SafeCloud

Privacy-preserving storage and computation techniques
D3.2

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	01.09.2016
Version	1.0
Responsible Partner	INESC TEC

Dissemination level

Public

Revision history

Date	Editor	Status	Version	Changes
05.07.2016	Francisco Maia	Draft	0.1	ToC
24.07.2016	Bernardo Portela	Draft	0.2	Complete Draft
08.08.2016	Hugues Mercier	Revision	0.3	UniNE Revision
08.08.2016	Miguel Pardal	Revision	0.4	INESC-ID Revision
20.08.2016	João Paulo	Revised Version	0.8	Revised version
22.08.2016	Bernardo Portela	Final Version	1.0	Submission-ready version

Contributors

Bernardo Portela (INESC TEC)
João Paulo (INESC TEC)
Francisco Maia (INESC TEC)
Rogério Pontes (INESC TEC)
Tiago Oliveira (INESC TEC)
Reimo Rebane (CYBER)
Karl Tarbe (CYBER)

Internal reviewers

Hugues Mercier (UniNE)
Miguel Pardal (INESC-ID)

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
CE	Convergent Encryption
DML	Data Manipulation Language
HCE	Hash and Convergent Encryption
HGD	Hypergeometric Distribution
IND-CCA	Indistinguishability under Chosen Ciphertext Attack
IND-CPA	Indistinguishability under Chosen Plaintext Attack
INT-CTXT	Integrity of Ciphertext
IV	Initialization Vector
JDBC	Java Database Connectivity
MLE	Message-Locked Encryption
mOPE	Mutable Order Preserving Encryption
MPC	Multiparty Computation
OPE	Order-preserving Encryption
POPF-CCA	Pseudorandom Order-preserving Function under Chosen Ciphertext Attack
RCE	Random Convergent Encryption
TTP	Trusted Third Party
TD	Trusted Deployment
UD	Untrusted Deployment

Table of contents

List of Figures

List of Tables

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 data processing requests. The overarching idea is to allow system developers to combine implementations by employing these three layers to achieve application-specific deployments that 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.

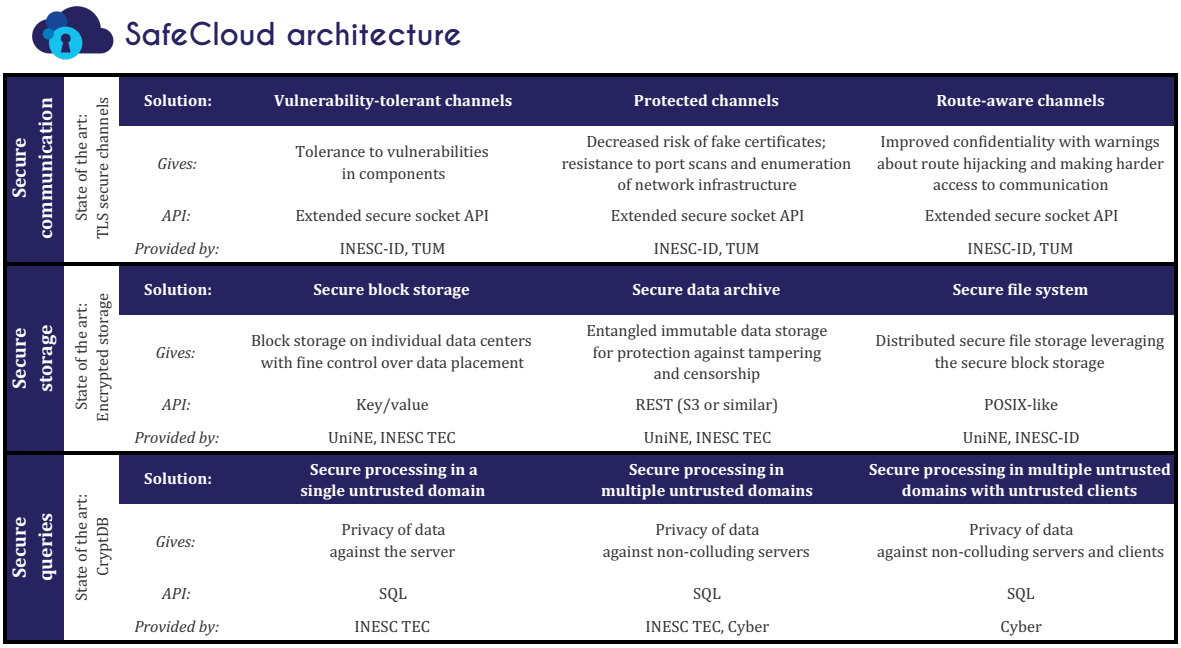

Figure 1: The SafeCloud framework.

This document has been produced in the context of work developed during work package 3 (WP3), dedicated to the development of mechanisms for ensuring privacy-preserving data processing. The central focus in this regard is to procure the balance between security assurances and acceptable levels of functionality and performance with respect to application-specific requirements. Deliverable D3.1 has been delivered in month 6, describing the general architecture for SafeCloud and its individual components, as well as several solutions for private data protection in SQL query execution.

Deliverable D3.2 is the second document of WP3, and succeeds the previous work of D3.1. Given a complete architecture, component and scenario description, we now provide a thorough overview of security models for reasoning over the security of our protocols, and present an in-depth evaluation of available privacy-preserving techniques for data processing, as well as an existing state-of-the-art implementation for these techniques. Furthermore, we show how the proposed architecture can instrument a wide variety of privacy-preserving cryptographic methods (and how it can combine

them), towards validating the SafeCloud framework feasibility with respect to the specified use cases.

The document is structured as follows. Section 1 will provide a recap of the previous deliverable proposing the architecture and its designated components, as well as set the general conceptions for security and adversarial behaviour used to describe application-specific security requirements. Sections 2, 3 and 4 map Solutions 1, 2 and 3, respectively. For each Solution, we present the considered techniques and compatible state-of-the-art implementations, detail how these can be instantiated given the available components of SafeCloud architecture, and conclude with an overall discussion regarding the relevance and feasibility scope. Finally, Section 5 closes the document with an overview of the solutions made available by SafeCloud, and how the approach can lead to meaningful contributions towards higher-assurance cloud deployments.

1 Privacy-preserving storage and computation

1.1 Architecture

The development and maintenance of IT infrastructures are usually associated with very high-costs. This has been a major contributor to the success of the cloud computation paradigm, where a very considerable part (or all, in some instances) of these IT concerns can be offloaded to remote providers. While this is a meaningful advantage with respect to cost reduction, having potentially sensitive data stored in - and manipulated by - untrusted environments must be accounted for. Indeed, the lack of secure trustworthy solutions for data storage and computation in Cloud environments typically demand specific deployment compromises, which in turn lead to suboptimal systems. Within this context, modern application and service deployments can fall into one of three major categories:

- In-house deployments, where everything from IT infrastructure to software is designed, developed and maintained within the company or organization facilities. This is usually the case for companies managing and computing over data that is too sensitive to be offloaded to standard Cloud solutions.
- Everything is handed over to a Cloud or a set of Cloud providers, and the organization/company manages its infrastructure remotely. This is possible when the company does not handle sensitive data, or the security requirements are low enough to be enforced by a somewhat trustworthy cloud environment (e.g. one that provides basic access control measures).
- A compromise between the previous two, in which critical systems and services remain within trusted premises, and complementary services are offloaded to convenient Cloud providers. This aims to be the "best of two worlds" solution, which allows for extracting the maximum potential of the Cloud provider without jeopardizing information security, but requiring a case-specific analysis of security and functional requirements.

The general architecture of SafeCloud solutions for privacy-preserving storage and computation is heavily rooted in the Cloud Computing paradigm - focused on exploring the potential of this third category - comprising a set of comprehensive solutions to perform private and secure data storage and computation.

Recall Figure 2 depicting the SafeCloud general architecture for privacy-preserving data storage and processing at a high level. We make use of *squares* to represent processing components, *rounded edge squares* to represent interfaces, and *cylinders* to represent storage components. In this context, we will refer to the set of components running on-premises as *trusted deployment*, and to the components executing on third-party systems as *untrusted deployment*.

The trusted deployment infrastructure is assumed to be completely trusted, and can therefore receive, handle and process sensitive data without requiring cryptographic techniques. On the other hand, untrusted Cloud infrastructure refers to providers that may display adversarial behaviour. Typically, a proxy will be required for integrating local and remote deployments, to be located on trusted premises. Communication is made over an untrusted network where adversaries can actively eavesdrop and tamper with transmitted packages. However, we assume the availability of secure and authenticated channels to protect communication from external malicious behaviour.

This can be provided by the SafeCloud private communication middleware, developed in SafeCloud WP1.

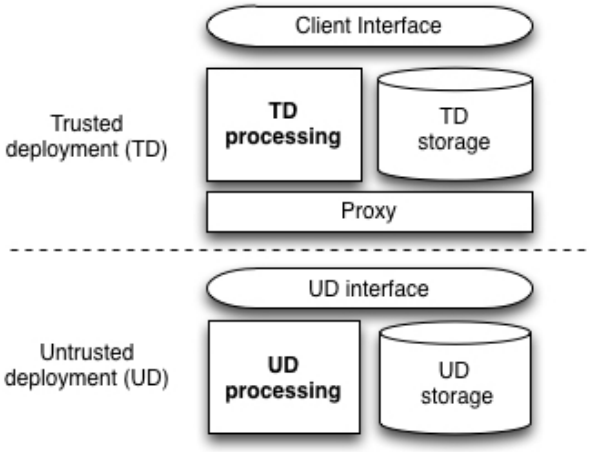

Figure 2: High-level SafeCloud architecture for privacy-preserving data storage and processing.

The provided architecture allows the accomplishment of several solutions, describing different deployment scenarios that are compatible with a variety of use cases. We now briefly overview the three solutions, as they will be later detailed regarding associated privacy-preserving techniques and deployment considerations.

- Solution 1 considers a **single trusted client**, offloading data and computations to a **single untrusted deployment**, potentially a cloud provider. The usage of cryptographic techniques in this setting depends on the type of adversarial corruption considered remotely, and on the functionalities required for the use case, i.e. what computations the system expects the untrusted deployment to perform.
- Solution 2 also considers a **single trusted client**, but now the required remote efforts can be distributed amongst **multiple untrusted deployments**. In addition to previous security considerations, now cryptographic schemes can leverage scenarios in which only a subset of untrusted entities can be corrupted for providing more efficient/secure data processing implementations.
- Solution 3 is a slight variation of Solution 2, in which **several clients** can interact with a set of **untrusted deployments**, inputting and querying over data. Security concerns must now account for providing data analysis algorithms returning aggregated data from multiple clients without disclosing sensitive data from any individual local deployment.

A significant factor in validating the success of the SafeCloud project is its practical applicability. Accordingly, it is very important to provide interfaces that accommodate current practices, to enable compatibility with the provided solutions. The current landscape of data management systems allows for the highlight of two large groups with respect to interface: query language-based systems (SQL) and NoSQL databases. The solutions described in this document will provide the end-user with either a SQL interface with transactional support or a NoSQL interface. However, in order to meet certain performance levels without sacrificing security guarantees, the actual language coverage of each respective solution will be individually specified.

1.2 Security model

Deliverable D3.1 proposed several implementation scenarios associated with SafeCloud's solutions. These scenarios differ with respect to computational and trust assumptions placed on the participants involved in the protocols. Solution 1 assumes an environment with a single local client, which we establish to be trusted, and a single remote untrusted cloud provider, which we assume to be untrusted. Solution 2 allows the deployment of distributed trust protocols, as it assumes multiple non-colluding untrusted cloud providers. Solution 3 enables different clients to upload/query the remotely stored data, by allowing untrusted clients to request computations over sensitive information (e.g. executing data analysis over a dataset shared amongst companies).

Considering different deployment scenarios allows for leveraging fundamentally different cryptographic techniques, which are suitable for a variety of real-world scenarios. This is a relevant factor because it not only complies with the security and performance requirements of the SafeCloud's use cases, but also extends the SafeCloud framework towards broader practical applicability.

Regardless the actual deployment scenario, the feasibility and application of each cryptographic scheme ultimately depends on the considered security model for such scenario. The security model characterizes the considered adversarial power (i.e. what we assume an adversary is capable of doing, or which participants it is able to corrupt), which allows us to objectively describe under what circumstances the different SafeCloud solutions can be considered secure. In this regard, we assume the adversary to be a monolithic entity that may be corrupting multiple participants and eavesdropping on several communication channels simultaneously, which is a simpler and stronger model than assuming multiple adversaries that may or may not act in accordance to each other. We now present models for trust, adversary power and corruption types that are used to determine adversarial behaviour.

Participants can be either *trusted* or *untrusted*, which determines the participants that the adversary can influence. This differentiation is already explicit in SafeCloud's architecture, where the deployment is divided between a *trusted deployment* and an *untrusted deployment*, but this notion can also be applied to additional participants. For instance, if we want to employ SafeCloud's framework to implement a protocol that considers public verification of outputs, the instantiation may depend on the trust that can be placed in said remote audit.

