uses TTL (Time-To-Live) field in TCP headers, and also various approaches based on Traceroute, Secure Traceroute [PS03] and IP options (Record Route, Strict Source Record Route, Loose Source Route). In addition, SC3 will monitor network latency, bandwidth and packet loss.

Anomaly detector. SC3 will keep a history of metrics collected and runs statistical analysis on the metrics to detect anomalies.

Multi-path routing. SC3 will provide ways to achieve multi-path routing with the help of overlay network consisting of relays. Multi-path routing splits data across different network paths and therefore may make it hard for an eavesdropper to get access to the communication between two nodes.

2.3.2 Deployment and integration

Stakeholders

1) **Client** - is one endpoint of the communication.
2) **Server** - is the other endpoint of the communication.
3) **Relay Host** - provides proxy services to allow multi-path routing.

Client and Server are just two entities who want to communicate. Relay Hosts consist in a number of relays forming an overlay network that will be used for multi-path routing between Client and Server.

SC3 components

1) **TTL monitor** - the component to be deployed on both communication endpoints (Client and Server). Provides multiple ways to determine the number of hops on the communication path.
2) **Route metrics monitor** - the component to be deployed on both communication endpoints (Client and Server). Provides metrics like network latency, bandwidth and packet loss.
3) **Anomaly detector** - the component to be deployed on both communication endpoints (Client and Server). Finds anomalies by using statistical analysis on the metrics collected by the monitors.
4) **Relay service** - the component to be deployed on Relay Host. Provides application-layer routing for multi-path communication.
5) **Membership service** - the distributed service to be used by all Stakeholders, provides a membership service, so relays do not have to be fixed. Multi-path communication uses the information from membership service to choose the actual relays and paths used. The membership service has to be secure so replication and cryptographic techniques will be used to achieve this goal.

SC3 consists of two related services: route monitoring and multi-path routing. In the reminder, we first discuss deployment and architecture for route monitoring, and then the deployment and architecture for multi-path communication.

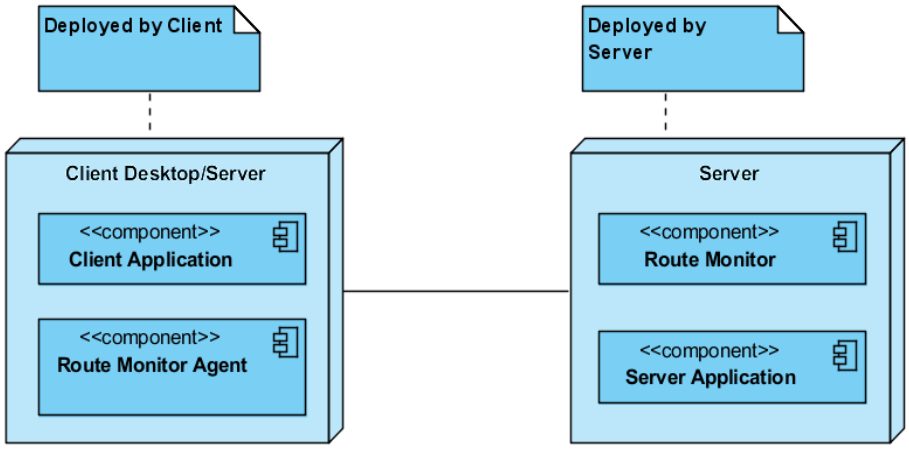

Figure 7: SafeCloud SC3, route monitor deployment diagram

Figure 7 illustrates the relations between stakeholders and the deployed components. Route monitoring is active on both endpoints. The Client and Server components will deploy the Route Monitor.

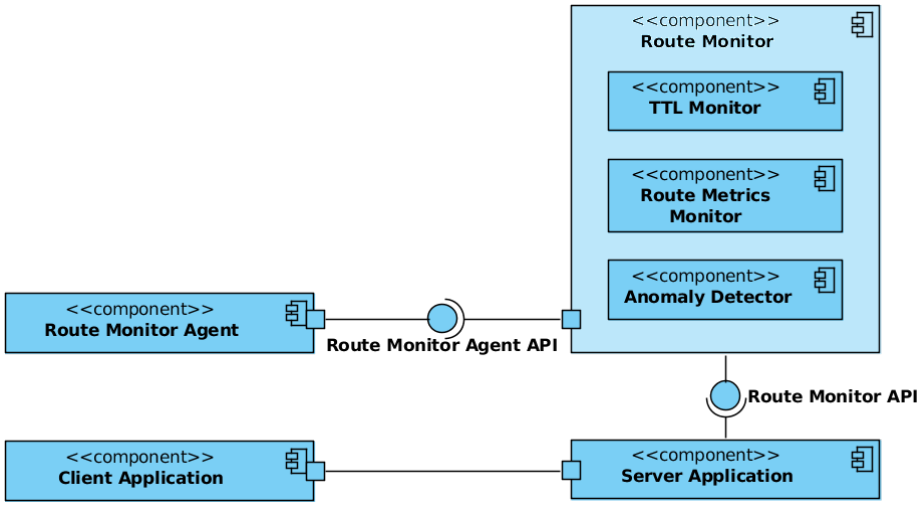

Figure 8: SafeCloud SC3, route monitor component diagram

The components of SC3 route monitoring are depicted on Figure 8. Route Monitor is a service that consists of many components which extract data. An additional Anomaly Detector component is deployed to detect anomalies using statistical analysis. Further details for each component can be found in WP1 deliverable D1.1.

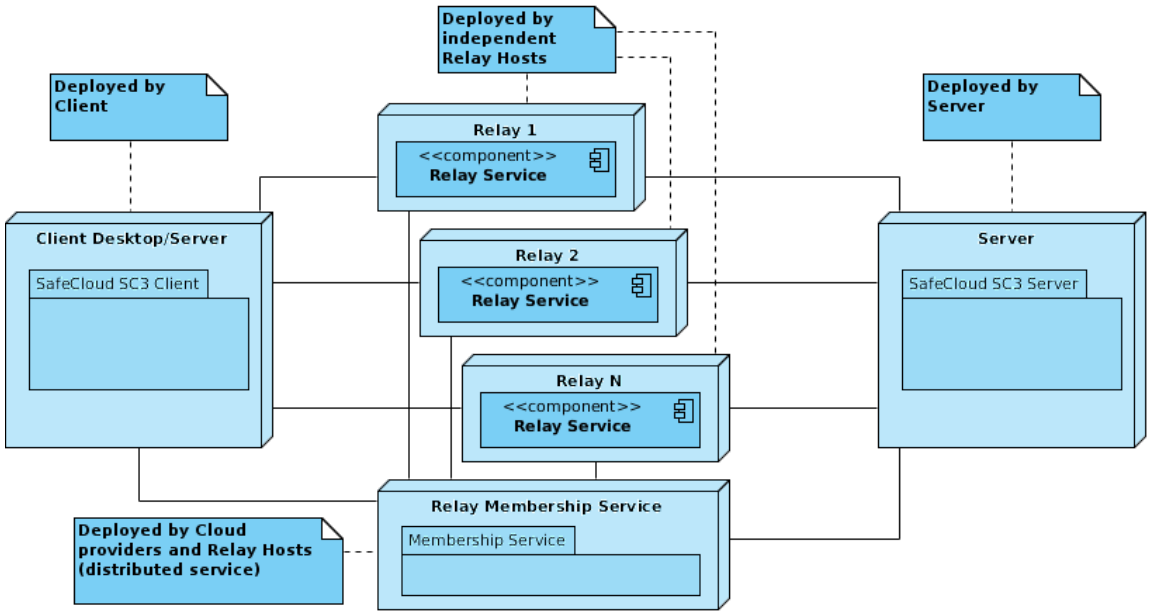

Figure 9: SafeCloud SC3, multi-path communication deployment diagram

Relations of SC3 multi-path communication stakeholders and deployed components are illustrated on Figure 9. For constructing overlay networks with distinct physical network paths, there are a number of Relay Service nodes needed. Different cloud providers currently host these relay service nodes, typically one per data center. To check the availability of relays, Relay Membership service exists, distributed between all stakeholders.

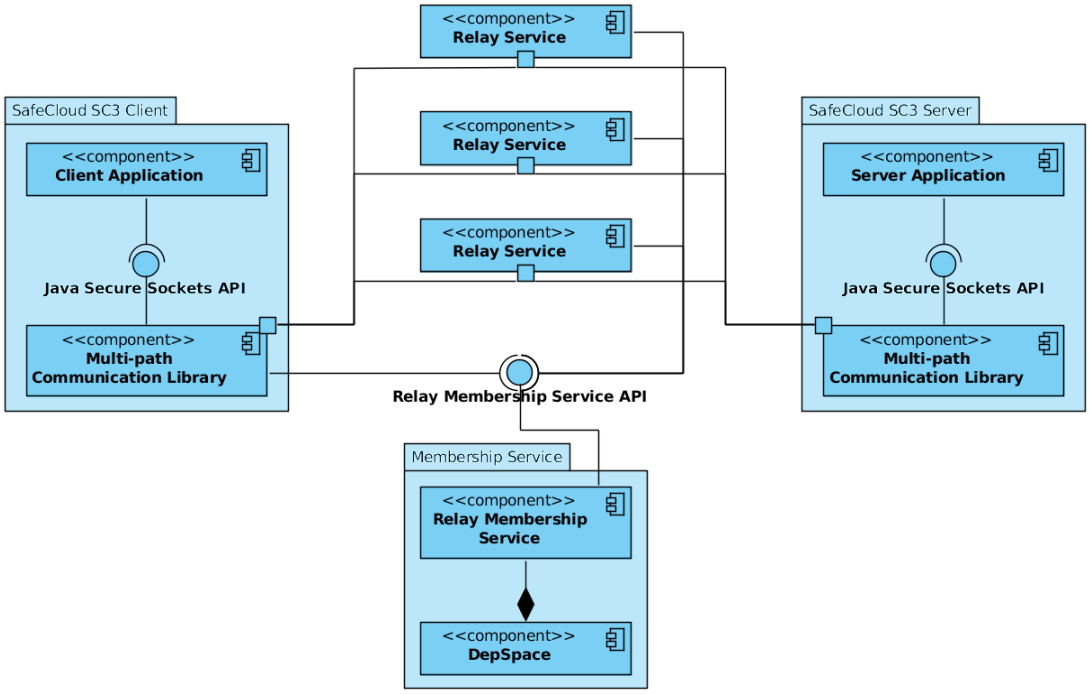

Figure 10: SafeCloud SC3, multi-path communication component diagram

Components of SC3 multi-path communication are depicted on Figure 10.

