Querying Always-Encrypted Data on Cloud Databases

Guilherme Rosas Borges*
Advisor: Henrique Domingos

DI-FCT-UNL, NOVA LINCS Research Centre

Extended Abstract. In this paper we propose a system that uses partial homomorphic encryption on SQL cloud databases to allow querying remote encrypted data, providing, at the same time, attestation features and a proposed extension of SQL to support multimodal queries. For these purposes we analysed the CryptDB system, on which we based our own system. With the support from CryptDB it is possible to have permanent data encryption in a remote database, while still being able to execute normal SQL queries over this data, with almost no change to existing applications.
In our system we also include support for remote attestation, basing our architecture on the TPM 2.0 standard. Therefore, our system provides the ability to determine that computations running remotely on cloud servers, in services like DBaaS (database as a service), are trustable. Furthermore, we analysed a content-based image retrieval (CBIR) algorithm working over an encrypted repository, to support searchable encryption using image proximity on colour features.
Our system consists of three main components: server, proxy and client. The server is totally located on the cloud, the proxy is client-side. Clients of this system interact with the proxy, which in turn communicates with the server. This server is based entirely on the cloud, taking advantage of deployable services (DBaaS) when possible.
We guarantee protection against two main threats on cloud servers: honest-but-curious (HbC) attackers, i.e. passive attackers snooping on private data; and active attackers, who could tamper with data and code on these servers. Protection against data tampering is also provided on the client-side proxy. Finally, we also guarantee security between the three components of the system using standard communication protocols, like SSL and HTTPS.
We implemented a system based on our reference architecture, using both local machines and a cloud service provider. We tested the functionality of our system and gathered information on the comparative performance of the system's components, conducting a critical analysis of the overall implementation, as well as a throughput analysis of CBIR when querying images by proximity. With the data from our test benches, we advance some possible ideas of future work based on our analysis.

Keywords: privacy, cloud services, homomorphic encryption, MIE ciphers, remote attestation

* This work is partially supported by the NOVA LINCS Research Centre under the context of a grant supporting introduction to research activities in computer science.

1 Introduction and motivation

Nowadays, many different platforms are following a trend to move their computations and storage to cloud-based services, more commonly services like IaaS (Internet as a Service) - like Amazon AWS [1], but also PaaS (Platform as a Service), of which Microsoft Azure [2] offers several examples, and SaaS (Software as a Service) solutions, like Google Docs [3]. These services bring many advantages, like those of constant availability and ubiquity, but they also raise some new concerns. With local servers, companies and users could guarantee the safety of their data by keeping it from the outside world. However, by moving data and computations to a third-party cloud server, users lose control of their data, thus raising the fear of third-party data theft or tampering. This is also a potential concern when we migrate owned previously unprotected DBMS solutions to currently available DBaaS environments on the cloud, like Amazon RDS [11] or Google Cloud [12].

Possible clients of services like DBaaS are aware of those problems, citing the lack of guarantees on privacy and confidentiality as the biggest deterrents when adopting these services [4, pp. 45-47]. Availability and integrity were considered to be very important factors as well. These concerns can be classified into three main categories: confidentiality, reliability and trustability concerns. These are the main problems we addressed in the system developed, as conjugated criteria in a dependable DBaaS solution.

1.1 Problem definition

As we stated before, data confidentiality and integrity guarantees are not offered by current cloud providers. Following these concerns, we need to guarantee data confidentiality against honest-but-curious system administrators and other attackers on the cloud. Current approaches on database encryption (like [5]) usually require all computations to be executed on the client side (an approach usually known as a "security on the rest approach"). This solution is not particularly interesting for on-line applications managing larger databases, because it introduces a high performance penalty and high latency conditions in data transferring between the client and the cloud. Other assume secure servers [6], while some implement some kind of homomorphic encryption ([7], [8], [9] and [10]). Most of these approaches, however, require heavy encryption operations to be executed on client applications, which may difficult its use in mobile or other devices.