Adversary power formalizes what capabilities we assume from the adversary. In this regard, we follow the definitions proposed by Aumann and Lindell [AL07], and distinguish three models that can be used to describe the adversary:

- *Active* adversaries can behave arbitrarily, regardless of what the protocol dictates. This means a party corrupted by an active adversary can provide wrong inputs, tamper with messages, or even stop responding outright.
- *Covert* adversaries are behaviourally similar to active adversaries, but they do not wish to get "caught" in doing so. Essentially, these adversaries behave arbitrarily, as long as the possibility for the system to identify them as an adversary does not exceed some previously defined threshold.

- *Semi-honest* adversaries (also known as honest-but-curious) follow the specified protocol, but may attempt to learn additional information by analysing the local transcript for messages exchanged during the computation.

Real-world scenarios of malicious entities gaining access to protocol participants are modelled as *corruptions*. The corruption model defines how the system expects these corruptions to occur. Corruptions can be performed over a subset of *untrusted* participants up to a predetermined threshold, and they may be *static* or *adaptive* [DN14]:

- *Static* corruption means that the corrupt participants are predetermined before starting the protocol, and remain unchanged for its full duration.
- *Adaptive* corruption is strictly stronger than static corruption, as the participants may be corrupted at any time during the protocol execution, based on the information the adversary gathered thus far.

Furthermore, we also characterize the type of corruptions to be either *snapshot* or *persistent* [PBP16]:

- *Snapshot* corruption models an adversary that briefly accesses the system, by giving the adversary access to all information held by the corrupt party at the moment of corruption.
- *Persistent* corruption refers to the standard corruption definition, where the adversary gains full control of that participant from that moment onwards (what it can do is bound by the considered adversary power), as well as access to the local transcript.

Following the overall approach of WP3, we now describe the three SafeCloud solutions for privacy-preserving data storage and processing. For each of these solutions, we propose several privacy-preserving techniques, and contextualize with the security model to which they are applicable. We will also instantiate the components of the general architecture to demonstrate how the proposed cryptographic techniques can be implemented, and how the interfaces provided in deployment can be instrumented accordingly.

2 Solution 1: Secure processing in a single untrusted domain

2.1 Privacy-preserving techniques

Solution 1 (Figure 3) depicts the classic cloud computation scenario, in which a local trusted domain wants to offload data and computation to a potentially untrusted cloud environment. This approach assumes that SQL query processing and transactional support can be performed entirely under trusted deployment, where NoSQL provides the necessary functionalities to support an ANSI SQL database system. The system proxy is the component responsible for executing the cryptographic techniques, and it will be executed in a separate layer from the SQL engine, thus benefitting from the provided abstraction to execute according to simple NoSQL commands such as *Put*, *Get* or *Scan*.

This setting allows a plethora of cryptographic mechanisms to be employed, providing varying levels of security. We now present several approaches for privacy-preserving techniques compatible with this setting, and detail the security parameters in which their application is feasible.

2.1.1 Authenticated Encryption

When storing sensitive data, the guarantees most often required from protection mechanisms (i.e. employed cryptographic schemes) are confidentiality and integrity. At a high level, confidentiality means that the protected data leaks no information regarding its original value, while integrity means that any tampering of the stored information can be identified with overwhelming probability. The most standard way of achieving this is via the usage of authenticated encryption schemes.

An authenticated encryption scheme is defined by a triple of algorithms (Gen, Enc, Dec):

- $Gen(1^n)$ takes a security parameter[1] and generates a symmetric key k.
- $Enc(k, m)$ takes a key and a message and produces a ciphertext c.
- $Dec(k, c)$ takes a key and a ciphertext and produces a message m.

The feasibility of applying these schemes in practice hinges on the fulfilment of correctness and security properties. Correctness is a general concept for encryption schemes, and describes the intuitive notion that a decryption after encryption produces the original message, i.e.

$$Dec\big(k, Enc(k, m)\big) = m$$

The security of these schemes is often analysed with respect to meeting the requirements of semantic security [GM82], which states that an adversary should not be able to infer any information about the original value from the associated ciphertext, given a specific context. The formalization of security for authenticated encryption schemes, in particular, can be analysed by considering indistinguishability under chosen-plaintext attacks (IND-CPA) and ciphertext integrity (INT-CTXT), which together imply security for Authenticated Encryption (a strictly stronger notion than the standard IND-CCA, indistinguishability against chosen-ciphertext attacks).

[1] The security parameter is standardly expressed via unary representation. In this case, n represents a string of n 1s.

These schemes are used exclusively for ensuring security of information, since they exclude any possibility of computation over ciphertexts. This actively disables the possibility for an untrusted storage entity (the untrusted deployment, according to SafeCloud architecture) to perform computation over the stored data, and thus prevents leveraging the potentially higher computation capabilities associated with it.

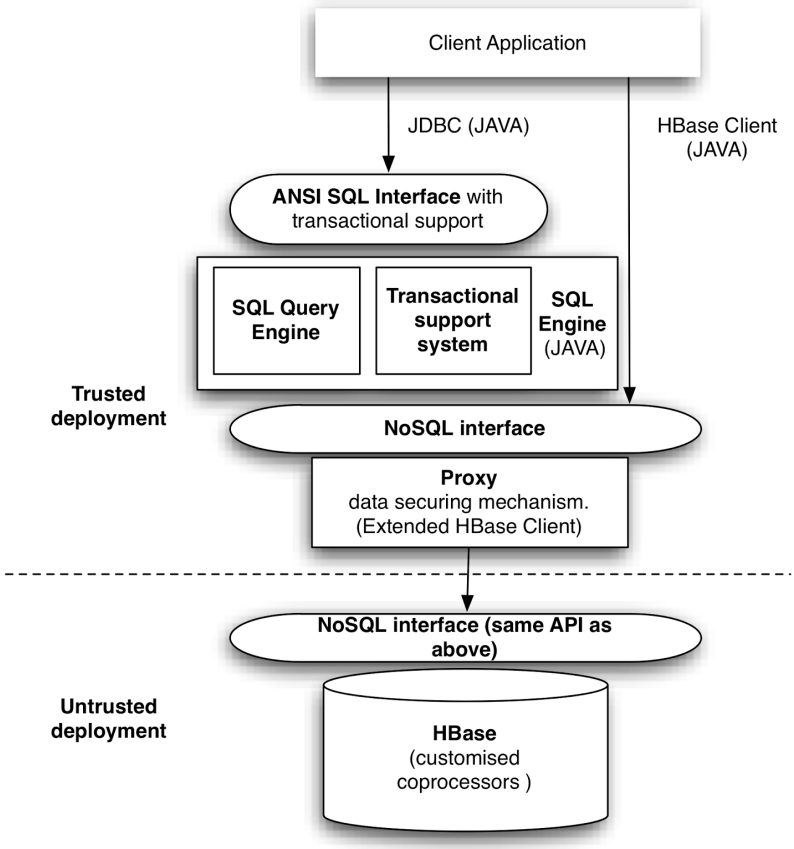

Figure 3: Solution 2 architecture: single untrusted domain.

2.1.2 State-of-the-art Implementations of Authenticated Encryption

Following the ENISA (European Union Agency for Network and Information Security)[2] algorithms, key size and parameters report, several mechanisms for the implementation of authenticated encryption schemes are proposed. We now provide some examples described at a high level to serve as baseline.

Encrypt-then-MAC with CBC mode and CBC-MAC as the message authentication code is a two-pass process whose analysis in [BN00] provides very positive feedback for practical usage. Other related constructions such as Encrypt-and-MAC or MAC-then-Encrypt are generally not advised to be put into practice, as real world attacks have been implemented in systems using these variants. More information about this can be found in the referenced report.

[2] https://www.enisa.europa.eu/

Assuming the underlying block cipher is secure, OCB mode is a highly efficient one-pass mode of operation proposed by Rogaway [RBB03], whose wide adoption in practice has been mostly hindered by two U.S. patents. However, in January 2013, the author stated that OCB mode is free for software usage under a GNU General Public License, and for other non-open source software under a non-military license [R13].

Galois Counter Mode (GCM) [M05] has been developed as an improvement to CWC mode [KVW04], as a highly parallelizable online alternative to similar mechanisms for authenticated encryption. GCM has been shown to be provably secure, assuming that the IV is never used more than once for a given key and that the underlying block cipher is secure. Attacks on specific weaknesses of deployments can be found in the referenced report. One major advantage of this approach is its design, which facilitates efficient implementations in hardware.

2.1.3 Deterministic Encryption

Deterministic encryption is the concept of a cryptosystem that always produces the same ciphertext for the same key plaintext pair. Loosely speaking, this means that by encrypting the same value multiple times it will result in the same ciphertext, which can then be verified by anyone in possession of those ciphertexts. Similar to what was previously presented, deterministic encryption schemes are also characterized as a triple of algorithms (Gen, Enc, Dec) following the same broad descriptions.

Employing deterministic encryption schemes also provides specific correctness and security properties. Correctness, in addition to the previously mentioned definition where a decryption after encryption produces the original message, must ensure that two ciphertexts are the same if, and only if, the associated plaintexts are the same, i.e.

$$Enc(k, m_1) = Enc(k, m_2) \Rightarrow m_1 = m_2$$

Semantic security under chosen-plaintext/chosen-ciphertext attacks can only be achieved for randomized encryption algorithms. Consequently, the security of these schemes must follow definitional models in which the adversarial power to query values is limited, as proposed in [BBO07]. Essentially, employing deterministic encryption entails forfeiting security against an adversary that can correlate ciphertexts and gain meaningful information from perceiving which messages are equal (albeit not necessarily recovering the associated plaintext), or the frequency in which they are stored.

Alternatively, this also leads to very efficient solutions for so-called searchable encryption. If the untrusted deployment can trivially compare ciphertexts to confirm equality, then the client may have the remote provider search for specific values. For a setting where the client has information remotely stored, using deterministic encryption is the difference between simply encrypting an identifier search value and sending it to the server for comparison, and having to request the full database, decrypt it, and search it locally. When considering real-world applications, which often require the remote storage to hold a significant amount of information, the latter option becomes infeasible.

Additionally, techniques such as Convergent Encryption or, more generally, Message-Locked Encryption (MLE) are often used for the identification and removal of

duplicate files, also known as secure deduplication. These schemes derive keys from the plaintexts themselves, which are then used to produce the ciphertext, whereas general deterministic encryption generates keys independently of plaintext, and therefore requiring equal keys for correct comparisons. Formally, this approach considers an additional algorithm $Tag(c)$, which receives a ciphertext and provides a tag that can be used to compare values. This means that the encryption of two equal messages will always produce a result that is comparable using the Tag algorithm, i.e.

$$m_1 = m_2 \Leftrightarrow Tag(Enc(k_1, m_1)) = Tag(Enc(k_2, m_2))$$

The idea of using this subset of cryptographic mechanisms allows a remote cloud provider serving multiple users, such as Dropbox, for example, not to have to store duplicate information uploaded by different users (locally and separately encrypting their data). Suppose Alice encrypts m_1 into c_1, and sends it to the server; now suppose Bob encrypts m_2 into c_2, and sends it to the server; the server is aware if $m_1 = m_2$ by checking if $Tag(c_1) = Tag(c_2)$, and can actively choose not to store the duplicate c_2 into the database. Whenever Bob requests the supposedly uploaded c_2, the server can simply provide c_1, which it knows to contain the same information.

The usage of MLE schemes leads to more efficient storage management by the server, but it can also expose vulnerabilities, especially if the cloud side does not require the user to upload his file when it already exists remotely. Suppose that Alice suspects that Bob is in possession (i.e. has uploaded the value to the server) of a particular file f that is unlikely to be uploaded by anyone else (a payroll with predictable structure and Bob's name on it, for instance). Alice can create f locally, encrypt it by using MLE and transmit it to the server. If the server does not request the upload of the file, then Alice has confirmation that the file is already remotely stored, and thus can conclude with high probability that it is Bob's. Alternatively, if the file follows a predictable structure and has sensitive information that is not entirely predictable (Bob's payroll with a value that Alice is unsure of, but curious about), Alice can create files $f_1, \dots, f_n$ containing likely payment values and upload them sequentially. If the server does not request the upload for some particular f_i, then Alice has successfully guessed Bob's specific payment file.

2.1.4 State-of-the-art Implementations of Deterministic Encryption

The component that confers probabilistic nature to the standard symmetric encryption schemes is the initialization vector. As such, secure block ciphers running in ECB mode (which does not use IV) or in other modes with the IV set to a constant value can be implemented as deterministic encryption schemes.

Alternatively, [BBO07] also introduces two searchable encryption schemes with particular focus on performance, considering a public-key scenario (*Gen* provides a public and a private key, *Enc* is run with the public key, and *Dec* is run with the secret key):

- Encrypt-with-Hash makes use of a standard IND-CPA encryption scheme and a hash function over the plaintext and public key to produce the randomness coins used in the encryption scheme, and therefore producing equal ciphertexts for equal plaintexts.

- RSA-DOAEP is a length-preserving deterministic version of RSA-OAEP, where the randomness employed for encrypting the message is also extracted from the message itself by applying hash functions to the plaintext.

Several techniques for Convergent Encryption have been employed in practical deployments for a broad range of applications [M+08, DR+02, AZ10, CTP04]. However, the general primitive of MLE [BKR13] has contextualized and proposed further mechanisms enabling this particular type of data protection.