The Multi-path Communication Library is the component that is integrated into the Client and Server applications for providing multi-path communications through Java Secure Sockets compatible APIs.

The Relay Service is responsible for providing application-layer routing. Data flow is routed through distinct Relays to minimize the number of network hops that are shared between network paths.

The Relay Membership Service is a distributed service built on top of DepSpace [BAC+08], and it is used for listing active relays.

Further details regarding these components can also be found in WP1 deliverable D1.1.

2.3.3 Security

Trust assumptions

1) Relay membership service is trusted.
2) Network is untrusted.
3) There's an upper bound for the number of relays that can be compromised.

Relay membership service. Relay membership service is trusted, but it is replicated using an intrusion-tolerant protocol, so the assumption is that no more than a certain number of replicas is compromised.

Network. The adversary cannot eavesdrop on more than a certain number of communication paths over the Internet and control a certain number of relay nodes.

Relays. The adversary cannot control more than a certain number of relay nodes. Controlling all the relays, means that the adversary has access to all the data and physically distinct network path do not add any confidentiality.

2.4 Other solutions and comparison

Since the secure communication channels have different properties, it does not make much sense to compare them with each other. Here we refer to relevant related work for each of the SafeCloud Secure Communication mechanisms.

Vulnerability-tolerant channels

To the best of our knowledge there is no similar protocol available and it is the first time diversity approaches are applied to secure communication protocols.

Protected channels

Other solutions for port knocking like [AJ05], [K03], [VHT09] and [KG14] use shared secrets. In SafeCloud's Protected channels we use public key cryptography and PKI instead of pre-shared secrets.

Route monitoring

SafeCloud's mechanism for route monitoring is new, but leverages on existing methods. Other solutions that can detect ip prefix hijacking are [ZJP07] and [ZZH+08].

Multi-path communication

SafeCloud's multi-path communication scheme leverages a set of existing mechanisms — overlay routing, multi-homing, and MPTCP(Multi-Path TCP) — but combines them in a novel way.

3 Secure storage

The secure storage layer consists of services that provide security and dependability guarantees for data stored in an untrusted environment. Next we briefly describe each solution while further details can be found in deliverables D2.1 and D2.2.

3.1 Secure block storage

This SafeCloud service (Secure Storage 1 or SS1) is an abstraction that may be implemented in different ways, e.g., commercial cloud storage services like Amazon S3 or key/value stores like Cassandra. SafeCloud will implement an instantiation that provides a secure block storage.

3.1.1 Features and usage

Distributed key/value store. SS1 is essentially a key/value store that may include additional mechanisms to ensure data confidentiality and integrity. Individual storage instances are orchestrated by separate components, the block storage managers, which can explicitly place data on individual instances of the block storage.
Explicit placement. SS1 is able to explicitly place data in distinct locations to respond to the needs of the SafeCloud platform. Placement of data blocks can be enforced at multiple levels, e.g. administrative domain, application-defined partitions, geographical region or country, data center, rack, node, etc.
RESTful key/value store API. SS1 provides a REST-based key/value store API with support for explicit placement of data.

The Secure block storage targets clients who wish to outsource their data but want fine-grained control over how and where the data is stored. It can be used directly, but mostly serves as a building block for the other storage solutions of the SafeCloud platform.

3.1.2 Deployment and integration

Stakeholders

1) **Client** - owns the data, it is interested in outsourcing the storage.
2) **Storage Provider** - provides the infrastructure and platform for storing the data. The Secure block storage can be deployed on a single storage provider with multiple storage nodes or data centers, and on multiple storage providers.

SS1 components

1) **SS1 Block Storage Manager** - the component to be deployed by the Storage Provider. It provides the REST API for the key/value store.
2) **SS1 Block Storage Instance** - the component to be deployed by the Storage Provider, typically many in the same data center. It contains the actual block storage device.

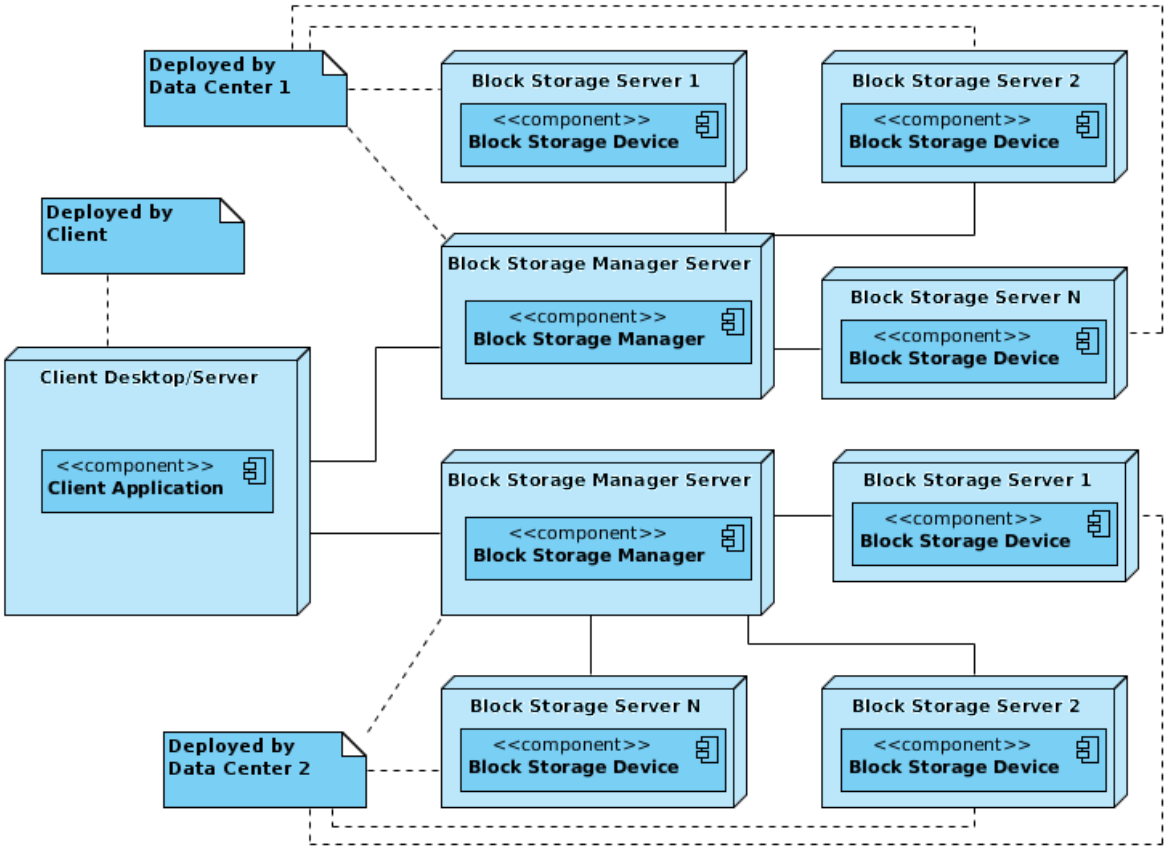

Figure 11: SafeCloud SS1 deployment diagram

Deployment of SS1 is illustrated on Figure 11, there are typically many Block Storage Instances per data center and at least one Block Storage Manager which manages the Block Storage Instances.

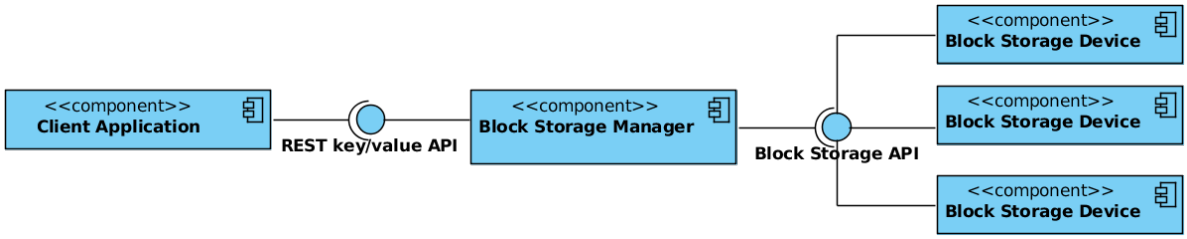

Figure 12: SafeCloud SS1 component diagram

Figure 12 shows the architecture of SS1. Client uses a REST API that acts as a key/value database. Block Storage Manager distributes the data across different Block Storage Instances.

3.1.3 Security

Trust assumptions

1) The Client is trusted.
2) The Storage Provider is untrusted.

Client. The Client has the rights to store and process the data. It is motivated to outsource storage to reduce its own infrastructure needs.

Storage Provider. The Storage Provider offers reliable storage services, which are focused on high availability and dependability. Typically, these services are encryption-agnostic, and a Client who wishes to store confidential data is responsible to encrypt it before storing it.

3.2 Long-term distributed encrypted data storage

This SafeCloud service (Secure Storage 2 or SS2) provides a long-term distributed encrypted data storage. It builds on top of Secure block storage(SS1) and makes the stored data tamper-resistant.

3.2.1 Features and usage

Amazon S3 compatible document store API. SS2 provides a REST-based document store API largely compatible with the Amazon S3 API.
Immutable tamper-resistant storage. Documents stored in SS2 will be stored redundantly and will be protected against tampering with coding and entanglement techniques, i.e., they are encoded and combined with previous documents to ensure that no party can modify or delete them without affecting a significant portion of all documents.

A typical application for SS2 is a bank interested in storing its transaction records so that they cannot be tampered with. Other examples include financial transactions, medical records and scientific data. The purpose of SS2 is similar to that of a blockchain, but allows the storage of large amounts of data.

3.2.2 Deployment and integration

Stakeholders

1) **Client** - owns the data, wants to protect it from tampering.
2) **Service Provider** - provides the infrastructure to encode/decode the documents.
3) **Storage Provider** - provides the infrastructure for storage.