However, attacks are not only directed towards gathering plain-text data. Encryption does not prevent other types of attacks, like inference [18], and also does not prevent data tampering on databases. Furthermore, encryption cannot guarantee that the server's integrity is maintained while computations are being executed. Active attackers can manipulate results being returned to the client without its knowledge, even when data is encrypted. Remote attestation modules, for example the state-of-the-art TPM 2.0 specification [13], can be used to prevent these attacks and assess the server's integrity.

Finally, we considered new types of multimodal searches to be added to SQL-based DBMS. SQL only supports equality searches on multimedia like images or videos. Common search algorithms based on content similarity already exist, albeit not over encrypted repositories (with the exception of algorithms like the MIE/CBIR we analysed [15]). This kind of search operations are directly related to existing content-based information retrieval solutions on unencrypted data.

1.2 Objectives and contributions

To solve the problems stated in the previous section, we propose a system which provides protection against attackers on the cloud while using services, like DBaaS, with almost no changes. This can be achieved using partially homomorphic encryption. In our proposal we leverage the solution from the CryptDB approach [7].

To provide trustability and prevent active attacks against the cloud, our system includes attestation provided by a TPM 2.0 chip. Therefore, we can attest and trust the remote cloud service while computations are running. Both the server and the proxy are lie on this trusting computing base (TCB).

We integrated an encrypted CBIR algorithm to our system, allowing to search images based on similarity ([15] and [16]). This is integrated as an extension to SQL (extSQL), allowing users to search the image repository transparently. The extSQL queries allow for image-lookup operations and are of the form `SELECT * FROM images WHERE IMAGE ~<image_bytes>`, where *image_bytes* represents an array of bytes from an image.

1.3 Paper organization

The remaining of this paper is structured in the following way: section 2 presents a reference system model and architecture for our proposal, and defines the adversary model conditions we consider for our security solution; section 3 is dedicated to the implementation and the developed prototype; section 4 presents the experimental observations obtained and the related test-bench environment created for the evaluation; section 5 briefly describes the context of the designed solution, comparing it with relevant related work references; finally the section 6 concludes the paper.

2 System model and architecture

The system we designed is divided in three main parts: multiple clients, a proxy and a server. In our reference architecture, the clients are all connected to a proxy in a local network, while the server is located in a cloud environment. A diagram of this architecture is presented in Fig. 1.

Communication is started by clients, as their application issues extended-SQL queries (extSQL). These queries comprise a subset of normal SQL and the new extSQL query, designed to allow image search. The subset of SQL queries

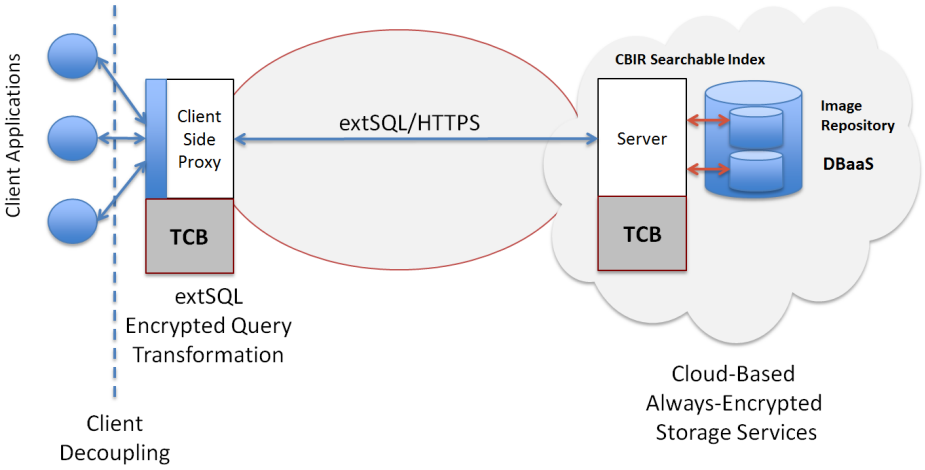

Fig. 1. Architecture adopted for our system. Its details are explained in Section 2.

supported is that of CryptDB, which is limited by the existing partial homomorphic encryption schemes, as described in the CryptDB paper [7].