Alongside a thorough security and performance analysis of these implementations, and considering the availability of an encryption scheme and a hash function, techniques proposed in [BKR13] encompass:

- Convergent Encryption (CE). This is the most straightforward approach, where the key is derived by simply performing a hash over the plaintext. The tag is the ciphertext itself.
- Hash-and-Convergent Encryption (HCE). This is a slight variation of CE where the key is derived in a similar fashion, but the encrypted ciphertext is a concatenation of the original ciphertext with the hash of the derived key, allowing more parallelizable implementations.
- Random Convergent Encryption (RCE). It allows the generation of key, message encryption and tag production to be done in a single pass. It is the construction that is parallelizable to a higher degree of granularity. This is achieved by selecting a random encryption key, encrypting the message with it and providing a XOR between the derived key and the randomly generated key.

2.1.5 Order-preserving Encryption

Range queries are a very common operation in database manipulation. For instance, a client might request to the server all records that have a certain numeric attribute that is greater than some number x. It is not possible to simply encrypt the numerical attribute by using traditional encryption schemes because these do not preserve any property from the original data. Furthermore, deterministic encryption or MLE schemes allow comparisons, but not order-related queries.

Order-preserving Encryption (OPE), is an encryption scheme that preserves the numerical order of the plaintexts in the corresponding ciphertexts. It was proposed by Agrawal et al. in 2004 [AKS+04] and aimed to provide a solution for the problem of performing efficient range queries over encrypted data. In these schemes, for any m_1 and m_2, the encryption of m_1 will only be greater than the encryption of m_2 if, and only if, m_1 is greater than m_2, i.e.

$$m_1 > m_2 \iff Enc(k, m_1) > Enc(k, m_2)$$

This flavour of encryption is relatively recent and non-standardized, and as such several different schemes were proposed in the last few years, as presented in Table 1, from [PLZ13]. Only a single work claims to leak nothing besides relative order [PLZ13]. Many of them, however, do not present a study or a formal proof addressing the issue of quantifying plaintext bits leaked by the technique.

A popular approach for modelling security of OPE schemes was proposed by [BCL+09] as the POPF-CCA (Pseudorandom Order-preserving Function under Chosen Ciphertext Attack), defining the pseudorandom order-preserving function advantage under chosen-ciphertext attacks. A security analysis of an OPE scheme under this definition, and further details regarding caveats and other insights with respect to general security discussion of OPE can also be found in the same reference.

2.1.6 State-of-the-art Implementations of Deterministic Encryption

The implementation of [BCL+09] relies on two main blocks: Hypergeometric Distribution (HGD) and the function $TapeGen$.

Authors	Year of Publication
Ozsoyoglu et al. [OSC03]	2003
Agrawal et al. [AKS+04]	2004
Boldyreva et al. [BCL+09] [BCO11]	2009
Agrawal et al. [AAE+09]	2009
Seungmin et al. [STD+09]	2009
Kadhem et al. [KAK10a]	2010
Kadhem et al. [KAK10b]	2010
Xiao et al. [XYH12a]	2012
Xiao et al. [XYH12b]	2012
Yum et al. [YKK12]	2012
Liu and Wang [LW12]	2012
Liu and Wang [LW13]	2013
Popa et al. [PLZ13]	2013
Ang et al. [AWW14]	2014

Table 1: Order-preserving Encryption (OPE) proposals.

To better understand the HGD construction, consider the following example. There are 100 balls in a bin, 20 of which are black and the remaining 80 are white. When taking one ball, it is intuitive that the probability of that ball being black is 20%. Now suppose we take y balls out of the bin without any replacement. There is a random variable X following a hypergeometric distribution that specifies the number of withdrawn black balls, following a hypergeometric distribution. Let us now assign the previously stated values to variables: let N be the whole set (100), and M be the set of black balls (20). Let y be the total number of withdrawn balls and x the number of withdrawn *black* balls. The probability of x black balls appearing in the y withdrawn balls is given by the following equation:

$$P_{HGD}(x, N, M, y) = \frac{\binom{y}{x}\binom{N-y}{M-x}}{\binom{N}{M}}$$

The binomial coefficient is defined as:

$$\binom{y}{x} = \frac{y!}{x!\,(y-x)!} \text{ for } 0 \leq x \leq y$$

In 1985, Kachitvichyanukul and Schmeiser designed and published an exact algorithm for sampling according to the hypergeometric distribution [KS85]. This algorithm takes the following arguments: the number of elements of the complete set, N; the number of elements of the subset, M; the number of samples, y; and a source of randomness, cc. The algorithm is deterministic. The output is x, a variable that follows the hypergeometric distribution, and that represents the number of withdrawn black balls.

The $TapeGen$ construction aims to provide all the necessary randomness required by the implementation of the OPE scheme in [BCL+09]. It produces inputs for the HGD algorithm and it is also used for selecting the return value of the encryption and decryption algorithms. The authors proposed a *length flexible pseudorandom function*, LF-PRF, that uses a *variable-input-length pseudorandom function*, VIL-PRF, F, and a *variable-output-length pseudorandom generator*, VOL-PRG, G. They also present a proof stating that if F is a VIL-PRF and G is a VOL-PRG then $TapeGen$ is a LF-PRF. The following equation presents a definition for $TapeGen$:

$$TapeGen(1^l, k, x) = G(1^l, F(k, x))$$

1^l is the generator output length[3], k is a key chosen uniformly at random from the key space and x is the input. The authors recommend to instantiate F with CMAC (Cipher-based Message Authentication Code). G can be instantiated with AES128 in CTR (Counter mode). A concrete instantiation of $TapeGen$ can be as follows:

$$TapeGen(K, x) = AES128CTR(CMAC(k, x), T)$$

T being a string of length l.

We now present a description of the OPE scheme in [BCL+09]. The key generation algorithm Gen is straightforward, so we will focus our presentation on the more intricate Enc and Dec. The pseudocode for encryption algorithm Enc is presented in Figure 4. Let $\mathcal{M}$ and $\mathcal{C}$ be the plaintext domain and ciphertext domain, respectively. Let m be the number to be encrypted. Since the algorithm is recursive, consider that the explanation starts at the moment of the first call.

In line 1 of the listing in Figure 4, the cardinality of the sets $\mathcal{M}$ and $\mathcal{C}$ is placed in M and N, respectively. In line 2, d is assigned with the minimum value of $\mathcal{M}$ minus 1. The same for variable r. Now, in line 3, y contains N divided by 2 plus r. Following the previously mentioned example, where N had the value of 100, and assuming that the ciphertext space contains values from 1 to 100, then y will contain the value 50. Since M is 20 and, as such, greater than 1, the execution jumps to line 8. $TapeGen$ is now used to generate a *random* token cc that will be used as HGD input.

[3] The generator output length representation is similar to the representation of the security parameter described in Footnote 1.

```
   Enc_{M,C}(k,m)
1  m <- |M|; N <- |C|
2  d <- min(M) - 1; r <- min(C) – 1
3  y <- r + ⌈N/2⌉
4  If M = 1 then:
5     cc <- TapeGen(k,m)
6     c <- C
7     Return c
8  cc <- TapeGen(k,y)
9  x <- HGD(M, C,y,cc)
10 If m ≤ x then:
11    M <- d + 1,…,x
12    C <- r + 1,…,y
13 Else:
14    M <- x + 1,…,d + M
15    C <- y + 1,…,r + N
15 Return Enc_{M,C}(k,m)
```

Figure 4: OPE Encryption Algorithm.

Recall that HGD receives as arguments the number of elements of the big set (ciphertext domain), the small set (plaintext domain) and the number of draws (samples). Now we replace the previously given N and M by $\mathcal{C}$ and $\mathcal{M}$, respectively.

The variable x will now be sampled by HGD, which is invoked as follows: $HGD\big((1,\dots,100),(1,\dots,20),50,cc\big)$. Given this execution, x will be assigned with the value 10 with $\approx 20\%$ chance, values 9 and 11 with a chance of $\approx 17.5\%$, and so on. Since the same key k and the same number m will generate the same token cc, and since HGD is deterministic, x will have the same value between different executions for the same k and m. This property is what allows this scheme to meet the correctness properties expected from OPE schemes.

At this point, the algorithm performs a comparison between m and x. If $m \leq x$, for example $m = 5$, then it will set $\mathcal{M}$ and $\mathcal{C}$ to $1,\dots,x$ and $1,\dots,50$ and then recursively call itself with the new domains. In fact, it is performing a binary search to find a correspondence (encryption) for m in domain $\mathcal{C}$. It only stops when the message space ($\mathcal{M}$) contains only one element, m itself.

The pseudocode for the decryption algorithm Dec is presented in Figure 5. Decryption is similar to encryption. In line 12, and comparing it with line 10 of the encryption algorithm, the ciphertext c is compared with y to select the direction of the search: *right* or *left*. In fact, it is intuitive to understand that the search path over the domains will be the same as the one performed by the encryption algorithm since order is preserved when encrypting. Regarding the last step (starting in line 4), if the message space contains only one element, the decryption of the ciphertext has been computed and that element is now assigned to m.

```
  Dec_{M,C}(k,c)
1  m <- |M|; N <- |C|
2  d <- min(M) - 1; r <- min(C) – 1
3  y <- r + ⌈N/2⌉
4  If M = 1 then:
5     m <- min(M)
6     cc <- TapeGen(k,m)
7     w <- C
8     If w = c then Return m
9     Else return Error
10 cc <- TapeGen(k,y)
11 x <- HGD(M, C,y,cc)
10 If c ≤ y then:
11    M <- d + 1,…,x
12    C <- r + 1,…,y
13 Else:
14    M <- x + 1,…,d + M
15    C <- y + 1,…,r + N
15 Return Dec_{M,C}(k,c)
```

Figure 5: OPE Description Algorithm.

2.1.7 Alternative approaches to Order-preserving Encryption: CryptDB

CryptDB, presented in the literature in 2011, was defined by the authors as a "system that provides practical and provable confidentiality" [PRZ+11]. The security of CryptDB's schemes was discussed in [AS14] and [NKW15], and in response to these publications CryptDB authors published a report where they described the best practices and guidelines for using CryptDB system in a secure way [PZB15]. One of the key components of CryptDB is OPE. CryptDB's approach is not as stateless as the one proposed in [BCL+09], instead requiring the server to hold more information than just data. It also requires additional coordination, as the employed protocols are interactive between client and server. However, CryptDB's solution requires significantly less computational power on the client side, with the encryption and decryption operations being 1 to 2 orders or magnitude more efficient than the previously presented solution.

CryptDB considers the following simple scenario. There is a client entity with sensitive data to store and query, and an untrusted server that is capable of storage and computation. The authors provide two implementations for order-preserving encryption that are compatible with this scenario (and therefore SafeCloud Solution 1). Both leak the relative order when considering any type of server corruption. The first has better performance but only provides security against a semi-honest adversary, while the second can withstand active adversarial behaviour. Given their similarity, we now describe how the first implementation works, and then detail how it can be adapted to the second.

Consider a client that wants to store the set of values {30, 20, 40, 10}. The first step is performed by the client, and consists in encrypting all the values using a symmetric deterministic encryption algorithm. Let us assume, for the sake of explanation, that the encrypted values are the following:

`0x303030, 0x202020, 0x404040, 0x101010`

The first approach uses mutable Order-preserving Encryption (mOPE). The server has two different auxiliary data structures to maintain mOPE-related information. These

structures are: an AVL Tree (a self-balancing binary search tree) and a Hash Table. Each node of the tree stores a (deterministically) encrypted value. All encrypted values are inserted in the tree considering the original value. Since the server only has access to encrypted values, the client has to *guide* the server through the tree traversal. The first insertion is obvious, so one can assume that the first value is already inserted. To insert the number 20 in a server that only contains the 30 encoded, the insertion protocol proceeds as follows:

1. The client requests the first encrypted value to the server and receives `0x303030`
2. The client decrypts the ciphertext, obtaining the value 30. It then compares with 20, and since $30 < 20$, the client requests the server for a left-side node ciphertext.
3. The server returns a message to the client indicating that the left node is free.
4. The client encrypts the value 20 and sends the `0x202020` to be stored on the left-side node in the server.

The client proceeds until all values are inserted in the remote tree, whose final result is shown in Figure 6. Now that the tree has all encrypted values inserted correctly, it is necessary to assign an encoding (the order-preserving encrypting result) to all of them. The number that will be assigned to each one of the nodes is given by the position of each node in the tree. This number will be built bit by bit as shown next.

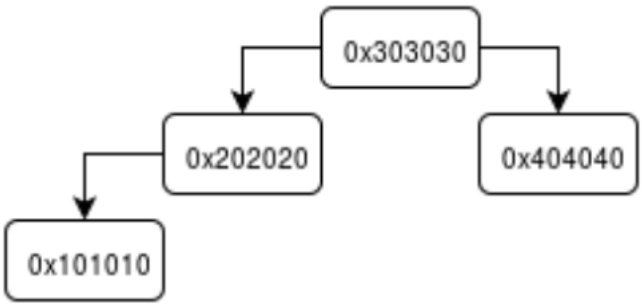

Figure 6: Resulting OPE Tree.

Encrypted value	Order-preserving Encoding Value	OPE Value in Decimal
`0x101010`	[00]10	2
`0x202020`	[0]100	4
`0x303030`	[]1000	8
`0x404040`	[1]100	12

Table 2: CryptDB Hash Table.