Components

1) **Document Storage Manager** - the component to be deployed by the Service Provider. It provides the REST document store API.
2) **Encoder** - the component to be deployed by the Service Provider. It handles the encoding, entanglement and reconstruction of documents.
3) **Storage Service** - the service to be deployed by Storage Provider. It stores the encoded and entangled documents.

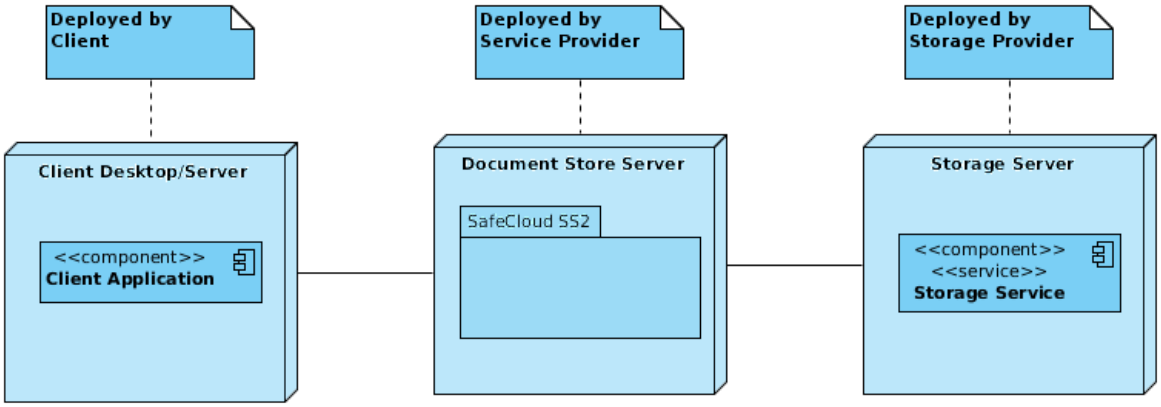

Figure 13: SafeCloud SS2 deployment diagram

Figure 13 shows the deployment of components needed for SS2. Document Store Server requires moderate processing power to encode and reconstruct the documents, whereas Storage Server requires storage.

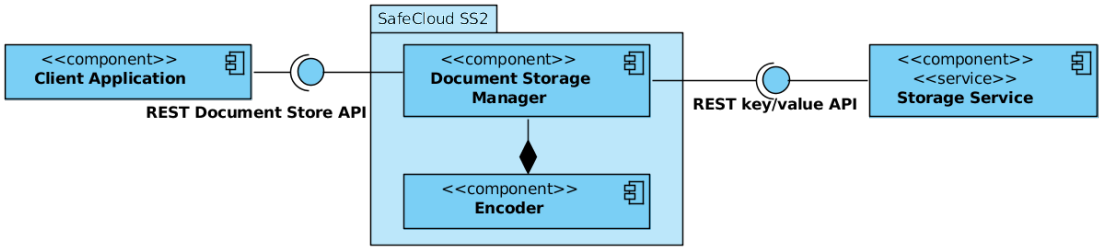

Figure 14: SafeCloud SS2 component diagram

Figure 14 illustrates the architecture of SS2. Client Application talks with the Document Storage Manager with a REST API that is compatible with the Amazon S3. Document Storage Manager also interacts with the storage service using a REST key/value API. Encoder is responsible for encoding and reconstructing the documents. Storage service can have multiple backends, but one of them is SafeCloud's Secure Storage 3.

3.2.3 Security

SS2 is built on top of SS1 and inherits its security features. The additional SS2 layer provides integrity and protection against tampering and censorship.

Trust assumptions

1) The Client is trusted.
2) The Storage Provider is untrusted.

Client. The Client has the rights to archive and process the data. It is motivated to outsource storage to reduce its own infrastructure needs. It is assumed that the data is immutable.

Storage Provider. The Storage Provider provides the storage infrastructure and wishes to convince its customers of its availability and reliability. The entanglement of data makes it very difficult for the Storage Provider to tamper with it without affecting the entire system.

3.3 Secure file system

This SafeCloud solution (Secure Storage 3 or SS3) provides a distributed file system. SS3 adds a filesystem API on top of Secure block storage(SS1). The solution takes care of encrypting the data.

3.3.1 Features and usage

POSIX compliant file system API. SS3 provides a FUSE-based file system API that is designed to be mostly POSIX compliant.
Policies for security and dependability. Placement of data in the file system can be guided by policies that express security and dependability requirements like replication degree, tolerance against whole data centre failure and geo-localization of data.

A first example for SS3 is provided by our first use case: Cloud&Heat operates micro-clouds and wishes to optimize data storage and processing among a large number of very small clouds. A second application comes from our second use case: Maxdata must physically store medical data in each patient's country.

3.3.2 Deployment and integration

Stakeholders

1) **Client** - owns the data and is interested in storing it in the cloud.
2) **Service Provider** - provides the infrastructure and platform for securely storing data.

SS3 components

1) **Client component** - the component to be deployed by the Client. It is based on FUSE and provides the file system API for the external storage.
2) **Storage service** - the service to be deployed by the Service Providers. It provides the storage service.

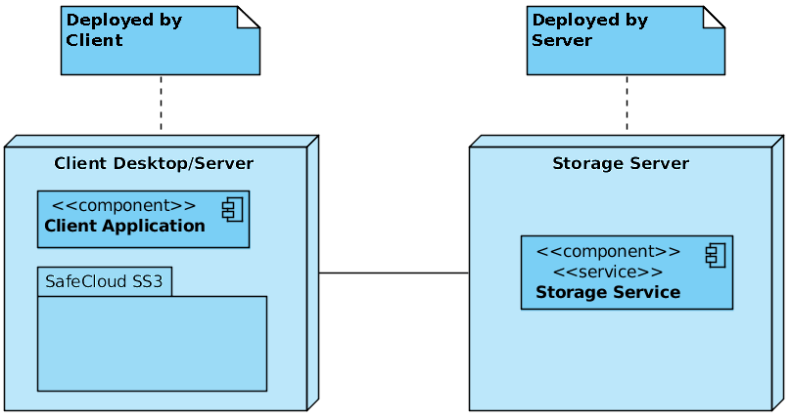

Figure 15: SafeCloud SS3 deployment diagram

Figure 15 shows the relations between stakeholders and the components they deploy. The Storage service is deployed by a set of Service Providers, i.e., it is replicated. There needs to be a lightweight service running on the Client deployment to run the FUSE-based component to provide a file system.

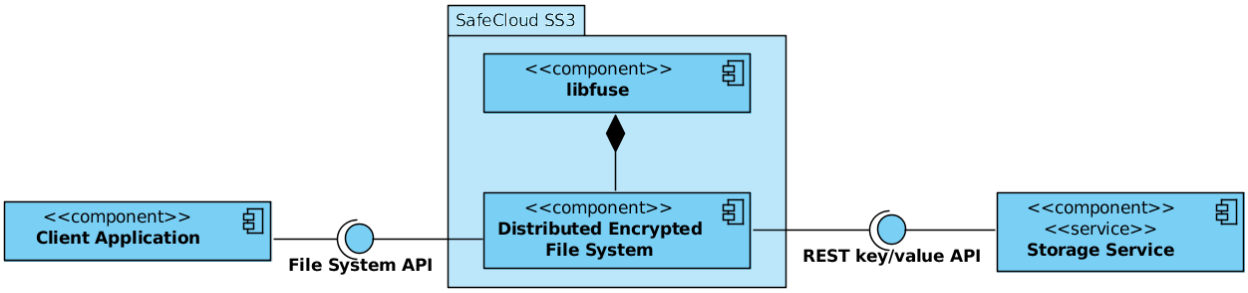

Figure 16: SafeCloud SS3 component diagram

The architecture of SS3 is depicted on Figure 16. The client interacts with the system by using the File System API. The file system is implemented using FUSE. The file system communicates with the storage services using a RESTful key/value API.

3.3.3 Security

SS3 is built on top of SS1 and inherits its security features.

Trust assumptions

1) The Client is trusted.
2) The Storage Providers are untrusted.

Client. The Client has the rights to store and process the data. It is motivated to outsource storage to reduce its own infrastructure needs.

Storage Provider. The Storage Providers deliver the storage infrastructure, typically the SafeCloud's block storage abstraction (SS1). The secure file system (SS3) is deployed by a set of Service Providers, i.e., it is replicated. This replication is critical for obtaining availability, integrity, and disaster tolerance. Byzantine fault-tolerant protocols are used to ensure the consistency of the data stored in the file system despite the failure of individual clouds or Storage Providers. Other mechanisms used include symmetric encryption, erasure coding, secret sharing, homomorphic encryption, and proofs of storage. Consult deliverables D2.1 and, in M18, D2.4 for more details.

3.4 Comparison

Table 1 gives a quick overview of different solutions for secure storage. They offer different capabilities. X shows that a feature is supported. Consult deliverables D2.1 and D2.2 for technical details about solutions in SafeCloud secure storage layer.

Feature \ Solution	SS1	SS2	SS3
Encrypted storage	X	X	X
Mutable data	X		X
Immutable data		X	
Physical placement of data blocks	X	X	X
Replication level control	X	X	X
Protection against data tampering		X	

Table 1: Comparison of solutions for secure storage

4 Secure queries

Real world deployments can consider wildly different scenarios with respect to security, functionality and performance requirements. Towards constructing a framework that can feasibly attend to these possibilities, the secure queries layer proposes three fundamentally different solutions, which can be instantiated to deploy tailor-made implementations to a wide variety of application scenarios.

The solutions in the secure queries layer make it possible to store data in untrusted environments while still being able to process the data. To achieve this goal, two main components exist in SafeCloud's processing architecture. The first is a SQL query engine that exports a SQL API to the clients and translates SQL queries into NoSQL operations. Then, a NoSQL backend is used to process the operations received from the query engine. In SafeCloud we chose to use HBase as the NoSQL backend since it is a highly known and used NoSQL database.

This separation is important to achieve a scalable solution. Since the query engine relies on the NoSQL backend storage and querying capabilities, it is easier to scale-up this component *i.e.*, to have several query engines running in a distributed setup. On the other hand, NoSQL backends are known for their inherent scalability. These databases don't support a complex query interface such as the SQL one, which is tackled by the translation mechanism built-in the query engine. The details of each component are further discussed in deliverables D3.1 and D3.2.