2.1 Server

The server is composed of three main components: a dispatcher, a CBIR server and a database management system (DBMS)[1]. The dispatcher receives, processes and answers to requests sent from the proxy, interacting in-between with the other components.

The CBIR server [15] receives requests from the dispatcher, in the form of an encrypted image. Its algorithm searches for similar images in the image repository, present in the server's local storage. Images deemed similar are returned encrypted to the dispatcher.

The DBMS is an unmodified MySQL system, provided as-is by a cloud-based DBaaS. While no changes have to be made to the DBMS, the cloud service provided must support TPM attestation.

The dispatcher is responsible for receiving queries from the proxy. It analyses whether these are normal SQL queries, in which case they are sent to the DBMS (MySQL), or extSQL queries. In the later case, the dispatcher separates the elements of the query, sending SQL parts to the DBMS, and processing image requests, then sent to the CBIR server. Both the DBMS and CBIR server send their answers to the dispatcher, which encapsulates the answers in JSON and returns them to the proxy.

All elements of the server support attestation. The dispatcher receives attestation requests and returns a quote, which contains PCR values of software

[1] Despite that we can conceptually use any DBMS (queryable by SQL) in our reference model, we used MySQL for our implementation prototype.

loaded on the server, namely all the server's components. PCRs are written at boot time, using tools provided by the TPM to load code into memory.

2.2 Proxy

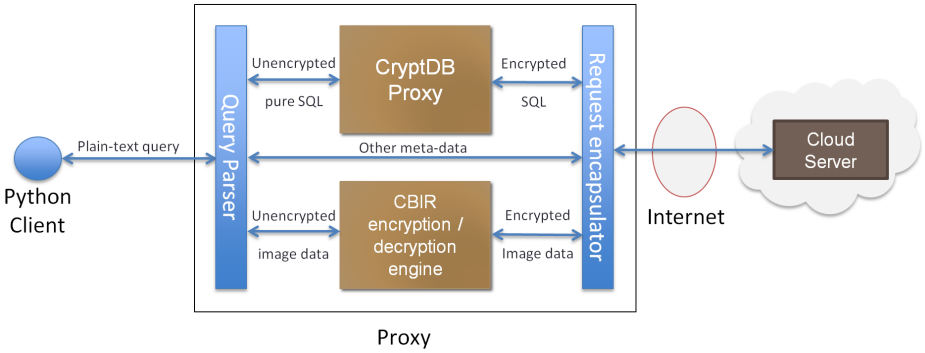

Fig. 2. Proxy's architecture. Queries from the client are first processed by a parser, which sends pure SQL parts to the CryptDB proxy, and image related queries into the CBIR algorithm. Encrypted queries are then rejoined to be sent to the server. When receiving a response from the server, this process is reversed for decryption. More details on the proxy are explained in Section 2.2.

The proxy, whose architecture is pictured in Fig. 2, is located on the client side, and is responsible for receiving client requests, encrypting them and communicating with the server's dispatcher. When receiving a request from one of the clients, queries are sent to a parser. Normal SQL queries are sent to CryptDB's own proxy [17], which takes a normal, plain-text, SQL query and encrypts it according to CryptDB's specification. Image-related extSQL queries are processed by the CBIR algorithm.

After processing from both components is done, queries are rejoined by the request encapsulator, which codifies information in the JSON format. When receiving a response from the server, the encapsulator splits information between CryptDB's proxy and the CBIR algorithm as needed. After that, plain-text data is re-aggregated in JSON to be sent to the client.

The proxy can also process meta-data requests, which are related with attestation and server status. These requests do not contain sensitive data, therefore there is no need to encrypt them.

Proxy's deployment In the development of the reference architecture, three possible deployments of the server were considered, which are described in this subsection.

In the first one, the proxy is in the same machine as the client application. This deployment allows for the lowest latency between these components, while also eliminating possible concerns of data snooping on the local network. However, this model requires a desktop or laptop computer, as a mobile environment does not provide neither the needed architecture for the proxy nor the performance requirements for encryption and decryption operations.