First, we define our order-preserving encoding domain, which we will assume to be 4 bits for the sake of the example, giving us the possibility of inserting 16 different values, 24 in the tree, and thus in the system. Starting from the higher bit, the corresponding bit should be set to 0 if the path goes to the right and 1 if it goes to the left. When the path has less than 4 bits, padding should be inserted. The first bit of padding is 1 and all the remaining are 0. This mapping will be stored in the previously mentioned hash table, illustrated by Table 2. The set of values {2,4,8,12} is the set that contains the result of the order-preserving encryption method. These are the numbers that will be used to allow the execution of queries over data.

The querying of values is made interactively. For instance, when the client wants to perform a record selection from the server "all records with a certain attribute greater than 30", it first encrypts the value 30, and asks the server for the encoding of `0x303030`, to which the server responds 8. The client finally changes the query to "all records with a certain attribute greater than 8" and sends it to the server.

The reason why ciphertexts are mutable is related to the underlying technique used by this method. Furthermore, it is proven by the authors of this work that the usage of a mutable ciphertext scheme is a necessary condition to leak no more than the order of the elements to the server. Since the encoding of the values depends on their position in the tree, and since it is not possible to control the insertion order of the values in a real-world environment, the tree can assume an unbalanced state. This means that the tree has to be balanced, and consequently the positions of the values will have to change. Therefore, from time to time, and depending on the insertion operations performed, the remotely stored encodings will also have to be updated.

In the available implementation of CryptDB, the authors selected a B-Tree structure instead of an AVL. The same principles of the above explanation apply for this kind of structure. The selection of B-Trees to support this key-component was due to the higher performance shown under real-world circumstances. In theory, scapegoat-trees would have better asymptotic performance, $\mathcal{O}(log\ n)$, in the number of updates to the encodings when performing balancing operations. This first approach is resistant against a semi-honest adversary, since it does not leak more than the order to the server when used in the appropriate conditions.

In order to prevent leakage associated to active adversaries, the client must have a method to verify the integrity of the tree, so as to reject adversarial tampering. In this case, when the server does not remove nodes upon request, a property called *same-time OPE security* is violated. This property states that the server is only allowed to have access to the order of the elements that are inserted in the storage at each moment and should never be able to perform inferences between the order of current and old elements. This can be achieved by using Merkle trees [M89], used to enforce integrity. However, if this is not implemented considering second pre-image attacks [K+09], the server can easily trick the client into accepting a tampered result.

2.1.8 Alternative approaches to Order-preserving Encryption: ARX

Following an approach somewhat similar to the one implemented in CryptDB, a strongly encrypted database system was proposed in [PBP16]. This Solution also considers a trusted client and an untrusted server, and addresses the issue of leaking the order of data to the server. For this type of system, all data is encrypted using standardly secure symmetric cryptography (see the Authenticated Encryption section for more details).

One of the main motivations for this approach is to provide resistance against snapshot attacks. Persistent corruptions can still infer order of stored data by analysing access patterns, so its security with respect to order applies only to snapshot corruptions. This means that an image of the server state and its encrypted data at any current time (excluding client interactions) must not be sufficient for inferring any meaningful information regarding the original data. This work only considers semi-honest adversarial behaviour.

To perform a range query, for instance “all records with a certain attribute greater than 20”, a special index is used, *ARX-RANGE*. For each searchable attribute there is one index, this index is a tree and it is located in the server. Each node of this index is a garbled circuit that receives as input a special token given by the client and it has 2 outputs: a direction (right or left) and a special token to be given to the next node, which is also a garbled circuit.

A garbled circuit is an encrypted program (created securely by the client) that allows the server to perform obfuscated computations. These computations are present in each of the nodes, and consist in comparisons between two numbers: the number contained in the input token, and an encoded number in the circuit. Following the provided example, the client first creates a token that contains the encoding of the number 20. Then, the client sends this encoding to the server. Let us consider that in the first node of the *ARX-RANGE* index there is a garbled circuit that contains the following encrypted computation: > 30. Since $20 < 30$, this first garble circuit should output direction *right*, and a token will be given to the next node. Now consider that the current right-node has the following computation: > 20, and it has no child nodes. Since the result of this computation is a valid result, the server will return to the client a list of encrypted values (also created by the client) that are linked with the condition of being greater than 20. The client decrypts this list, which contains all database identifiers of the records that have the considered attribute greater than 20. The decryption of these identifiers is, in itself, an encryption of the identifiers stored in the database. The client shuffles the encrypted identifiers (so they do not reveal the relative order between them) and requests the server to retrieve the correspondent records.

Security considerations demand that each garbled circuit is used only once. This implies that, for every range query, the client must provide new garbled circuits to the server to replace the ones used in the performed operation. The number of garbled circuits to be replaced is logarithmic in the number to different values for the corresponding attribute, so this cost is amortized as the amount of data increases.

2.2 Deployment

The architecture proposed for Solution 1 is divided in two domains, a trusted domain and an untrusted one. As shown in Figure 3, both domains are composed of several components that are independent and can have different implementation approaches.

Regarding the trusted domain, client applications may use our secure database solution by resorting to a SQL or a NoSQL API. Java Database Connectivity (JDBC) will be used to provide the clients with an ANSI SQL API while abstracting any specific API detail required by the SQL Engine implementation.

The SQL engine is based on previous work developed in the CumuloNimbo European Project (FP7-257993) and enables the cooperation of the SafeCloud project with the work developed in previous projects. The engine is implemented in Java and has no security/privacy considerations as its main focus is to provide a solution capable of answering SQL queries with high throughput.

The SQL query engine provides an ANSI SQL interface with transactional support (*Commit*, *Abort*, and *Begin*). This engine is responsible for planning, optimizing and executing queries, and works together with components for transaction management and data storage. The query engine architecture has the key advantage of being stateless regarding application data. Data manipulation language (DML) statements (*Select*, *Insert*, *Update* and *Delete*) can be executed without any coordination amongst different instances. As a result, the system can be deployed in distributed infrastructures and scaled to a large number of servers. The query engine layer translates user queries into a set of NoSQL operations (*Put*, *Get*, *Delete*, and *Scan* operations) to be invoked on the storage layer. This storage layer can be any NoSQL database with a similar API to HBase. This translation mechanism includes a mapping of the relational schema into an appropriate optimized NoSQL schema. Once again, SafeCloud architecture assumes an HBase NoSQL interface in the Proxy component, which matches the requirements of the SQL engine.

HBase already provides advanced query facilities (filters) to retrieve specific sets of key-values. The query engine component takes advantage of HBase features to push parts of the query execution to the NoSQL data store layer in order to leverage NoSQL computation and reduce the amount of information that must be sent and processed in the SQL engine. Moreover, by using HBase as the storage backend, the SQL Engine takes advantage of HBase's scalability to support a higher number of clients and achieve improved query performance.

Finally, transactions are handled in a holistic manner providing full ACID properties across the entire SafeCloud stack (SQL Engine and NoSQL backend). In particular, the transaction support system (also based on work presented in the CumuloNimbo European Project) is designed to provide transaction management for user queries.

The NoSQL client API will be compatible with a subset of the default HBase API. This API will be used both by NoSQL client applications and by the SQL Engine. Namely, the API will support the following HBase operations.

- *Put* - Insert a key-value record on the NoSQL database.
- *Get* - Retrieve a key-value record from the NoSQL database.
- *Delete* - Delete a key-value record from the NoSQL database.
- *Scan* - Retrieve a range of key-value records from the NoSQL database.

Moreover, the *Get* and *Scan* operations will support the following HBase filters: *SingleColumnValueFilter*, *WhileMatchFilter*, *RowFilter* and *BinaryPrefixComparator*. These filter requirements are posed by the SQL engine that uses them in order to provide an ANSI SQL API. A more detailed description of these operations can be read in deliverable D3.1.

The proxy component will be compatible with the previous HBase interface and will be responsible for processing client and SQL Engine requests, and issuing these to the untrusted domain. More concretely, the proxy will be implemented in Java and will modify the HBase client to issue functionally similar requests to the HBase cluster deployed in the untrusted domain, considering the underlying cryptographic techniques. The unmodified HBase client exports to the applications a NoSQL API as the one discussed above, and manages how clients connect and send requests/receive replies from HBase. Since requests and replies are handled by this component,

modifying it will enable the encryption of sensitive data before storage (*Put* operation) and the decryption of data upon retrieval (*Get* operation), and, depending on the functionalities made available by the techniques, *Scan* or *Filter* operations can be converted to similar operations for equality and range query retrieval of protected data.

Some cryptographic mechanisms, such as CryptDB or ARX, require additional remote state and computations in the HBase cluster (untrusted domain). These will be taken care of by employing the HBase Coprocessor mechanism. This mechanism can be seen as a plugin that adds additional behaviour to HBase clusters without modifying its core implementation.

Note that the joint effort of the local Proxy and (in some cases) the remote Coprocessor allows the system to display a transparent behaviour with respect to the chosen security techniques, and the way data is being stored/processed in the untrusted domain, from client applications and the SQL Engine. The untrusted domain is expected to be composed by a single cluster running an unmodified HBase deployment, modulo the extended Coprocessor mechanism.

2.3 Discussion

Solution 1 considers a somewhat standard scenario for the development of solutions in the cloud paradigm. Assuming that the remote cloud provider is untrusted (hence the untrusted deployment referred to in the SafeCloud architecture), security techniques should be enforced to prevent malicious entities to act upon the proposed systems. The first section of this chapter, in particular, has been dedicated to a thorough survey of the primitives and state-of-the-art implementations that are compatible with this simple scenario.

Unfortunately, by the fundamental properties of these cryptosystems, it is not possible to infer a scheme that is objectively "the best" for deployment on all possible solutions. In fact, the existing variety of approaches is a direct consequence of exploring multiple possibilities of trade-offs between security guarantees, available computational functionalities, and performance of implementations. To highlight the importance of different techniques in different contexts, we now propose two scenarios: A and B. In scenario A, a user wants to offload a database of highly sensitive files that will be used mostly for archival purposes, i.e. not to be queried over and only expecting to retrieve single files from time to time. In scenario B, a user that wants to offload a database containing mildly sensitive numerical data, to be often queried. The usage of standard authenticated encryption seems like a natural choice for A, as it provides strong security guarantees without any significant disadvantages for that usage, but useless for B, as frequent querying would imply retrieving the full database and performing the operations locally, which is completely unfeasible. Alternatively, the usage of order-preserving encryption (or similar techniques allowing range queries) might be a fitting solution if the user in B is willing to allow the relative order of the stored data to be leaked in exchange for enabling the cloud provider to take the responsibility for query computation. In turn, the user of scenario A is likely not interested in this technique, as it seems pointless to sacrifice confidentiality with respect to data ordering to enable a computation that the system is not expected to perform in the first place.

Contrary to the provided examples, when considering real-world applications, the selection of techniques for specific requirements is hardly often straightforward. Since most practical secure solutions for cloud deployment consider only a subset of proposed mechanisms (in most cases, even reducing the scope to a single flavour of data encryption), users lack the tools required to have their implementation tailored to their application-specific needs, so they often compromise to deploy their solution using a suboptimal system (functional, performance and/or security wise).

SafeCloud aims to fill this significant niche of secure frameworks by allowing the deployment of solutions that can opt to use different cryptographic mechanisms for different levels of data security/functionality requirements. This is by no means the only approach that has ever aimed to provide a more granular approach to data protection (CryptDB is an example of providing custom layered encryption, but several other systems have been designed according to a similar approach, such as Google's Encrypted BigQuery[4], SAP's SEEED[5], or Microsoft's Always Encrypted SQL Server[6]), and should instead be seen as a natural progression towards proposing solutions for the deployment of systems taking into consideration more granular application-specific security requirements.

One of the central objectives for the project is about providing this in a way that is transparent to the end-user. More specifically, we are interested in providing a framework in which, after the system has been initialized and the privacy-preserving techniques instrumented according to system specifications, the interaction of the user with the Client Application is the same, regardless of the underlying cryptographic schemes employed[7]. This is made possible by the joint action of the Proxy component, and by the remote HBase Coprocessor (which, for many of the proposed schemes, is not even required).

By enabling a fully customizable Proxy, SafeCloud enables the instrumentation of solutions that can not only freely select the type of encryption used (via standard authenticated encryption, order-preserving encryption, or other variants), but also select different techniques to be used in different contexts. The importance of this granularity level can be exemplified by recalling scenarios A and B, considering that systems generally store and manage data sets with different goals in mind: some might be highly sensitive for archive, others less sensitive and expected to be frequently queried. This essential component executes the required security mechanisms depending on the system specification, and inherently enables a higher level of customization of technique usage according to the specific application requirements.

As a whole, Solution 1 of the SafeCloud framework takes a user-centred approach by allowing the combination of security mechanisms, supporting an application-specific optimized implementation for practical deployments. Solutions 2 and 3 extend the scope of the framework to more generic scenarios, namely multiple untrusted domains and untrusted clients, further expanding the combination of allowed deployment scenarios for optimized approaches.

[4] https://cloud.google.com/bigquery/
[5] https://www.sics.se/sites/default/files/pub/andreasschaad.pdf
[6] https://msdn.microsoft.com/en-us/library/mt163865.aspx
[7] Modulo performance differences, which are inherent to the computation requirements of each implementation.