As another important detail, SafeCloud processing solutions are thought for a scenario where clients may have access or not to a trusted infrastructure (for instance, an on-premises infrastructure). This way, some of the solutions discussed next assume that the query engine component is deployed on a trusted infrastructure while the NoSQL backend is deployed on an untrusted infrastructure (for instance, a cloud infrastructure where data privacy must be protected from attackers). In such deployment, the query engine does not require privacy-aware security mechanisms while the NoSQL backend requires them. Ensuring that such security mechanisms are in-place is a major contribution of WP3.

In the remaining of this section we refer to the client or data owner infrastructure as the trusted environment and to the server or cloud infrastructure as the untrusted environment. Again further details regarding the deployment scenarios and secure processing solutions are discussed in D3.1 and D3.2.

The first solution (SQ1) considers a single trusted client interacting with a single untrusted deployment[2], where the challenge is to benefit from the untrusted deployment computational power while enforcing the required security guarantees. The second solution (SQ2) extends the previous scenario to allow for multiple untrusted deployments, which allows for employing security mechanisms that hinge on a more complex trust model. Finally, the third solution (SQ3) considers multiple (mutually untrusted) clients, allowing for joint querying of sensitive data for aggregated results, protecting the confidentiality of individual entries.

[2] In particular, this can be seen as an untrusted cloud/service provider, however the definition can encompass any instance of remote untrusted deployment.

A more in-depth discussion of cryptographic mechanisms and their deployment in the context of SafeCloud can be found in D3.2.

4.1 SQ1: Secure processing in a single untrusted domain

This SafeCloud solution (Secure Queries 1 or SQ1) provides a secure database querying capability with the following deployment scenario: Data Owner has a trusted deployment and a single Service Provider with an untrusted deployment is used.

4.1.1 Features and usage

NoSQL interface. SQ1 is built on HBase[3] and uses it as a storage backend. Thus, the native interface is very similar to NoSQL. The SafeCloud cryptographic behaviour is transparent from the end-user perspective.
Transactional ANSI SQL interface. SQ1 provides a full transactional ANSI SQL interface by integrating with the technology from EC FP7 CumuloNimbo project[4]. This builds on the NoSQL interface and performs additional operations needed to complete the queries on the client side.

SQ1 is designed to be instantiated with a variety of cryptographic techniques, depending on the requirements of SafeCloud applications. These techniques are often associated with specific tradeoffs between security and functionality, which makes then adequate in different circumstances.

Prioritizing security, standard secret key authenticated encryption can be employed when robust security guarantees are necessary, sacrificing functionality on the untrusted deployment side (effectively disabling the possibility for data querying). As security requirements are relaxed, other alternatives to remote computation can be employed. Deterministic encryption [BBO07] and other variants, such as Message-Locked Encryption [BKR13], can have the untrusted deployment check for duplicate plaintexts without disclosing the information stored. Order-Preserving Encryption schemes [AKS+04] can actively preserve the relative order of the plaintexts in the ciphertexts, which enables for remote querying of encrypted data. Alternatively, approaches such as CryptDB [PRZ12] and Arx [PBP16] allow for other tradeoffs for encrypted data querying with varying levels of performance values and security assumptions. Furthermore, these techniques can be combined in the same system, enabling for more optimized deployments.

As an example application for SQ1, 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 in the clear outside of what the laboratory considers to be its trusted domain. The laboratory can then specify the security levels of the offloaded data to have tailor-made levels of security and functionality, e.g. highly sensitive data is stored using standard authenticated encryption while order-insensitive data can be securely stored and queried using techniques such as order-preserving encryption.

[3] https://hbase.apache.org/
[4] http://www.cumulonimbo.eu/

4.1.2 Deployment and integration

Stakeholders

1) **Data Owner** - controls the data to be processed, is interested in protecting it from the hosts of the query service (e.g., cloud).
2) **Service Provider** - provides the infrastructure and platform for storing and securely querying data.

We assume that the Data Owner's computing environment has some computational power but scarce storage resources, whereas the untrusted deployment provided by the Service Provider has both complementary computational power and the necessary storage capabilities.

SQ1 components

1) **SQ1 Client** - the component to be deployed by the Data Owner, provides a secure querying capability and connects to the SQ1 Server for storage.
2) **SQ1 Server** - the component to be deployed by the Service Provider, provides a secure NoSQL storage capability.

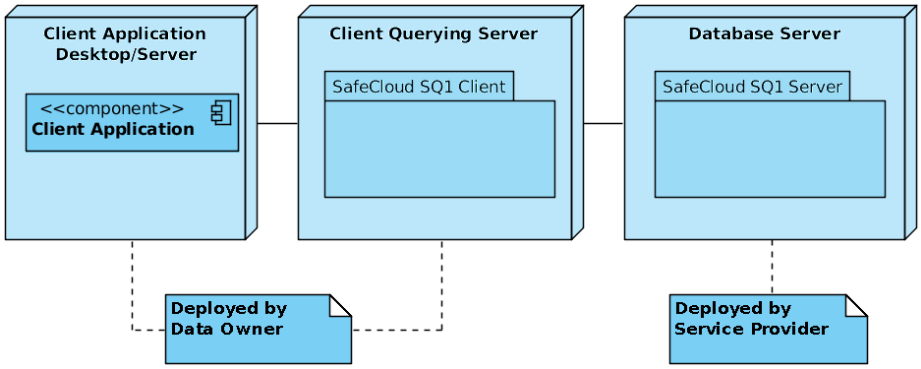

Figure 17: SafeCloud SQ1 deployment diagram

Figure 17 illustrates the relations between the stakeholders and the components they deploy. Note that the client application can be implemented in a number of ways: it can be an app on an application server, a standalone application, a mobile application, etc. It is observable that most of the processing is done by the Data Owner in the trusted deployment. Almost all storage, on the other hand, is left under the responsibility of the Service Provider.

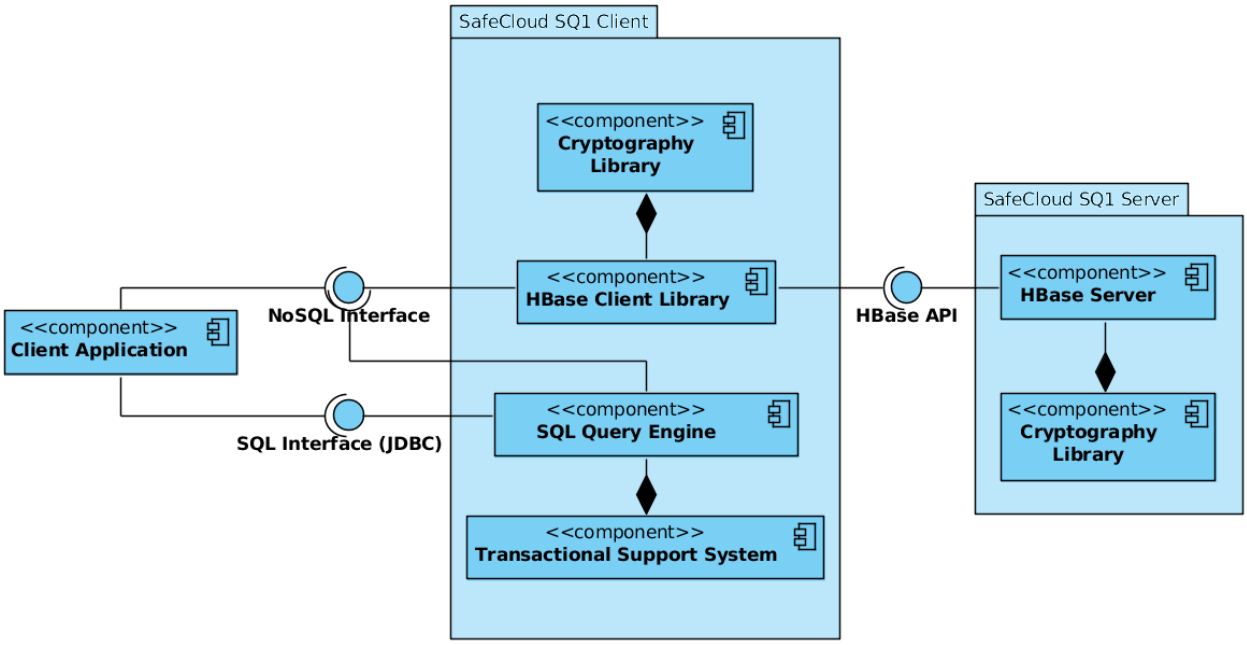

Figure 18: SafeCloud SQ1 component diagram

We give a more in-detail architecture of the components in Figure 18. In addition to the client application, the components are divided into two major parts: SQ1 Client and SQ1 Server. The client application has two interfaces available for use, offered by the SQ1 client: a SQL interface (accessible over JDBC) and a NoSQL interface (using the HBase Java API).
The SQL interface supports the full ANSI SQL standard which is implemented by the SQL Query Engine. The purpose of the SQL Query Engine is to translate SQL queries into NoSQL queries. It uses the same NoSQL interface that is offered to the Client Application. The Transactional Support System enables the use of transactions in SQL queries.
The NoSQL interface offers standard key-value database functions: get, put and scan. It is implemented by an extended version of the HBase Client Library which passes commands to the HBase Server.
The Cryptographic Library provides the functionality of handling encrypted data. The mechanisms employed will depend on schema specifications, which will ultimately instantiate the system according to application-specific security requirements. This component is central to the deployment of applications, as it will allow for HBASE core operations to be carried over encrypted data in a way that is transparent to the NoSQL engine.
In this setting, the SQ1 server consists only of an HBase server with its standard components.

4.1.3 Security

Trust assumptions

1) The Data Owner is trusted.
2) The Service Provider is untrusted.

Data owner. The data owner can freely process data as a trusted environment. This assumes that the trusted deployment in SQ1 is always protected from malicious activity, so all internal integrity/confidentiality attacks should be handled in an administrative

fashion. Implicitly, it is motivated (by regulation, contracts or general distrust) to enforce security mechanisms to protect the held data from untrusted environments.

Service Provider. The Service Provider provides infrastructure and platform services for customers to use. Since this is a platform that is considered by the Data Owner to be untrusted, the Service Provider should offer an environment to process over encrypted data, thus providing remote data processing to the client without significant data leakage.