In the second model, which was adopted in our implementation, the proxy is located on a local machine, allowing for local clients to connect to it. In this case, security of the communications on the local network must be assured (as stated in Section 2.5). The main advantage of this model is to allow for client decoupling, i.e. total separation of client and proxy operations, allowing for clients in different mobile and desktop environments.

The third approach is a variation of the second, where the proxy is located on a cloud environment (different from the server) instead of a local network. In this case, the need for a local dedicated machine is eliminated, however, the cloud where the proxy is located must be dependable, i.e. the adversary model would not include, for example, honest-but-curious system administrators. Nevertheless, this last model could provide better latency, as some intermediary communications are done between the proxy and server, and only the final result is sent to the client. In this case, having the proxy and server geographically close could allow for lower latency and thus better performance.

2.3 Client

The client component consists of the final application to be utilized by users of the system. In our case, the client is developed in Python and simply issues extSQL queries to the proxy and receives responses, presenting them to the user, while registering metrics of performance and latency. On first communication with the proxy, a login meta-request is issued to the proxy, which in turn asks for a TPM quote of the server. The response is sent to the proxy, which checks whether the server is trustable, i.e. that the PCR values are as expected. After the client is informed of the server's integrity, it issues another attestation request, directed only to the proxy, to also attest this compenent. Finally, it continues its normal operation, issuing requests if it deems all the system's components are trustable.

As the client only receives processed requests, its code is very simple and lightweight. We designed the client to be executable not only on a desktop environment, but also on mobile tablets and smartphones (Android). Due to the chosen architecture, all of the overhead related with encryption and decryption operations is allotted to the proxy, thus processing in the client is only that of sending and receiving data (with TCP) and displaying its contents to the user. Therefore, our system is oriented towards thin clients, which is particularly useful for applications running on resource-constrained devices.

An additional GUI to Android was also designed and implemented, supporting both normal SQL and extSQL queries with image support.

2.4 Communication

Communication between the client, proxy and server is done in TCP, encapsulated on a SSL layer [14]. Therefore, communication between components is proofed against man-in-the-middle attacks. This provides another layer of security, as data passed between the proxy and server is encrypted already before being transmitted in the internet.

Data passing between the client and proxy is plain-text before SSL encryption, however, to gain access to sensitive data, an attacker would have to gain access to the local network - assuming a client-side proxy, where all components are on the same network.

Data transferred between clients is formatted in JSON, which we considered to be both lightweight and cross-platform, as the system comprises components ranging from C to Python and Android. A simple protocol was devised, which integrated both SQL-like and image-related elements in its structure. Images are passed between the components in Base64.

2.5 Adversary Model

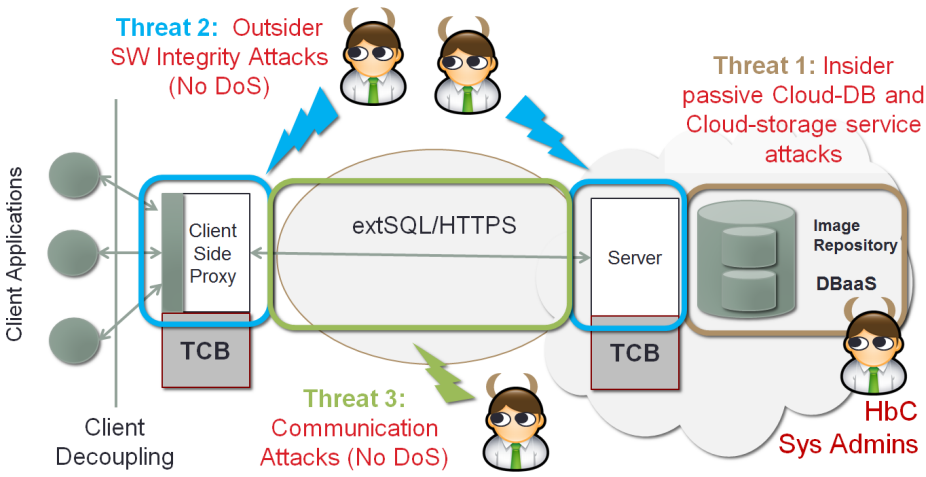

Fig. 3. Our system provides protection against three different types of adversaries or threats. The first one (brown) is an honest-but-curious system administrator, with access to the cloud servers. The second (blue) comprises outsider attacks on software integrity, both on the server and the proxy. Finally, the third (green) one is a man-in-the-middle type of attacker.