3 Solution 2: Secure processing in multiple untrusted domains

3.1 Privacy-preserving techniques

Solution 2 (Figure 7) extends the domain of the previous approach by allowing the usage of multiple untrusted domains instead of just one. Naturally, this does not exclude the deployment of techniques applicable on a single remote domain, and can even allow performance improvements, e.g. by distributing the load onto several remote machines. Furthermore, and more importantly, this enables the leverage of additional privacy-preserving techniques that hinge on a dynamic usage of untrusted domains.

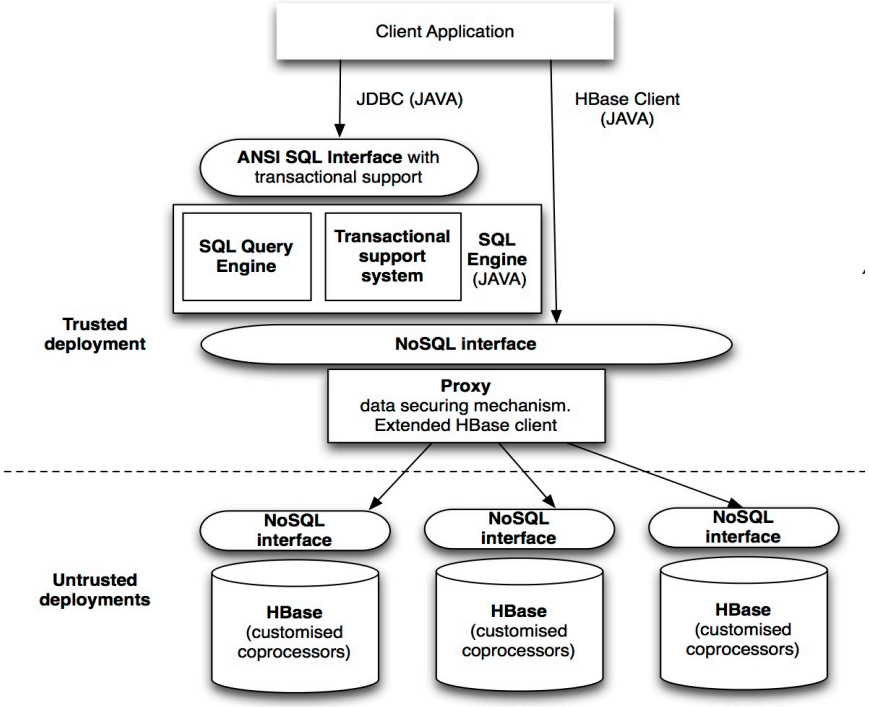

Figure 7: Solution 3 architecture: multiple untrusted domains.

3.1.1 Secure Multiparty Computation

The secure execution of code over multiple untrusted participants is a problem within the domain of multiparty computation. Secure Multiparty Computation (MPC) can be defined as a problem in which a number of n players $P_1, \dots, P_n$ in possession of inputs $x_1, \dots, x_n$ (respectively) agrees to compute a function $f(x_1, \dots, x_n) = (y_1, \dots, y_n)$, so that each P_i knows y_i and learns nothing additional that could not be deduced from its personal input and output. Yao's millionaire problem [Y82], where two millionaires attempt to evaluate which is wealthier without disclosing the real value of their wealth, is a simple example of MPC.

This can trivially be achieved if we consider the usage of secure channels and a trusted third party (TTP). In this assumption, each P_i would only need to send x_i to the trusted party and expect to receive y_i as, by definition of the TTP, function f is executed correctly and securely. However, it is not feasible to assume that participants of joint

protocol executions in real-world scenarios can agree on a trusted participant to compute over their sensitive data. As such, the problem of MPC refers solely to the cases in which these conditions are unavailable.

The generality of applications makes the MPC security definition a complex task. To assist in this effort, the approach of ideal versus real world is generally considered to be adequate towards formalizing a protocol's security definition. This concept is used in methods for formalizing composable security proofs, such as Universal Composability [R01], reactive simulatability [PW00], abstract cryptography [MR11] or inexhaustible Turing machines [KT13].

The real world describes the behaviour expected from an actual protocol execution for a specified number of participants. An ideal world is essentially a formalization of the same protocol under the TTP setting. In this world, an ideal functionality executes procedures, while a simulator that is fully aware of the occurrence of corruptions simulates towards the adversary what it expects to see in a real-world scenario.

The approach is based on the intuition that if such simulator exists, then the adversary cannot possibly gain more information than that he would obtain when interacting with the protocol, executing with the described TTP. If the views of the real and ideal worlds follow similar distributions, then we say that the protocol is a secure implementation of the ideal functionality[8]. Composability properties are a major strength for this approach. In this context, if a given functionality is simulatable and composable, we can assume it exists in the real world, and therefore it can be employed to prove other functionalities: if $\pi_1$is proven to be a secure realization of F, and π_2 is shown to be a secure realization of G assuming F, validity of G is dependent on the correctness of π_1's proof. Not all real- versus ideal-world approaches imply composability of functionalities. This can be achieved through inherent characteristics of the security model (universal composability makes use of a hybrid model to assert composability of ideal functionalities), but composability properties may also be formalized at a higher level.

As such, security analysis is particularly dependent on the types of adversarial behaviour and corruptions considered, as it is naturally easier to present an MPC protocol as secure if we consider more restrained adversaries. Feasibility of MPC implementation for real-world solutions hinges on these assumptions, as security against extremely powerful adversaries can be both extremely costly performance-wise (sometimes requiring the employment of cryptographic primitives which have very high computation and communication requirements) and not quite representative of the realistic malicious entities acting upon the system.

3.1.2 State-of-the-art Implementations of Secure Multiparty Computation

Secret sharing is an essential technique to protect sensitive data amongst independent entities in such a way that one single participant cannot have access to the sensitive information by itself. Formally, and more broadly, a (k, n) threshold scheme takes a sensitive value v and distributes it in n pieces $v_1, \ldots, v_n$. Recovering the original value v

[8] This similarity can refer to views that are either identical, producing perfectly secure implementations, statistically indistinguishable, producing statistically secure implementations, or computationally indistinguishable, producing computationally secure implementations.

requires the joint collection of at least k out of the n shares. In many practical implementations $k = n$, i.e. all shares are required to recover the original data.

With the motivational goal of protecting a private key, two independent researchers, Adi Shamir [S79] and George Blakley [B79], presented their solutions for secret sharing in 1979.

The solution proposed by Adi Shamir employs polynomial interpolation on a 2-dimensional plane to divide a sensitive value on a set of secrets. As with every secret sharing scheme, it is assumed that v can be converted into a number. Thus, given the numerical representation of v and a random polynomial $q(x)$ of degree $k - 1$ where the first element is v, the evaluation for each coefficient generates a new secret. Given k secrets and the corresponding indexes, it is possible to obtain the original coefficients by polynomial interpolation.

Blakley's scheme uses n nonparallel $(n - 1)$ dimensional hyperplanes that intersect in a single-point to encode the value v into multiple secrets. Value v is determined by a large collection of planes, while a plane can only determine reduced amounts of values. Therefore, to encode a sensitive value in secrets, a random plane L from the ones defined by the value is chosen. Furthermore, another set of subspaces is randomly chosen so that they intersect in L, and thus encoding the value as a set of coordinates. Each player will store enough information to generate a single sub-plane, and only when enough planes are intersected can the secret value be decoded.

The implementation of complex (k, n) threshold schemes can be both intricate and computationally demanding, when compared with simpler schemes such as the one employed by Sharemind. The Sharemind secret sharing uses a simple additive secret sharing scheme that works by taking a sensitive value v, generating two random numbers from a finite field $\mathcal{Z}_{2^n}$ and subtracting v from the sum of the two to obtain the third. This secret sharing scheme considers an additive ring, which is an essential component for the usage in secure multiparty computation protocols. In particular, this technique is trivially shown to be secure against adversaries corrupting 2 out of the 3 participants. Assuming that a trusted domain can securely distribute private information using secret sharing, MPC protocols are then capable of performing general computations over the data without leaking the information held in each domain, or any information related to the computation being carried out (besides the output, and what can be inferred from it).

The protocols chosen for Solution 2 and 3 are based on the protocols proposed by the Sharemind framework [B+12]. It is worth mentioning that this framework is being developed by CYBERNETICA, one of the partners of the SafeCloud project. Therefore, the project consortium has a deep know-how regarding the functionalities of Sharemind. Moreover, the research and outputs of the project will be beneficial for advancing the current state of this framework.

These schemes are shown to be secure under the circumstances of semi-honest adversarial behaviour, statically and persistently corrupting up to one out of the three protocol participants. This security model is common in the development of secure cloud solutions, where the computation is to be distributed over several competing cloud providers. Deployment assumes the pre-establishment of secure channels

amongst the untrusted deployments, which can be deployed using the SafeCloud secure communication layer developed by WP1.

Two main protocols from the Sharemind framework will be considered for performing database operations in this context: the *Equality* protocol and the *GreaterThan* protocol. These protocols enable the databases to perform record comparison and range queries over the protected data, and thus act as the building blocks for many SQL and NoSQL operations. Every NoSQL API operator requires the comparison of records to retrieve a single stored value, or to retrieve a set of records. Without equality matching or order comparison, a NoSQL database can only store information.

These two protocols, essential for basic secure query computation, are going to be described next with an overview of the computation on each party, as well as a description regarding how all participants jointly calculate the desired output without leaking information. To understand the presented code, consider that players are denoted by P, and that every calculation is performed on the additive ring $\mathcal{Z}_{2^n}$. Furthermore, it is assumed that each player holds two secrets, each one corresponding to the secret sharing of u and v. These values are composed by secrets (u_1, u_2, u_3) and (v_1, v_2, v_3), correspondingly. These two protocols are based on a simple arithmetic equation between u and v. However, since the values are encoded into secrets stored in independent parties, the equation is performed with an MPC protocol that takes additional steps to achieve the same result without compromising the security.

P_1 generates random r_2 <- $\mathcal{Z}_{2^n}$
P_1 computes r_3 <- (u_1-v_1)- r_2
P_1 sends r_i to P_i (i = 2,3)
P_i computes e_i = (u_i-v_i) + r_i (i=2,3)
P_1 sets T_1 <- 2^n-1
P_2 sets T_2 <- e_2
P_3 sets T_3 <- (0-e_3)
Return $BitConj$(T)

Figure 8: Algorithm BitConj.

Take as an example the Equality protocol. This construction is built upon the observation that if two values are equal, then the subtraction of these values yields 0. The subtraction of the global values u and v is calculated with the MPC protocol, expressed in Figure 8. This protocol builds on the stored secrets of each value, (u_1, u_2, u_3) and (v_1, v_2, v_3), to calculate new secrets that return the correct result. The first steps of the protocol are the generation of a random number r_2 and calculation of r_3 by the player P_1. r_3 is nothing more than the difference between u_1 and v_1 offset by the random number r_2. After these two steps, the first player sends r_2 to P_2 and r_3 to P_3 so that each player may calculate the difference of the secrets they hold and offset them with the secrets from P_1. The result from the local difference of each remaining player are the secrets e_2 and e_3, described in Figure 8. Both secrets hold the result from subtracting u and v when added, as can be verified as follows:

$$\begin{aligned} e_2 + e_3 &= (u_2 - v_2) + r_2 + (u_3 - v_3) + r_3 \\ &= (u_2 - v_2) + (u_3 - v_3) + (u_1 - v_1) \\ &= u - v \end{aligned}$$

In fact, the addition of e_2 and e_3 is calculated by another secure protocol, which is the function $BitConj$. This is also another Sharemind protocol for computing difference. Once the protocol is finished, three secrets are sent back to the client, which will recover either a 0 or any other result (for inequality).

The *GreaterThan* protocol, displayed in Figure 9, works very similarly to the equality protocol, by calculating the difference between values. The protocol returns 1 when u is greater than v and 0 otherwise. While the *Equality* protocol just provides the difference, the *GreaterThan* protocol knows that when u is greater than v, then the most significant bit of the calculated difference is 1. Thus, to retrieve the highest bit there is another secure protocol that performs a right shift. By executing a right shift that sends the most significant bit to the least significant bit, the resulting output becomes 1 or 0, depending on the highest significant bit value.

```
P_i calculates d_i <- u_1-v_i
w <- ShiftRight(p, n-1)
Return Reshare(w)
```

Figure 9: Algorithm GreaterThan.

The complete execution of *GreaterThan* requires two additional protocols $ShiftRight$ and $Reshare$, demanding a much higher level of message exchange than the *Equality* protocol.

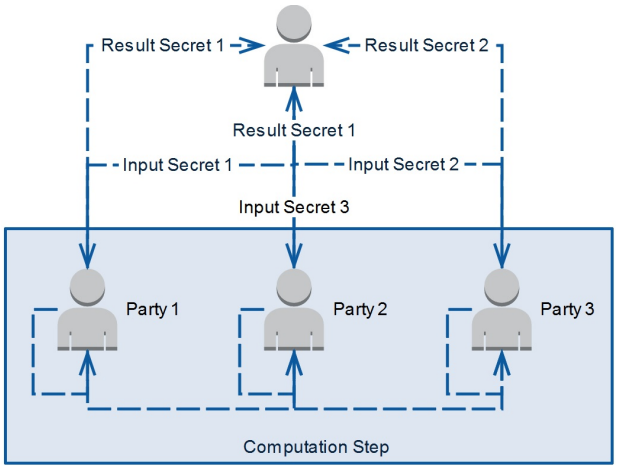

Figure 10: Interaction between MPC protocols and secret sharing.