From the fundamental benefits and limitations of the available cryptosystems, it is not possible to infer a scheme that is strictly better for deployment on all possible solutions. Furthermore, when considering real-world applications, the selection of techniques for specific requirements is hardly often straightforward. Since most practical solutions for cloud deployment consider a very reduced subset of cryptographic mechanisms, users lack the tools required for deploying implementations tailored to their application-specific security requirements.

SafeCloud's approach for SQ1 targets this significant niche of secure frameworks, allowing for solutions that opt into different cryptographic mechanisms for different security assumptions. This is not the first approach targeting a more granular approach to data protection (e.g. Google's Encrypted BigQuery[5], SAP's SEEED[6] and Microsoft's Always Encrypted SQL Server[7]), but should rather be interpreted as a natural progression towards improved user control with respect to secure data storage and processing.

4.2 SQ2: Secure processing in multiple untrusted domains

This SafeCloud solution (Secure Queries 2 or SQ2) provides a secure database querying capability. In this case more versatile mechanism can be offered than in SQ1, because in SQ2, there are multiple Service Providers with their untrusted deployments.

4.2.1 Features and usage

NoSQL interface. SQ2 is built on HBase and uses it as a storage backend. Thus, the native interface is very similar to NoSQL. The SafeCloud cryptographic behaviour is transparent from the end-user perspective.
Transactional ANSI SQL interface. SQ2 provides a full transactional ANSI SQL interface by integrating with the technology from EC FP7 CumuloNimbo project. This builds on the NoSQL interface and performs additional operations needed to complete the queries on the client side.

The design of SQ2 considers a scenario where multiple untrusted deployments are available. This extends the scenario of SQ1 for more complex trust models, where cryptographic techniques can be employed that benefit from this characteristic. In particular, secure multiparty computation protocols can be deployed over a set of untrusted participants if they can be assumed to not collude. This is often considered to

[5] https://cloud.google.com/bigquery/
[6] https://www.sics.se/sites/default/files/pub/andreasschaad.pdf
[7] https://msdn.microsoft.com/en-us/library/mt163865.aspx

be a realistic assumption for some use cases, as multiple cloud services are available in competing scenarios, and are therefore motivated to not cooperate.

Secret sharing with homomorphic properties [WCK+14, D+12, LTV12] has been a relatively common approach for delegating secure computations over untrusted participants. This is the one taken by Sharemind, which is employed in SQ2 for three main advantages: i) it allows SafeCloud to take advantage of a mature protocol where practical multiparty computation has been executed and evaluated, ii) the security/functionality considerations complement the alternative solutions SQ1 and SQ3, and iii) the know-how of the involved partners regarding the referred tool allows for comprehensively building upon and expand the framework state-of-the-art.

A typical example for the usage of SQ2 to instrument a solution would be a use case where highly sensitive data must be remotely processed. The employment of multiparty computation protocols allows for arbitrary functionalities to be run over protected data, while maintaining strong security guarantees for confidentiality and integrity.

4.2.2 Deployment and integration

Stakeholders

1) **Data Owner** - controls the data to be processed, is interested in protecting it from the host of the query service (e.g., cloud).
2) **Service Providers** - provide the infrastructure and platform for storing and securely querying data.

We assume that the Data Owner's computing environment has some computational power but scarce storage resources, while the untrusted deployment provided by three independent Service Providers have both complementary computational power and the necessary storage capabilities. We also assume that the Service Providers are non-colluding.

SQ2 components

1) **SQ2 Client** - the component to be deployed by the Data Owner, provides a secure querying capability and connects to the SQ2 Servers for storage.
2) **SQ2 Server** - the component to be deployed by the independent Service Providers, provides a secure NoSQL storage capability.

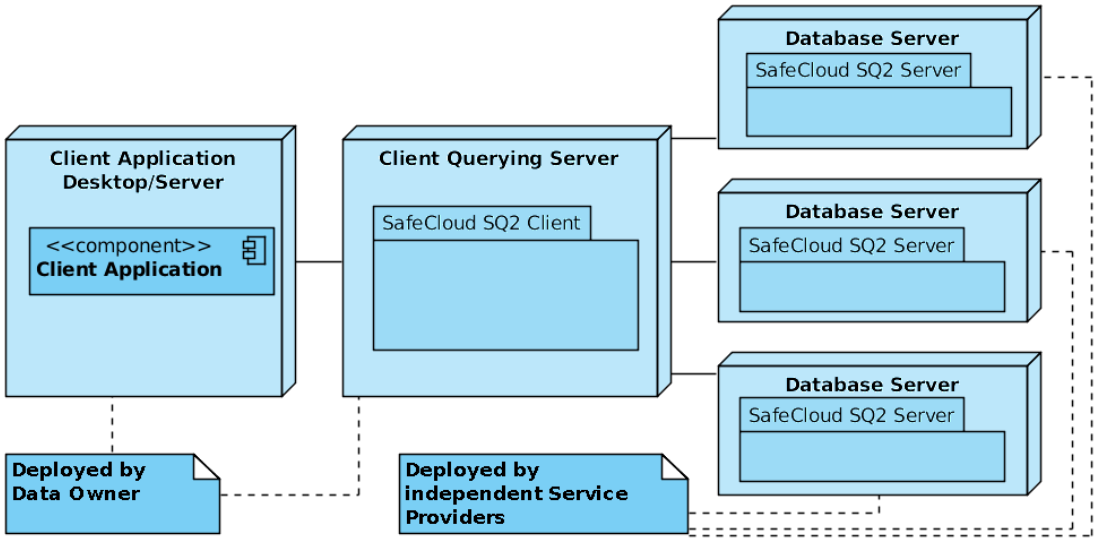

Figure 19: SafeCloud SQ2 deployment diagram

Figure 19 illustrates the relations between the stakeholders and the components they deploy. Like in SQ1, the client application can be implemented in a number of ways: it can be an app on an application server, a standalone application, a mobile application, etc. It is observable that most of the processing is done by the Data Owner in the trusted deployment. Almost all storage, on the other hand, is left under the responsibility of the Service Providers. The difference from SQ1 is that in SQ2 there are 3 independent, non-colluding Service Providers.

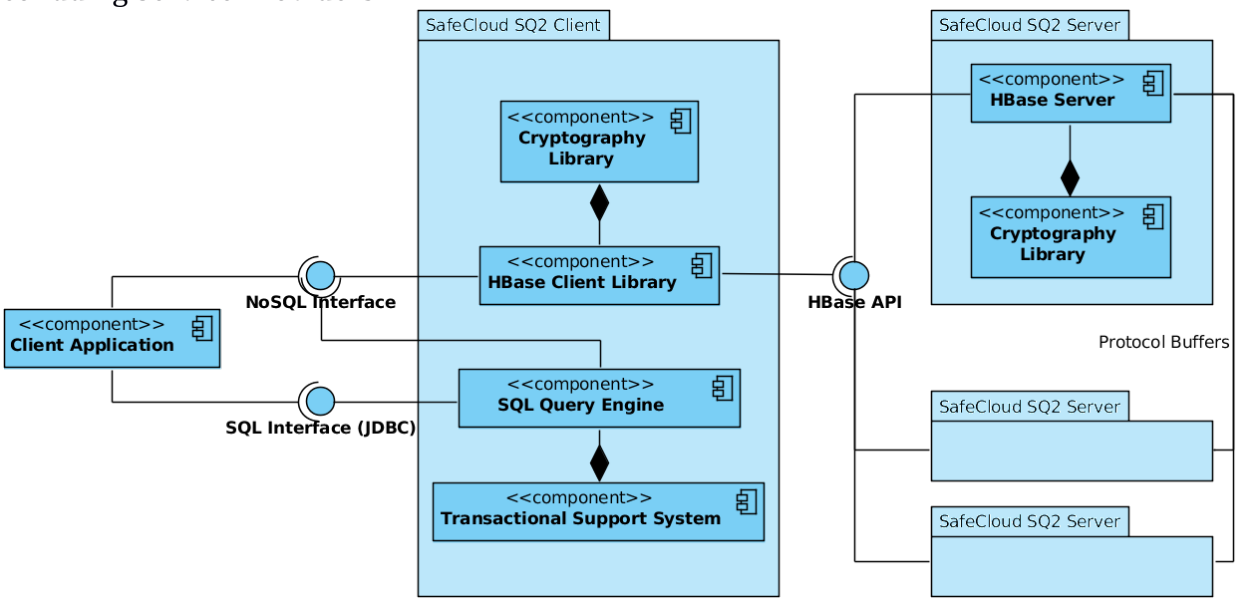

Figure 20: SafeCloud SQ2 component diagram

Figure 20 depicts the architecture for SQ2. The major architectural difference from SQ1 is that now there are three HBase instances hosted in different untrusted infrastructures.

Similar to SQ1, the Cryptography Library handles protected data in a fashion that is transparent to the end-user. This component can employ techniques described in SQ1 as well as multiparty computation protocols for data storage and processing. For the latter, the Data Owner secret shares sensitive data over three HBase remote servers, which are equipped to remotely process multiparty queries accordingly.

The SQ2 server is composed of two basic components, a regular HBase server as in SQ1 and a Multi-party Computation Library. By default, HBase cannot process secret shared data, thus requiring an extension to its core functionality. This extension is achieved through an HBase Coprocessor, an embedded framework of the HBase server described in detail in Deliverable D3.2. Briefly, this framework and a Multi-party Computation library are leveraged to enable regular HBase operations on top of encrypted data (secret shared data).

4.2.3 Security

Trust assumptions

1) The Data Owner is trusted.
2) The Service Providers are untrusted.
3) The Service Providers do not collude.

Data Owner. The assumptions about Data Owner are exactly the same as in SQ1. The Data Owner has the rights to process the data and does it in a trusted environment, assumed to be secure at all times. It is motivated to outsource storage and querying to reduce its infrastructure needs, while also under the obligation (by regulation, contracts or distrust) to keep untrusted participants from freely accessing the clear data. Thus, it does not trust the Service Provider to process it in an unprotected fashion. The SafeCloud SQ2 gives the Data Owner full access to the data, so all internal confidentiality attacks should be handled in an administrative fashion.

Service Provider. Like for SQ1, the Service Providers provide infrastructure and platform services, and allow customers to offload data and computation. These participants should be equipped to compute over encrypted data, since they expect to handle protected information.

Non-collusion. If all Service Providers collude, SQ2 would be a particular case of SQ1. Data security of SQ2 should be able to leverage a scenario where the untrusted participants are not cooperating, as this is a fundamental assumption for the usage of schemes such as Secret Sharing [BNT+12].

Solutions presented in SQ1 can be enforced against arbitrary malicious behaviour from the Service Provider. SQ2 proposes alternative protocols that can both provide stronger security guarantees and offer a wider variety of functionalities over protected data, if the Data Holder can assume the Service providers to be semi-honest and non-colluding. This is yet another possibility for deployment tradeoffs in the SafeCloud framework.

4.3 Secure processing in multiple untrusted domains with untrusted clients

This SafeCloud solution (Secure Queries 3 or SQ3) provides a secure database querying and generic secure processing capability. SQ3 considers multiple (mutually untrusted) clients, allowing for joint querying of sensitive data for aggregated results, protecting the confidentiality of individual entries.

4.3.1 Features

SQL interface. SQ3 provides a SQL interface, however it is not fully ANSI SQL compliant. For example, any filtering on less than *n* rows where *n* is a security parameter, will yield an empty result to protect the confidentiality of individual data rows.
Stored procedure support. The SQ3 SQL interface provides support for executing stored procedures. The stored procedures are implemented in the SecreC [BLR14] privacy-preserving programming language. Stored procedures offer the full capabilities for processing data, while thanks to type level privacy modifiers in SecreC, the amount of unintentional data leaks is minimized.
Low computation needed on the client side. SQ3 stores private data using additive secret sharing. All the database operations are executed on the service providers and use multi-party computation (MPC) techniques to ensure the confidentiality of data. MPC techniques have moderate overhead in terms of performance when compared to plain-text processing.
Mutually untrusted clients. In SQ1 and SQ2, a moderate amount of processing is done in the trusted deployment with the decrypted data. In SQ3, Data Owners do not need to trust the client, because data is reconstructed from secret shares only in the last step. All the intermediate steps for processing a query are done securely. Furthermore, clients of SQ3, can only make aggregating queries.

SQ3 considers the common use case where several entities want to jointly execute some function (data analytics, for instance) over highly sensitive data. If no trusted third party can be agreed upon (which is the case for many real-world scenarios), then this is a textbook example of a multiparty computation scenario. Contrary to SQ1 and SQ2, SQ3 now assumes that untrusted deployments can receive data from several Data Holders, and must therefore provide computational results that do not leak information regarding the stored sensitive information.
As an example application for SQ3, consider multiple hospitals in the same region. Hospitals have the data about their patients, but they cannot share that data because of privacy concerns. However, they are interested in aggregate queries over the data to detect region-wide epidemics. With SQ3, the hospitals can solve their data sharing problem and even deploy the system using the infrastructure of an untrusted cloud service provider.

4.3.2 Deployment and integration

Stakeholders

1) **Data Owner** - controls the data to be processed, is interested in protecting it from other Data Owners, Service Providers and Analysts.
2) **Analyst** - uses the SQL interface to query the system.
3) **Service Providers** - provide the infrastructure and platform for storing and securely querying data.

We assume that there are multiple Data owners, otherwise SQ1 or SQ2 can be used. It is perfectly valid and in some cases even preferable that one entity fulfils multiple roles. As an example, Data Owner might want to also get the benefits from sharing its data and therefore, should also take on the Analyst role.

SQ3 components

1) **SQ3 Client** - the component to be deployed by the Data Owner and Analyst, provides secure querying capability and connects to SQ3 Servers for running the queries.
2) **SQ3 Server** - the component to be deployed by the independent Service Providers, provides a secure database.

SQ3 is built on top of Sharemind[BNT+12] which is a programmable distributed secure computation framework utilizing MPC.

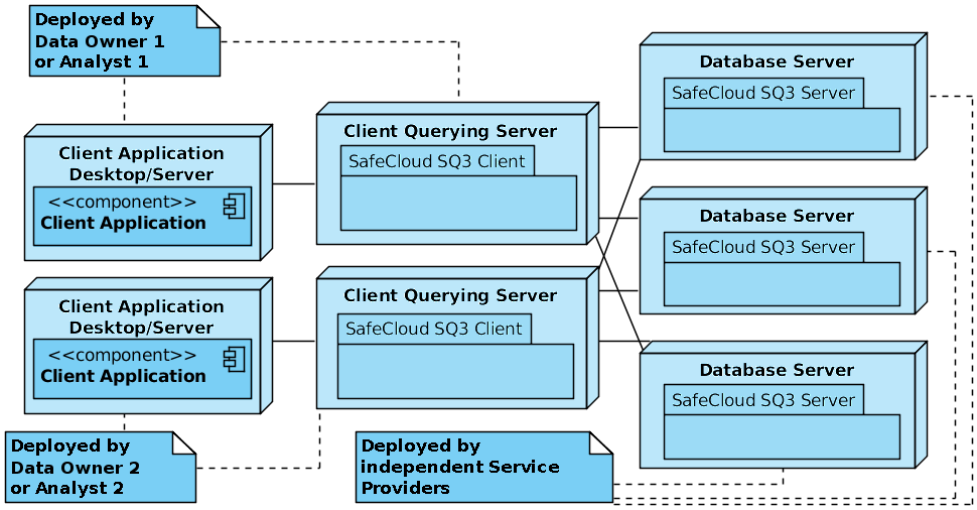

Figure 21: SafeCloud SQ3 deployment diagram

Figure 21 illustrates the relations between the stakeholders and the components they deploy. In many cases it makes sense for the Client application and the trusted domain component (SQ3 Client) to be deployed on the same machine. Like for SQ1 and SQ2, the client application can be implemented in a number of ways: it can be an *app* on an application server, a standalone application, a mobile application, etc. Note that the SQ3 Client component does very little processing and all data intensive work is left for the Service Providers.

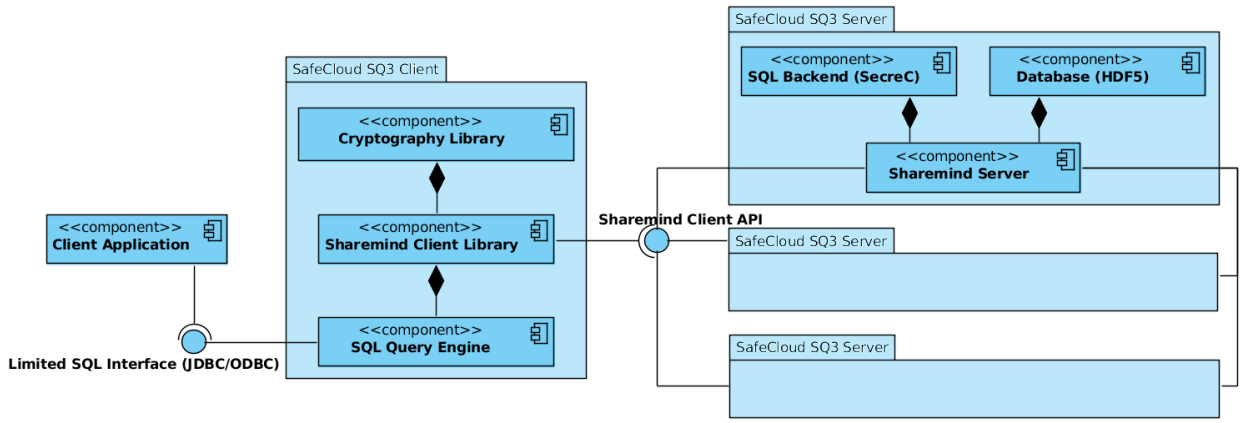

Figure 22: SafeCloud SQ3 component diagram

Figure 22 provides a more detailed description of the components of SQ3. Client application communicates with the SQ3 Client component via PostgreSQL database drivers. The SQL Query Engine offers a SQL interface, which mimics PostgreSQL server, however it will not support all the commands. SQL Query Engine handles the query and

calls procedures on the servers to handle the processing. The Sharemind Client Library handles communication with the SQ3 servers (secure channels are used). The Cryptography Library provides functionality for secret sharing data and reconstructing data from shares. The SQ3 Server consists of a Sharemind Server, a Database and a SQL Backend. The SQL Backend provides the algorithms for privacy-preserving database operations and is implemented in the SecreC language. The Database component is built on top of HDF5 and is used primarily for storage.

4.3.3 Security

Trust assumptions

1) The Data Owners are mutually untrusted.
2) The Analyst is trusted.
3) The Service Providers are untrusted.
4) The Service Providers do not collude to break privacy.

Data Owners. The Data Owners have sensitive information and are interested in performing computations with it, as well as cooperate with other Data Owners for joint computations.

Analyst. The Analyst is the stakeholder, who will get the query results in clear, therefore analyst must use a trusted computing environment. This environment is not equipped to perform meaningful amounts of computation, so it should not be considered as a potential trusted party for data storage and processing.

Service Providers. As is the case with previous solutions, the Service Providers provide infrastructure and platform services, and allow customers to offload data and computation. These participants should be equipped to compute over encrypted data, since they are expected to handle protected information.

Non-collusion. Service Providers are assumed to not collude under any circumstance, which is an essential requirement for the considered underlying mechanisms.

4.4 Comparison

Table 2 provides a broad overview of different solutions suggested for secure query processing. X shows that a feature is supported, X* shows that a feature is partially supported.

All solutions are designed to allow for encrypted storage and data processing, which is the central scope of the secure query layer of SafeCloud. Furthermore, all of these also provide both a SQL and a NoSQL interface, as described in deliverable D3.1.

SQ1 and SQ2 provide a more complete SQL interface with transactional support, which can be a requirement for specific use cases. However, these are limited to a single client (Data Holder), which narrows their focus to the access and management of single data items, and restricts their application from use cases that require joint data processing from several different sources. Alternatively, SQ3 is designed to consider several Data Holders, and allows for external participants (Analysts) to request data processing from

multiple sources, receiving aggregated results while maintaining confidentiality of single data items.

SQ1 and SQ2 allow for the usage of a wide variety of cryptographic mechanisms, where a subset of these is resistant to malicious adversarial behaviour from the Service Provider(s). As a tradeoff to employ multiparty computation algorithms of Sharemind in SQ2 and SQ3, semi-honest behaviour must be assumed from the Service Providers.