The interaction between MPC protocols, secret sharing and parties is depicted in Figure 10. The picture displays a conceptual model where there is a *Dealer*, i.e. a trusted participant holding sensitive information that shares secret values and sends them to participants, where they are stored. Each party holds multiple secrets, but just a single share from every encoded value. Furthermore, the parties are able to execute secure protocols and communicate amongst them using secure channels. The request for a Sharemind protocol computation by the dealer in this setting is executed in two main steps. First, each party generates a new secret from their private inputs, this new secret does not disclose any information about the protected data and cannot be used by a party or attacker to decode the original value. The other step is performing a calculation based on the secrets exchanged between the players. This two steps are performed multiple times to achieve a desired output.

By leveraging secret sharing and MPC, the overarching goal of Solution 2 is to provide a fully functional NoSQL database built on top of three independent untrusted domains, each running its own untrusted NoSQL database. When using MPC, data privacy is ensured by storing a distinct secret in each domain, and thus protecting the original data from an attacker that has access to data shares held in a single domain. Additional techniques (such as the ones described in Solution 1) can also be employed in conjunction with MPC to achieve a tailor-made solution for the security/performance requirements of SafeCloud applications.

3.2 Deployment

Similar to Solution 1, the architecture proposed for Solution 2 is divided in two domains, a trusted domain and several untrusted ones. As shown in Figure 7, both domains are composed of several components that are independent and can have different implementation approaches. Client applications may use our secure database solution by resorting to a SQL or a NoSQL API. JDBC will be used as the interface between clients and the SQL Engine that will be identical to the component discussed for Solution 1. The operations supported by the NoSQL client API will also be the same as the ones supported in Solution 1, i.e. *Put, Get, Delete, Scan* and respective filters.

Again, the proxy component - a modification of the HBase client - will be compatible with the previous NoSQL interface. However, unlike in Solution 1, it will be responsible for managing the connections and issuing requests to three untrusted domains, while providing a centralized NoSQL database abstraction for NoSQL client applications and the SQL Engine. This component will perform the required translation between NoSQL calls received from the NoSQL interface into their equivalent operations. For instance, data insertion made by a *Put* operation will divide sensitive data into three secrets and store them into the corresponding HBase clusters. A *Get* operation for sensitive information will collect the secrets sent back by the HBase clusters, which will perform the necessary multiparty computation to retrieve the desired key-value pair, and will send the original data back to the clients. *Filter* operations can be converted into similar MPC operations in the HBase clusters for range query retrieval of protected data.

Each HBase cluster runs an unmodified HBase deployment. This does not require modifying HBase core implementation, since the added functionality will be implemented by resorting to the HBase Coprocessor mechanism. The Multiparty protocols will be provided as a library, and will resort to Coprocessors to perform the required computation over stored secrets in each HBase cluster. For instance, when the proxy issues a *Get* request for retrieving a specific protected value, each HBase backend receives a secret and performs the Sharemind equality protocol to look up for that value in the cluster storage. Note that both the secrets sent by the proxy and the computation done by the MPC protocols ensure that sensitive information is not leaked throughout the process.

Finally, the multiparty protocols require HBase clusters (in this solution, the Coprocessors) to exchange secrets amongst them. These secrets are intermediate computation results that are protected from attackers by the security of the protocol. To exchange such secrets, we will employ a middleware communication layer ensuring that secrets are delivered correctly across clusters.

3.3 Discussion

Solution 2 is a natural extension to the previous setting by allowing multiple untrusted environments (cloud providers, in the most likely scenario) to be used for the deployment of SafeCloud applications.

Note that if we assume that all of the remote environments are untrusted and cooperate, this is a particular case of Solution 1, where the untrusted environment is simply a distributed system deployed over multiple clouds. However, this approach allows us to make use of techniques that leverage the possibility of only a subset of these untrusted deployments acting maliciously. This is often considered to be a realistic assumption for some use cases, as different cloud providers offer services in a competing environment, and are therefore highly motivated to not cooperate. This obviously does not imply that all systems should assume (unlikely as it might be) that the three providers will never collude, but instead gives us another trade-off for deployment setting/assumptions that can be offered to the client. To exemplify this advantage, assume a scenario where the user wants to offload sensitive numerical data to the cloud, which he intends to query often, similarly to scenario B presented in the previous discussion (Section 2.3). With Solution 1, feasibly deploying this system would require for the relative order to be leaked to the untrusted deployment (assuming persistent corruptions). Solution 2, however, proposes an alternative: if the user considers reasonable to assume the non-collusion of the competing cloud providers, then this system can be deployed over multiple untrusted systems so that querying can be done remotely, and the ordering of original data is kept confidential from the untrusted environments. This can likely be a meaningful factor considering the potential of vulnerabilities related to ordered ciphertexts, such as frequency attacks.

Secret sharing and similar techniques have been used in the past to store data across several cloud domains while ensuring its privacy [BCQ+11, ZYT+15, TLS+15]. However, unlike our solution, such systems do not support any kind of processing over stored data. More recently, the usage of secret sharing with homomorphic properties has been a relatively common approach to the delegation of secure computations over protected data, notably [MB10, WCK+14, D+12, LTV12]. The employment of Sharemind for executing MPC in our scenario has three main advantages, namely: i) it allows SafeCloud to take advantage of a mature protocol where practical MPC use cases have been successfully executed and evaluated [LW15], ii) the security/functionality trade-offs provided by Sharemind nicely complement the considerations in which the previously proposed techniques can be employed, and iii) the in-depth know-how of the involved partners regarding the Sharemind tool allows us to more thoroughly build upon and expand the existing state-of-the-art of the framework.

Solution 2 is a particularly hot R&D topic, and similar European research projects such as SUPERCLOUD[9] are focused on this particular scenario. SUPERCLOUD differs from SafeCloud in its fundamental approach, by focusing primarily on allowing system deployments under highly heterogeneous cloud providers, and on enhancing user experience with respect to provider lock-in protection and administration complexity reduction via automation.

[9] https://supercloud-project.eu/

Similar to the arguments presented for the previous discussion, the Proxy and the remote Coprocessors are central to the availability of these techniques in a transparent way. Deployments for Solution 2 are not limited to using secret sharing and multiparty computation for the entire system, but can instead employ a combination with single untrusted deployment techniques to meet different levels of security and performance considerations. The approach taken in this adaptation is consistent with how these components were previously proposed, as the proxy executes the adequate security mechanisms depending on the specifications for the data being stored and managed.

Therefore, Solution 1 and Solution 2 of the SafeCloud framework provide a highly granular approach for combining security mechanisms according to application-specific requirements. The exclusivity between the solutions is dependent on the deployment scenario the user is considering, namely regarding the multitude of untrusted domains. Solution 3 further extends the system to allow untrusted clients to query sensitive data, a setting that is not encompassed by the previous approaches.

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

4.1 Privacy-preserving techniques

Solution 3 differs from the previous solution by enabling the possibility of having multiple local domains, where not all of them are trusted. The general idea for this scenario is to allow multiple client entities to upload their sensitive data to a shared database that can be later queried over by potentially untrusted end-users. This solution relies heavily on Sharemind, a programmable distributed secure computation framework using MPC. Sharemind is designed as an application server that runs applications written in the SecreC [DLR13] language, which distinguishes between private and public data on a type-system level. When Sharemind servers execute SecreC applications, they automatically use MPC protocols to process private data.

In general, the Sharemind framework supports different protocol suites which provide different security guarantees. This solution uses the shared3p protocol suite which is the oldest and most matured protocol suite for Sharemind, ensuring security against non-colluding *semi-honest* adversaries with *persistent* corruption of one *compute* party. The theoretical background behind shared3p is essentially the same as in solution 2: shared3p uses additive secret sharing and multiparty computation with 3 parties. The computations are *oblivious*, i.e. the parties running the computations will not learn the inputs nor the outputs. Sharemind offers a clear distinction between different actors in the system. There are 3 actor classes:

- *Input* party is an entity that provides private information to the system. In this case, it is the party that securely secret shares the inputs and transmits them to 3 *compute* parties.
- *Compute* parties store shares of secret-shared data and run the cryptographic multiparty protocols to perform computation over those shares of private data.
- *Result* parties are the ones who get the secret-shared results of the operations and analytics performed by the *compute* parties. Having all three shares, the *result* party can reconstruct the actual results.

One entity can play the roles of *input*, *compute* and *result* party at the same time. Generally, you have several *input* parties who do not want to reveal their inputs. For *input* parties, it makes sense to control one of the *compute* servers, which enables the *input* party to revoke access to its data. An *input* party can just turn off the *compute* server it controls and this stops further multiparty computations on its secret-shared data. The *result* parties must be trusted, otherwise an adversary can corrupt the *result* party, and therefore alter the outcome of the system.

For statistical analysis on the Sharemind framework, there is a tool called Rmind [BKL+14], designed to mimic command-line user interface of the popular data analysis system R [rsystem]. As seen in Figure 11, depicting its architecture, Rmind is composed of three main components. The first is the query interface, which translates R commands into execution of Sharemind scripts. The second is the library of scripts that handles the actual data transformations and statistical analysis inside the Sharemind application server. The third component is a utility application called *CSV importer* which, as the name implies, imports data from CSV files and secret shares it between *compute* parties.

In the Rmind setting: *input* parties run instances of CSV importer, *compute* parties run Sharemind servers with the privacy-preserving and statistical analysis algorithm library written in SecreC, and *result* parties run the Rmind tool to query the shared database.

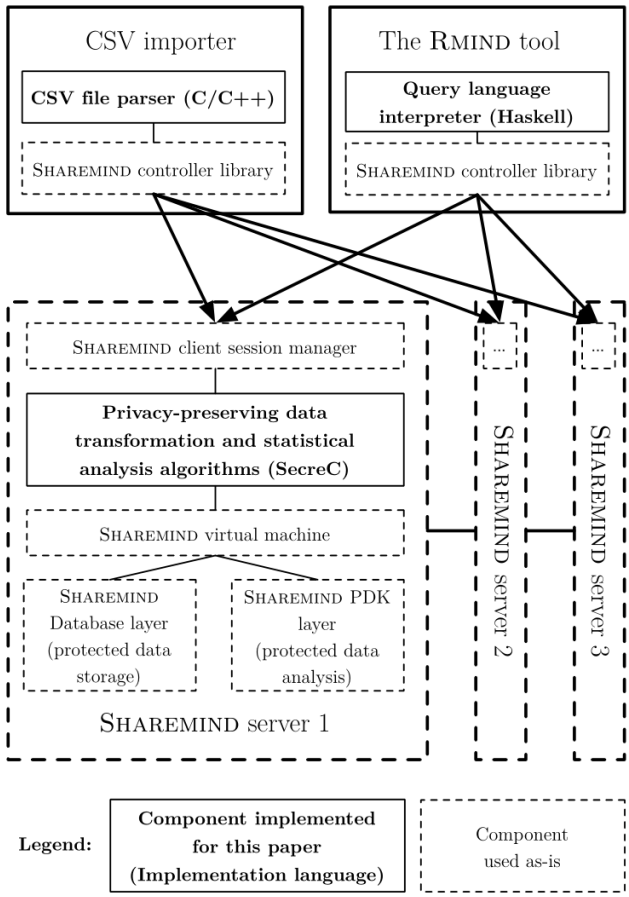

Figure 11: Architecture of Rmind.

Solution 3 has an architecture very similar to the Rmind tool, as shown in Figure 12. In fact, the only difference between solution 3 and the Rmind tool is the query interface. Instead of providing a command-line user interface as done for Rmind, solution 3 contains a component called SQL-controller. While Rmind has R-like query language, SQL-controller uses SQL as the query language. Solution 3 reuses the same library of privacy-preserving data transformations and statistical analysis algorithms to do the data processing. The library implements functions like aggregating, joining and filtering secret-shared data without leaking the original values.

The SQL-controller is written in Haskell and it relies on the Sharemind controller library for secret-sharing and communicating with Sharemind servers run by *compute* parties. The SQL-controller makes sure that query parameters are secret shared. However, the format of the query, the table, and the column names are not hidden from the *compute* parties. SQL-controller also uses the Sharemind controller library to reconstruct the results of SQL queries from the secret-shared responses obtained from the Sharemind servers.

The library of privacy-preserving data transformations and statistical analysis contains many useful functions. Almost all of these store their results in a new temporary table, and publish to the SQL-controller only the name of the new table. Therefore, only

metadata about table names and function calls is exchanged while processing a SQL query. Only in the end, when the final query results are obtained, are they sent back to the SQL-controller.