As a whole, the versatility of proposed techniques for secure queries is helpful in the context of the framework, as the architecture allows for a deployment that implements these cryptographic mechanisms in a way that is transparent to the end-user. The provided modularity is crucial to the success of the SafeCloud project towards solving a broader set of real-world problems: it allows for users of SafeCloud-enabled solutions to experience a standard well-known API that is maintained regardless of the underlying techniques, and enables the deployment of services according to fine-grained security and performance requirements.

Deliverable D3.2 provides more detailed information regarding these three solutions of SafeCloud secure queries layer, as well as individual discussions regarding applicability and relevance in different contexts.

Feature \ Solution	SQ1	SQ2	SQ3
Encrypted storage and processing	X	X	X
Transaction Support	X	X	
SQL interface	X	X	X
NoSQL interface	X	X	X
Allows for malicious adversaries	X*	X*	
Secure data processing between clients			X
Single data items can be accessed	X	X	

Table 2: Comparison of solutions for secure query processing

5 Integration and specific use cases

We apply the SafeCloud framework to the Maxdata and Cloud&Heat use cases. All the use cases are described in more detail in D5.1 and D5.2. Both project partners have multiple use cases with different requirements which we address by using the framework components described in this deliverable.

The instantiation of the SafeCloud architecture with regard to the Maxdata use cases is illustrated in Figure 23, Figure 24 and Figure 25. In use case 1 and 2, the application uses the SafeCloud APIs to either store the data directly on the distributed secure filesystem residing in the untrusted domain or to store the data through the secure SQL engine. The latter option also provides the functionality to process queries on the securely stored data. The SQL engine uses a NoSQL database backend. The NoSQL database can either reside directly on the local storage, or run on top of the secure filesystem or block storage. In the trusted domain, the local storage is required for caching purposes. Use case 3 is similar to the previous scenarios with the following differences: the NoSQL database is replaced with the distributed Sharemind engine, and the SQL engine in the trusted environment no longer requires a local storage. For each use case, the communication layer components of the framework are used to provide security in trusted-untrusted and untrusted-untrusted domain communications.

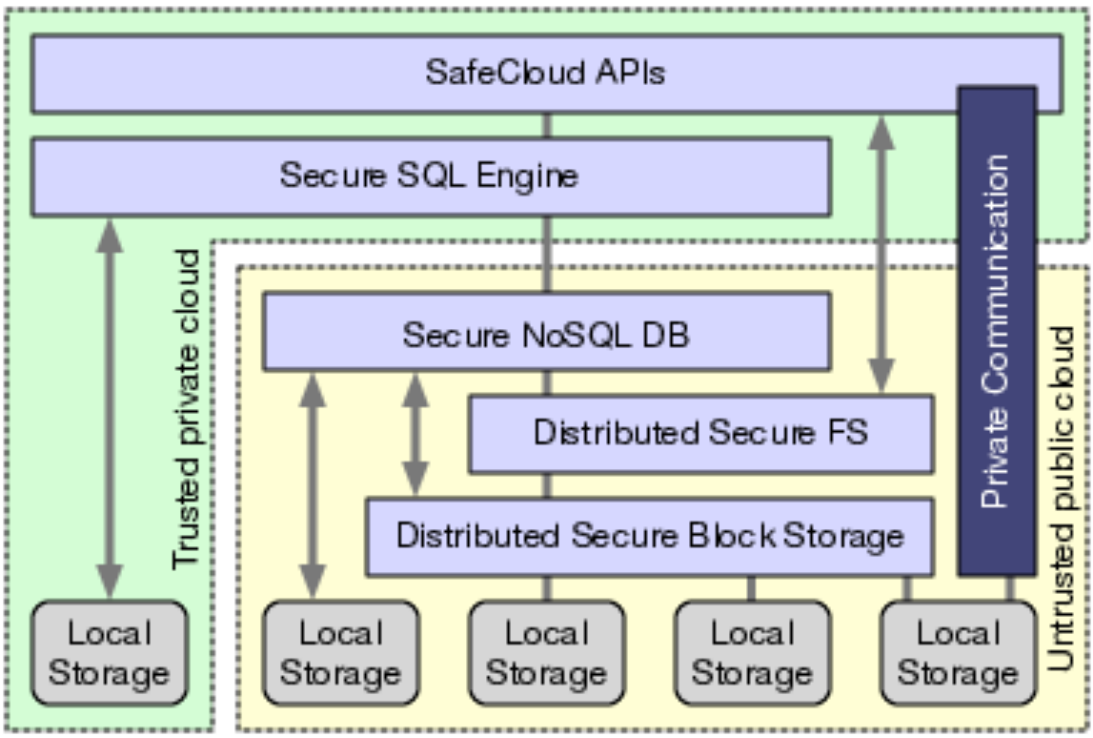

Figure 23: Maxdata use case 1

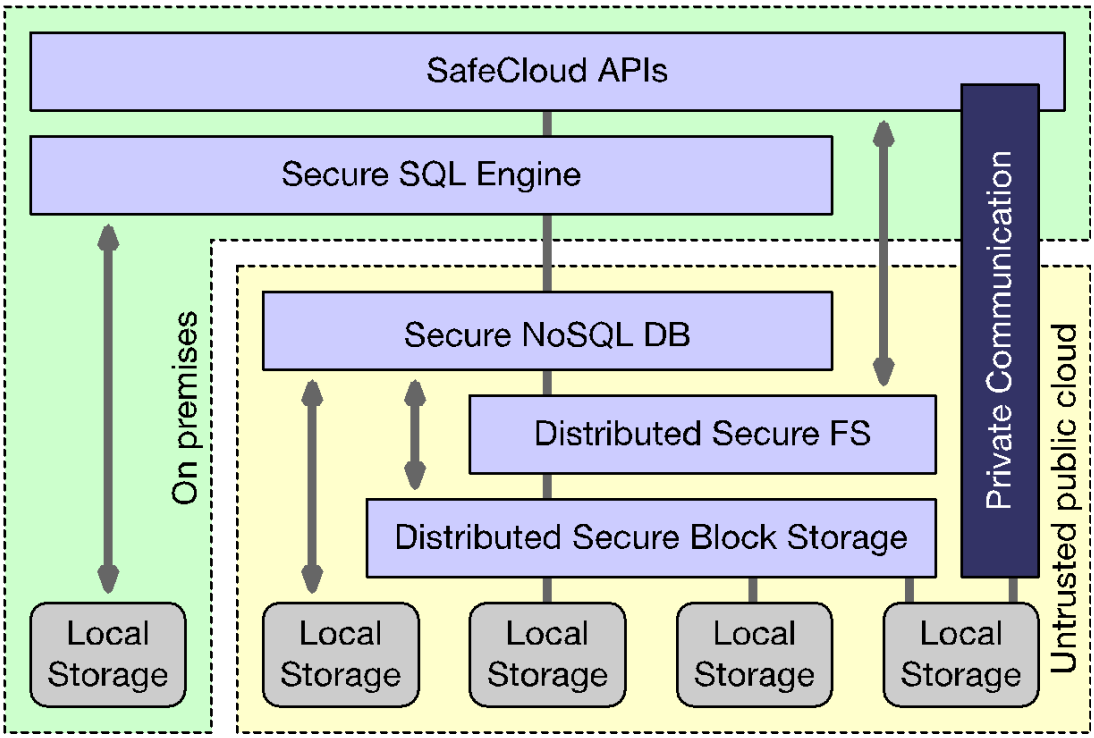

Figure 24: Maxdata use case 2

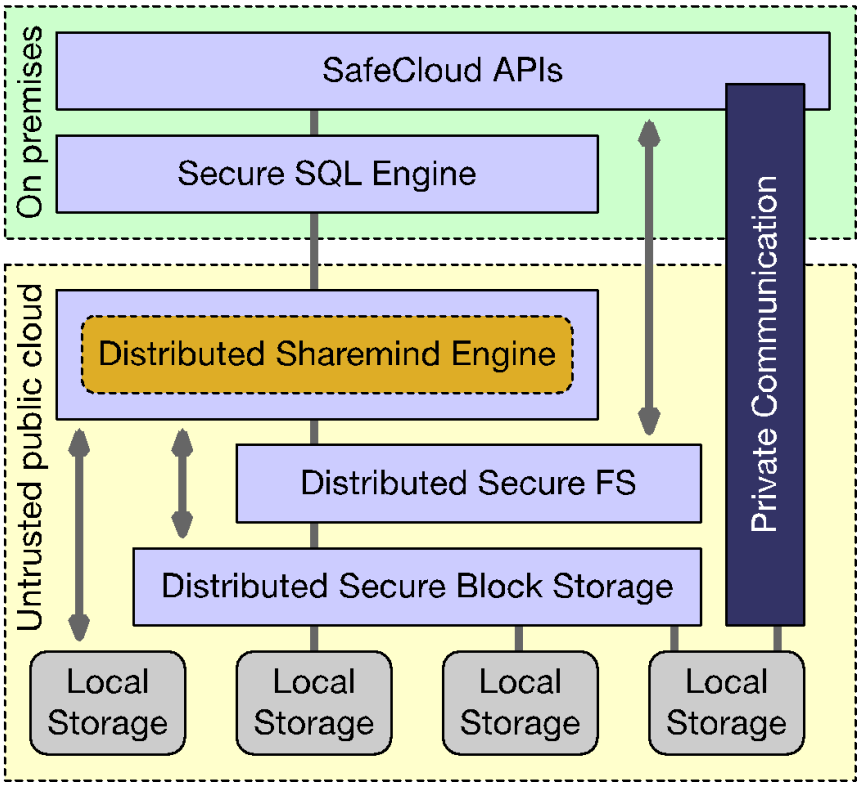

Figure 25: Maxdata use case 3

The Cloud&Heat architecture for its two use cases is respectively shown in Figure 26 and Figure 27. Both use cases take advantage of the distributed nature of the Cloud&Heat micro-clouds. In use case 1 we deploy the distributed secure block storage on the data centers. The block storage can then be used by the virtual machines provisioned by Cloud&Heat. This will allow to provide reliability and provisioning over

multiple data centers. In use case 2 we deploy an additional file system layer on top of the secure block storage and leverage the properties of the long-term data store.

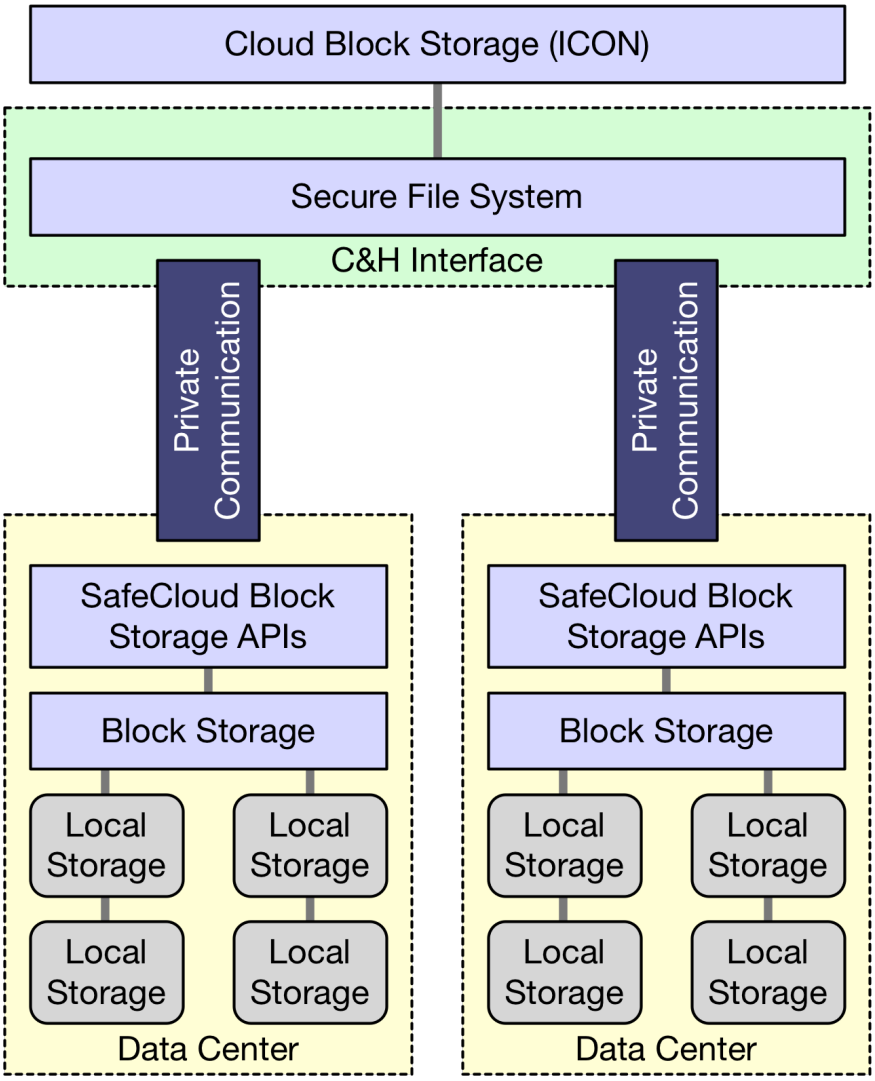

Figure 26: Cloud&Heat use case 1

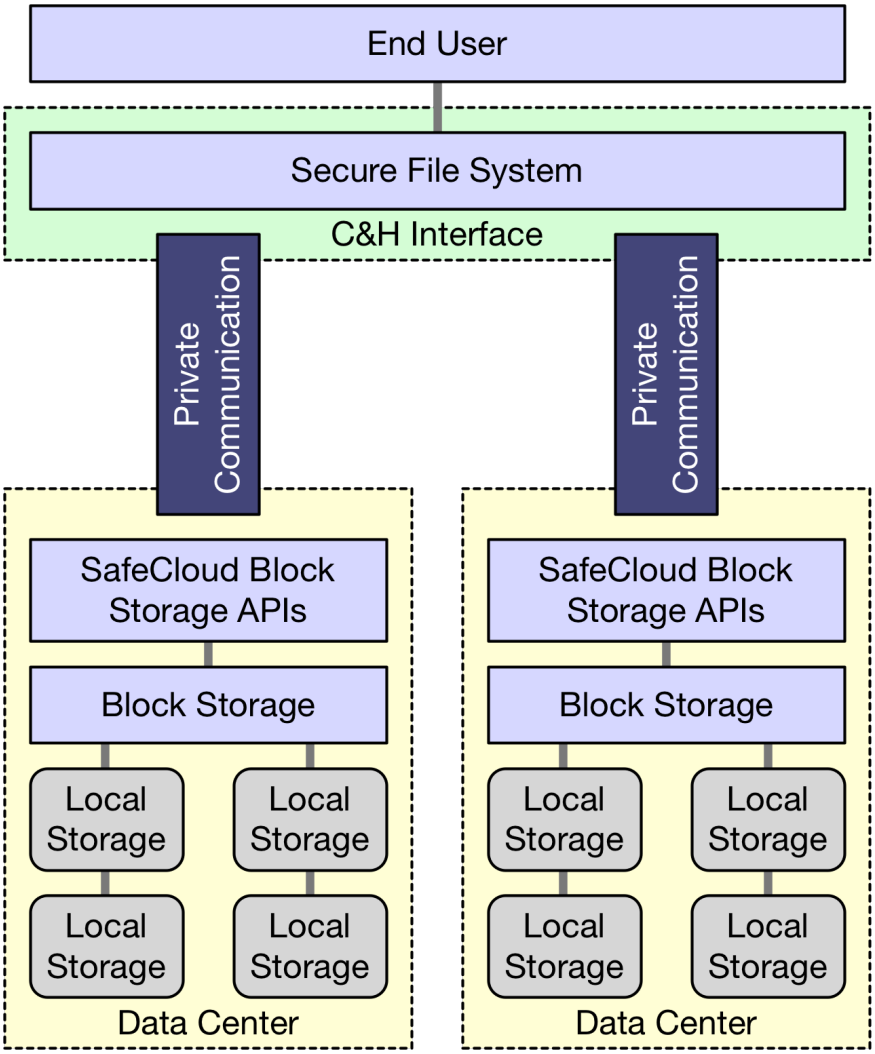

Figure 27: Cloud&Heat use case 2

6 Conclusion

The SafeCloud architecture consists of three layers: secure communication, secure storage and secure queries. Each of these layers contains multiple solutions that address various problems in those areas.

In the communication layer the solutions improve over existing solutions by using techniques such as redundant encryption, port knocking, route monitoring and multi-path routing. These solutions can be deployed in distributed systems, such as the cloud, to archive better security guarantees for the communication, where much of the traffic travels over the public internet. The storage layer uses encryption, geographic distribution and entanglement to provide storage solutions that are well suited for cloud environments, where a client wants to ensure its data is kept private and is not tampered with. The queries layer provides different secure SQL and NoSQL database solutions. These databases allow to outsource the storage of an application while keeping the data itself private.

The described solutions are deployed to solve real-life problems in the Maxdata and Cloud&Heat uses cases. We show concrete examples how the solutions are integrated into existing applications and infrastructure.

7 References

[AJ05] Aycock, John, and M. Jacobson. "Improved port knocking with strong authentication." 21st Annual Computer Security Applications Conference (ACSAC'05). IEEE, 2005.

[AKS+04] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563-574. ACM, 2004.

[BAC+08] Bessani, Alysson Neves, et al. "DepSpace: a Byzantine fault-tolerant coordination service." ACM SIGOPS Operating Systems Review. Vol. 42. No. 4. ACM, 2008.

[BBO07] Mihir Bellare, Alexandra Boldyreva and Adam O'Neill. Deterministic and efficiently searchable encryption. *Annual International Cryptology Conference*. Springer Berlin Heidelberg, 2007.

[BKR13] Mihir Bellare, Sriram Keelveedhi and Thomas Ristenpart. "Message-locked encryption and secure deduplication." *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer Berlin Heidelberg, 2013.

[BLR14] Bogdanov, Dan, Peeter Laud, and Jaak Randmets. "Domain-polymorphic programming of privacy-preserving applications." Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security. ACM, 2014.

[BNT+12] Bogdanov, Dan, et al. "High-performance secure multi-party computation for data mining applications." International Journal of Information Security 11.6 (2012): 403-418.

[D+12] Ivan Damgård et al. "Multiparty computation from somewhat homomorphic encryption." *Advances in Cryptology–CRYPTO 2012*. Springer Berlin Heidelberg, 2012. 643-662.

[K03] Krzywinski, Martin. "Port knocking from the inside out." SysAdmin Magazine 12.6 (2003): 12-17.

[KG14] Kirsch, Julian, and C. Grothoff. "Knock: Practical and Secure Stealthy Servers." (2014).

[LTV12] Adriana López-Alt, Eran Tromer and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. ACM, 2012.

[NKW15] Naveed, Muhammad, Seny Kamara, and Charles V. Wright. "Inference attacks on property-preserving encrypted databases." Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015.

[PBP16] Poddar, Rishabh, Tobias Boelter, and Raluca Ada Popa. "Arx: A Strongly Encrypted Database System." .

[PRZ12] Popa, Raluca Ada, et al. "CryptDB: processing queries on an encrypted database." Communications of the ACM 55.9 (2012): 103-111.

[PS03] Padmanabhan, Venkata N., and Daniel R. Simon. "Secure traceroute to detect faulty or malicious routing." ACM SIGCOMM Computer Communication Review 33.1 (2003): 77-82.

[VHT09] Vasserman, Eugene Y., Nicholas Hopper, and James Tyra. "SilentKnock: practical, provably undetectable authentication." International Journal of Information Security 8.2 (2009): 121-135.

[WKC+14] W. K. Wong, B. Kao, D. W. L. Cheung, R. Li and S. M. Yiu. Secure query processing with data interoperability in a cloud database environment. Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '14, 2014, pp. 1395–1406.

[ZJP07] Zheng, Changxi, et al. "A light-weight distributed scheme for detecting IP prefix hijacks in real-time." ACM SIGCOMM Computer Communication Review. Vol. 37. No. 4. ACM, 2007.

[ZZH+08] Zhang, Zheng, et al. "Ispy: detecting ip prefix hijacking on my own." ACM SIGCOMM Computer Communication Review. Vol. 38. No. 4. ACM, 2008.