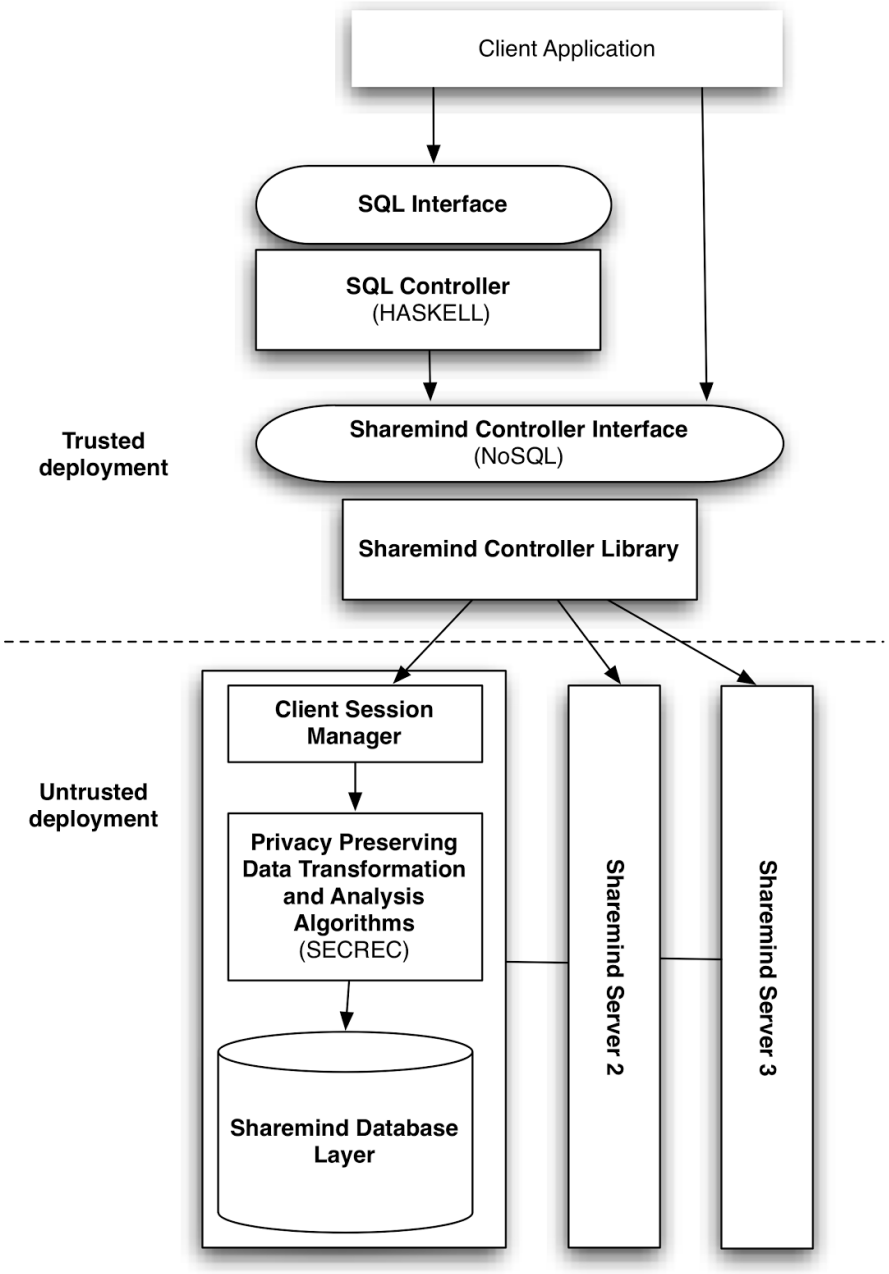

Figure 12: Solution 3 architecture.
(Note the resemblance with Rmind architecture in Figure 11)

Whether an output table is aggregated enough to hide information about individual rows or not, it is a non-trivial question which is particularly relevant for real-world applications. Data transformation algorithm library enforces a strict limit where a filter which would return less than *n* members will return none. The same limit is used in aggregations: when a particular group contains less than *n* members, it will be discarded from the results. In general, the problem of leaking information about inputs when publishing aggregates might be solved with differential privacy. In practice, access

policy-based solutions will be used to enforce that only the allowed information is extracted from the shared database.

To obtain efficient database operations like SQL JOIN and GROUP BY clauses, we need to compare a significant amount of secret-shared inputs for equality. These two algorithms follow the described behaviour:

- First, the input rows are obliviously shuffled.
- Afterwards, JOIN or GROUP BY keys are combined into one identifier which is then obliviously encrypted using AES.
- Finally, the AES encrypted identifiers are declassified and GROUP BY/JOIN operations can be executed by using public data (namely over the AES encrypted identifiers).

Figure 13 describes the aggregation algorithm. More details on JOIN operation can be found in [LTW13]. A formal description of the aggregation operation required for GROUP BY clause is detailed in [BKK16]. To satisfy ORDER BY clauses, we employ the sorting methods described in [BLT14].

Algorithm 1: Privacy-preserving aggregation

Data: Dataset $[\![\mathbf{A}]\!]$ with m attributes and N records, indices of attributes $C \subseteq \{1, \ldots, m\}$ by which to group the dataset, tuple of indices $(b_1, \ldots, b_v), b_i \in \{1, \ldots, m\}, (1 \leq v)$ of attributes that will be included in the resulting dataset, tuple of identifiers $(op_1, \ldots, op_v)$ of aggregation operations

Result: Dataset $[\![\mathbf{D}]\!]$ with v attributes and n records $(1 \leq n \leq N)$

1 Obliviously shuffle rows of $[\![\mathbf{A}]\!]$
2 Combine the values of attributes $c \in C$ into composite keys $[\![k_1]\!], \ldots, [\![k_N]\!]$
3 Use oblivious AES to encrypt the composite keys, denote them as $[\![k'_1]\!], \ldots, [\![k'_N]\!]$
4 $(k'_1, \ldots, k'_N) \leftarrow$ `declassify`$([\![k'_1]\!], \ldots, [\![k'_N]\!])$
5 Let n be the number of unique groups in $(k'_1, \ldots, k'_N)$
6 **for** $i \in \{1, \ldots, n\}$ **do**
7 **for** $j \in \{1, \ldots, v\}$ **do**
8 Obliviously apply operation op_j to elements of attribute b_j within group i
9 **end**
10 Write the result into $[\![\mathbf{D}]\!]$
11 **end**
12 **return** $[\![\mathbf{D}]\!]$

Figure 13: Aggregation algorithm.

4.2 Deployment

Deployment of this Solution is inherently more complex as there are more participants considered to operate in the system.

Instead of providing a user interface directly, the SQL-controller mimics a popular open source database PostgreSQL[10]. PostgreSQL server and clients use a specific network

[10] PostgreSQL - https://www.postgresql.org/

protocol called the wire protocol[11] to interact. PostgreSQL server has to be contacted through the wire protocol. Likewise, the SQL-controller does not have a direct user interface, and instead it can be used through PostgreSQL wire protocol. This decision was made to avoid writing a lot of language specific SQL interface drivers, such as JDBC and ODBC. PostgreSQL is a widely used database, and therefore already has bindings to use it from other languages. All the bindings end up using the same wire protocol to actually communicate with the PostgreSQL server. By emulating PostgreSQL server in the SQL-controller, client applications of SQL-controller can be written with ease in every language that has existing drivers or bindings for PostgreSQL.

At the core of the deployment are the 3 *compute* parties, which should reside in different administrative domains. These 3 domains can be either trusted or untrusted, as it is only important that no adversary can corrupt more than one. *Compute* parties are running Sharemind application servers.

Every *input* party must secret share its private input data and distribute the shares to 3 *compute* parties. Since the data before secret sharing is unprotected, the *input* parties must be in the trusted domain. *Input* parties can use a simple proxy application like the *CSV importer* from Rmind, or the client application can use the Sharemind Controller library directly to secret share the data and store it on *compute* parties. However, *input* parties can also run the SQL-controller and insert the data using SQL INSERT commands.

It is also possible for the *input* party to be a website visitor. In that case, a JavaScript library will do the secret sharing in the browser, thus not requiring for the user to trust the web server. Deployments with inputs coming from the browser are further discussed in [T16].

In the *result* parties there is a SQL-controller that processes the query and uses Sharemind Controller library to secret share the query parameters and remotely execute the query on *compute* parties. The SQL-controller imitates PostgreSQL server, so client application can use existing drivers (JDBC, for example) which are meant to be used with PostgreSQL server. It should be noted that only a subset of the PostgreSQL commands is actually supported.

A picture describing the deployment scheme can be seen in Figure 14. Note that, in general, there is no limit on the number of *input* and *result* parties. There must be, at least, one input party using the Sharemind Controller library directly. On three separate domains, there are the 3 Sharemind servers. The *result* party uses JDBC driver to talk to SQL-controller, which runs the MPC protocols on the *compute* parties to obtain query results.

This solution uses the Sharemind framework which is written in C/C++. Employing the communication middleware developed in the context of WP1, makes a Java API available for usage, by introducing an extra layer of indirection and requiring additional integration, but also allows the establishment of secure channels in a modular way. Calling Java from the C/C++ application is possible. However, Sharemind already has a very complicated network stack which can also be used to instantiate secure channels with framework-specific optimizations for low latency and high throughput. There

[11] PostgreSQL Frontend/Backend protocol - https://www.postgresql.org/docs/9.4/static/protocol.html

might be a trade-off between additional security provided by the communication middleware and performance offered by Sharemind's existing native network stack, which is something to consider depending on the SafeCloud application.

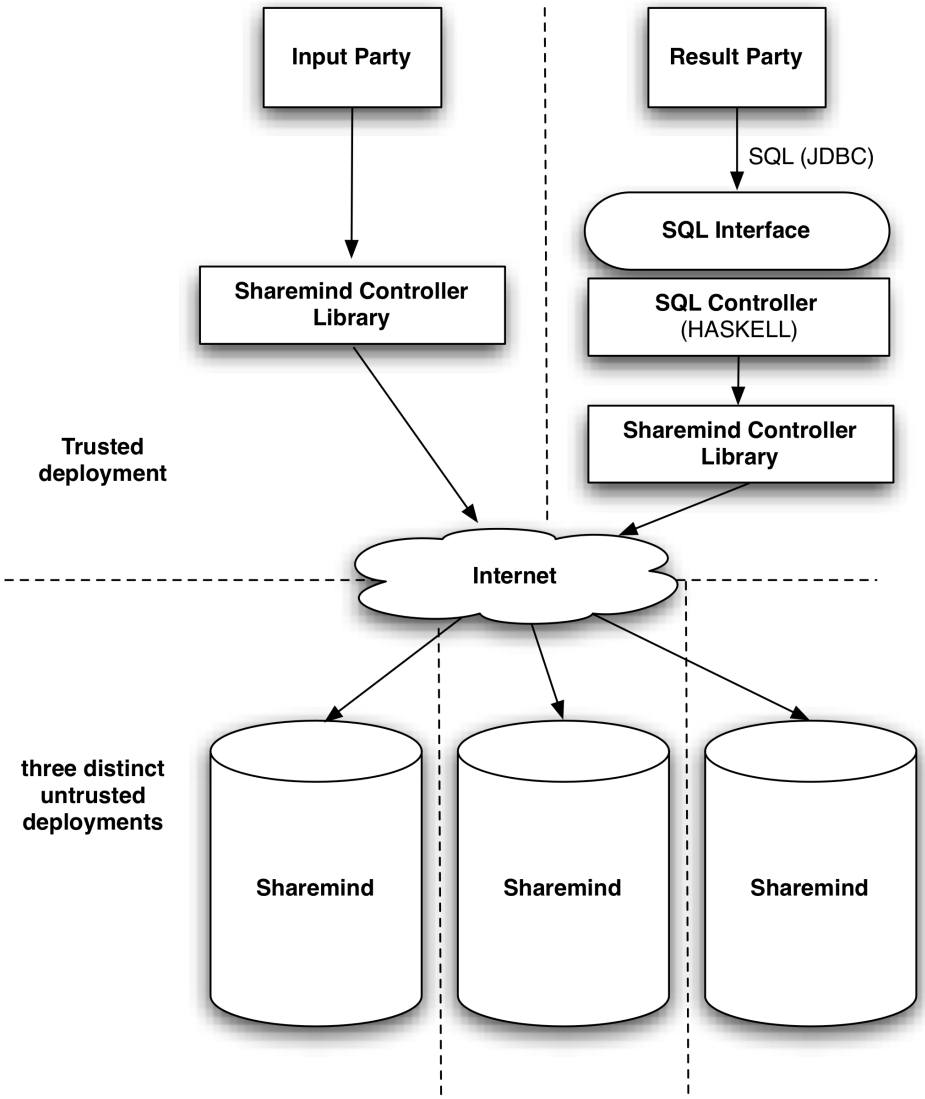

Figure 14: Deployment of Solution 3 showing different administrative domains.

4.3 Discussion

One considerable difference between this Solution and the previous is that Solution 2 allows some of the processing to be done over plaintext (in the trusted environment). Note that this approach considers multiple trusted domains (not necessarily trusting each other) interested in keeping their data inputs private, while computing over the whole aggregated set of data, which encompasses other sensitive information of the client. To address this issue, Solution 3 ensures that only aggregations of data will be converted back as plaintexts to respond to remote data querying. These differences in domain and requirements directly imply that this Solution cannot be offered as a direct alternative for previous scenarios.

One of the benefits of this Solution is its application in the popular demand for secure queries over joint collections of sensitive data. This has been the main focus of European

research projects such as PRACTICE[12] and, more specifically, SafeCloud. This also fits the required criteria for the MaxData analytical use case described in Deliverable D5.2. The scenario describes multiple hospitals (*input* parties) providing patient data as sensitive inputs to the system, which, in one hand, must be stored securely and confidentially and, on the other hand, have to be available for query computation for the identification of epidemics. Solution 3 guarantees security properties for information privacy regarding patient data, while allowing statistical queries to be performed on this data.

Alternative solutions to the query processing of joint databases exist in the literature. Aircloak[13] employs differential privacy techniques, but requires all parties to trust the Aircloak server, which is similar to a TTP scenario, and not consistent with SafeCloud's approach to untrusted remote deployments. Other solutions for providing these functionalities by employing MPC can be listed as potential alternatives [CMF+14, RSH+14], and some have even been shown to be feasible for real-world applications [D+15]. Using Sharemind allows us to take advantage of a framework that has also been shown to be feasible for practical applications using the Rmind tool [BKK16], and therefore validating the basis of this approach.

One limitation associated with this third Solution is the disability of querying individual data items, from the domain-specific consideration that multiple *input* parties provide sensitive data and have individual demands for confidentiality. When considering a single *input* party, this limitation can be safely removed without affecting privacy, and this can be offered as an alternative instantiation to the problem of Solution 2, as we now have a single trusted deployment.

[12] https://practice-project.eu/
[13] https://www.aircloak.com/

5 Conclusion

The first deliverable of WP3 presented and detailed the unifying general architecture of SafeCloud, alongside three main solutions for enabling privacy-preserving storage and computation. This deliverable directly follows that line of work by describing the security models allowed by the several solutions of SafeCloud, detailing the available privacy-preserving techniques compatible with said models, and specifying how these techniques can be instantiated seamlessly in the SafeCloud general architecture.

The first Solution maps a standard cloud deployment scenario of a single trusted client outsourcing computations to a single untrusted cloud provider. This allows the usage of standard encryption schemes, providing excellent security, but disabling computation functionalities. Alternatively, this Solution also allows the deployment of cryptographic schemes that enable specific computations over encrypted data, such as equality comparisons and range queries. This leads to a more complete usage of the superior computational power of the cloud provider, by sacrificing security and performance at different levels.

The second Solution adds the possibility of having multiple untrusted cloud providers. This does not prevent the usage of any of the aforementioned schemes, while expanding the available techniques to MPC schemes that make use of a more complex trust model. This further allows the execution of a broader scope of computations over data, whilst providing stronger security guarantees for the sensitive stored data, under wider variety of trust models.

The third Solution allows multiple local users to upload their data to a collective distributed untrusted environment. This allows the execution of more ambitious processing algorithms that require significant amounts of data to extract useful analytics. By enabling MPC algorithms to be executed over this joint database, we have guarantees that the information obtained by the clients requesting data queries does not include private inputs, which are kept secure at all times.

The versatility of the proposed techniques is helpful in the context of the SafeCloud framework, as the general architecture allows the existence of a multi-purpose Proxy on the client side and a customized HBase Coprocessor/Sharemind application server that implements these techniques seamlessly. This plays a central role in the transparency of the implementation with respect to the end-user, as the local proxy is responsible for trusted operations, such as data encryption/decryption or secret sharing, whilst remote deployment translates the requested queries into the application-specific operations.

The provided modularity is crucial to the success of the SafeCloud project for two main reasons. First, it allows the users of a SafeCloud-enabled solution to experience a standard widely used API that remains constant regardless of the underlying techniques, enabling service providers to offer security guarantees based on complex cryptographic mechanisms without sacrificing usability. Second, it allows the deployment of services using this approach to be tailored by following specific fine-grained security and performance requirements of SafeCloud's use cases, as well as expand the scope of the SafeCloud framework towards solving a broader set of real-world problems.

6 References

[AAE+09] Divyakant Agrawal, Amr El Abbadi, Fatih Emekci, and Ahmed Metwally. Database management as a service: Challenges and opportunities. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on, pages 1709-1716. IEEE, 2009.*

[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.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Theory of Cryptography, pages 137–156. Springer, 2007.

[AS14] Ihsan Haluk Akin and Berk Sunar. On the difficulty of securing web applications using cryptdb. In *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*, pages 745-752. IEEE, 2014.

[AWW14] George Weilun Ang, John Harold Woelfel and Terrence Peter Woloszyn. System and method of sort-order preserving tokenization, May 27 2014. US Patent 8,739,265.

[AZ10] Paul Anderson and Le Zhang. Fast and Secure Laptop Backups with Encrypted De-duplication. *LISA*, 2010.

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

[B79] George Robert Blakley. Safeguarding cryptographic keys. Proc. of the National Computer Conference, pages 313-317, 1979.

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

[BCL+09] Alexandra Boldyreva, Nathan Chenette, Younho Lee and Adam O'Neill. Orderpreserving symmetric encryption. In *Advances in Cryptology EUROCRYPT 2009*, pages 224-241. Springer, 2009.

[BCO11] Alexandra Boldyreva, Nathan Chenette and Adam O'Neill. Order preserving encryption revisited: Improved security analysis and alternative solutions. In *Advances in Cryptology-CRYPTO 2011*, pages 578-595. Springer, 2011.

[BCQ+11] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: Dependable and secure storage in a cloud-of-clouds," in Proceedings of the Sixth Conference on Computer Systems (EuroSys), 2011.

[BKK16] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, Riivo Talviste. Students and Taxes: a Privacy-Preserving Social Study Using Secure Computation. In Proceedings on Privacy Enhancing Technologies, PoPETs, 2016 (3), pp 117–135, 2016.

[BKL+14] Dan Bogdanov, Liina Kamm, Sven Laur and Ville Sokk. Rmind: a tool for cryptographically secure statistical analysis. Cryptology ePrint Archive, Report 2014/512, 2014. http://eprint.iacr.org/.

[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.

[BLT14] Dan Bogdanov, Sven Laur, Riivo Talviste. A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation. In Proceedings of the 19th Nordic Conference on Secure IT Systems, NordSec 2014, LNCS, vol. 8788, pp. 59-74. Springer, 2014.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, ASIACRYPT, volume 1976 of Lecture Notes in Computer Science, pages 531-545. Springer, 2000.

[CMF+14] Koji Chida, Gembu Morohashi, Hitoshi Fuji, Fumihiko Magata, Akiko Fujimura, Koki Hamada, Dai Ikarashi and Ryuichi Yamamoto. Implementation and evaluation of an efficient secure computation system using 'R' for healthcare statistics. Journal of the American Medical Informatics Association, 04, 2014.

[CTP04] Joe Cooley, Chris Taylor and Alen Peacock. ABS: the apportioned backup system. MITLaboratory for Computer Science, 2004.

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

[D+15] Ivan Damgård et al. *Confidential benchmarking based on multiparty computation*. Cryptology ePrint Archive, Report 2015/1006, 2015.

[DLR13] Dan Bogdanov, Peeter Laud and Jaak Randmets. "Domain-polymorphic language for privacy-preserving applications." *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*. ACM, 2013.

[DN14] Ivan Damgård and Jesper Buus Nielsen. "Adaptive versus static security in the UC model." *International Conference on Provable Security*. Springer International Publishing, 2014.

[DR+02] John R. Douceur et al. "Reclaiming space from duplicate files in a serverless distributed file system." *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*. IEEE, 2002.

[GB10] Martin Geisler and Joakim Bittel. *Cryptographic Protocols: Theory and Implementation*. Diss. Aarhus University, Faculty of Science, Department of Computer Science, 2010.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. *Proceedings of the fourteenth annual ACM symposium on Theory of computing*. ACM, 1982.

[K+09] Jia Keting et al. "Distinguishing and second-preimage attacks on CBC-like MACs." *International Conference on Cryptology and Network Security*. Springer Berlin Heidelberg, 2009.

[KAK10a] Hasan Kadhem, Toshiyuki Amagasa and Hiroyuki Kitagawa. Mv-opes: Multivalued-order preserving encryption scheme: A novel scheme for encrypting integer value to many different values. *IEICE TRANSACTIONS on Information and Systems*, 93(9):2520-2533, 2010.

[KAK10b] Hasan Kadhem, Toshiyuki Amagasa and Hiroyuki Kitagawa. A secure and efficient order preserving encryption scheme for relational databases. In *KMIS*, pages 25-35, 2010.

[KS85] Voratas Kachitvichyanukul and Bruce Schmeiser. Computer generation of hypergeometric random variates. *Journal of Statistical Computation and Simulation*, 22(2):127-145, 1985.

[KT13] Ralf Kusters and Max Tuengerthal. The iitm model: a simple and expressive model for universal composability. IACR Cryptology ePrint Archive, 2013:25, 2013.

[KVW04] Tadayoshi Kohno, John Viega and Doug Whiting. "CWC: A high-performance conventional authenticated encryption mode." *International Workshop on Fast Software Encryption*. Springer Berlin Heidelberg, 2004.

[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.

[LTW13] Sven Laur, Riivo Talviste, Jan Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In Proceedings of the 11th International Conference on Applied Cryptography and Network Security, ACNS 2013, LNCS, vol. 7954, pp. 84-101. Springer, 2013.

[LW12] Dongxi Liu and Shenlu Wang. Programmable order-preserving secure index for encrypted database query. In *Cloud Computing (CLOUD), IEEE 5th International Conference on*, pages 502-509. IEEE, 2012.

[LW13] Dongxi Liu and Shenlu Wang. Nonlinear order preserving index for encrypted database query in service cloud environments. *Concurrency and Computation: Practice and Experience*, 25(13):1967-1984, 2013.

[LW15] Kamm, Liina and Jan Willemson. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security* 14.6 (2015): 531-548.

[M+08] Norman H. Margolus et al. Data repository and method for promoting network storage of data. U.S. Patent No. 7,412,462. 12 Aug. 2008.

[M05] David A. McGrew. Efficient authentication of large, dynamic data sets using Galois/Counter mode (GCM). In IEEE Security in Storage Workshop, pages 89-94. IEEE Computer Society, 2005.

[M89] Ralph C. Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218-238. Springer, 1989.

[MR11] Ueli Maurer and Renato Renner. "Abstract cryptography." *In Innovations in Computer Science*. 2011.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644-655. ACM, 2015.

[OSC03] Gultekin Ozsoyoglu, David A. Singer, and Sun S. Chung. Anti-tamper databases: Querying encrypted databases. In *DBSec*, pages 133-146, 2003.

[PBP16] Rishabh Poddar, Tobias Boelter and Raluca Ada Popa. Arx: A Strongly Encrypted Database System, 2016.

[PLZ13] Raluca Ada Popa, Frank H. Li and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 463-477. IEEE, 2013.

[PRZ+11] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85-100. ACM, 2011.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In Proceedings of the 7th ACM

conference on Computer and communications security, pages 245-254. ACM, 2000.

[PZB15] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. Guidelines for using the cryptdb system securely. Technical report, Cryptology ePrint Archive, Report 2015/979, 2015. http://eprint. iacr. org.

[R01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE, 2001.

[R13] Phillip Rogaway. http://www.cs.ucdavis.edu/~rogaway/ocb/license.htm, Free OCB licenses, 2013.

[RBB03] Phillip Rogaway, Mihir Bellare and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Trans. Inf. Syst. Secur., 6(3):365-403, 2003.

[RSH+14] Khaled El Emam, Saeed Samet, Jun Hu, Liam Peyton, Craig Earle, Gayatri C. Jayaraman, Tom Wong, Murat Kantarcioglu, Fida Dankar, and Aleksander Essex. A Protocol for the Secure Linking of Registries for HPV Surveillance. PLoS ONE, 7(7): e39915, 07 2012.

[rsystem] R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

[S79] Adi Shamir. How to Share a Secret, Commun. ACM, pages 612-613. ACM, 1979.

[STD+09] Lee Seungmin, Park Tae-Jun, Lee Donghyeok, Nam Taekyong and Kim Sehun. Chaotic order preserving encryption for efficient and secure queries on databases. *IEICE transactions on information and systems*, 92(11):2207-2217, 2009.

[T16] Riivo Talviste. Applying Secure Multi-party Computation in Practice. PhD thesis. University of Tartu. 2016.

[TLS+15] H. Tang, F. Liu, G. Shen, Y. Jin and C. Guo. UniDrive: Synergize Multiple Consumer Cloud Storage Services. Proceedings of the 16th Annual Middleware Conference (Middleware), 2015

[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.

[XYH12a] Liangliang Xiao, I-Ling Yen and Dung T. Huynh. Extending order preserving encryption for multi-user systems. *IACR Cryptology ePrint Archive*, 2012:192, 2012.

[XYH12b] Liangliang Xiao, I-Ling Yen and Dung T. Huynh. A note for the ideal order preserving encryption object and generalized order-preserving encryption. *IACR Cryptology ePrint Archive*, 2012:350, 2012.

[Y82] Andrew C. Yao. Protocols for secure computations. *Foundations of Computer Science, SFCS'08. 23rd Annual Symposium on*. IEEE, 1982.

[YKK12] Dae Hyun Yum, Duk Soo Kim, Jin Seok Kim, Pil Joong Lee and Sung Je Hong. Order-preserving encryption for non-uniformly distributed plaintexts. In *Information Security Applications*, pages 84-97. Springer, 2012.

[ZYT+15] R. Zhao, C. Yue, B. Tak and C. Tang, "SafeSky: A Secure Cloud Storage Middleware for End-User Applications," in IEEE 34th Symposium on Reliable Distributed Systems (SRDS), 2015.