In the following subsections we present the three types of adversaries included in our adversary model, pictured in Fig. 3.

Threat 1: Honest-but-Curious System Administrators This adversary is any user with access to cloud servers, typically a system administrator. This kind of attacker only snoops on the cloud servers, never interfering with computations or altering data. Therefore, data encryption on the server will suffice to prevent against this kind of attack. As our system uses partially homomorphic encryption schemes to encrypt data sent to the server, plain-text data is never revealed to the server, and thus can not be pried on.

This kind of attacker is the same described by CryptDB [7], as the relevant mechanisms used in our system are the same of CryptDB.

Threat 2: Outsider integrity attacks on software This adversary can attack both on the proxy and the server and, unlike the previous threat, it is an active adversary. These attackers can change computations and deployed software on real-time to induce false and different results from the ones expected. By using a TPM 2.0 module, we can assure these attacks can be identified and stopped.

A TPM 2.0 chip guarantees a trusted computing base [18] on which both the proxy and server lie on. On boot, the TPM extends its PCR registers with hashes from software loaded into memory. After boot, these registers contain signatures from the operative system and our system's software. To attest the server's and proxy's integrity, the client issues a quote request to both these components. With the received answer, the client can judge whether the software loaded into these machines is trustworthy or not. If not, communications are stopped until a reboot of the affected machines is made, therefore loading the software into memory again, initiating the process described all over again.

Threat 3: Communication attacks These kind of attack is similar to a man-in-the-middle attack, where the attacker sniffs data transiting on the internet, between the proxy and the cloud server.

Prevention against eavesdropping is first guaranteed by only sending encrypted data to the cloud. As data is always encrypted after leaving the proxy, both by CryptDB's mechanisms (regarding SQL traffic) and CBIRs', attackers gain no information from capturing these packets. Moreover, usage of the SSL protocol over the TCP traffic also guarantees another layer of encryption of transiting data.

Exceptions to data encrypted by CryptDB or CBIR are quote requests. Although these requests also pass through an SSL layer, they are not further encrypted. A replay attack could be made if an attacker got hold of a quote answer sent from the server - a legitimate answer could be sent over and over again, masking the true state of the server. However, the quote system [19] used by us includes a nonce on each request, whose answers are then signed by the TPM. Therefore, the client can attest the answers they receive are the legitimate ones to the quotes requested.

3 Implementation and prototype

In our implementation of the system we followed a similar architecture to that described in the previous section, although with some changes due to implementation constraints. In the current version, the server's dispatcher only receives and answers to attestation requests. All other communications from the proxy to the server are directly issued to the DBMS and the CBIR server. Therefore, the proxy described previously on Section 2.2 does not need a request encapsulator to the server, although answers to the client are still codified as was described.

Due to constraints related to mobile clients, and the fact that TPM emulators [20] are coded in C and strongly depend on their implementation, quotes issued from the proxy could not be attested by mobile clients. Therefore, we did not implement a TCB on the proxy, as the client would not be able to take advantage of it. Should it be implemented, a process similar to what is done between the server and proxy would be followed. The implementation of TPM tools on different platforms is one of the open issues we identified on Section 6.1. Due to the fact that TPM 2.0 emulators and tools are still on development, the specification and implementation we used, for test purposes, was that of version 1.2.

To implement our system in a cloud environment, we used Google Cloud services. The CBIR server and attestation dispatcher were deployed on a Ubuntu virtual machine, while the DBMS was deployed on a MySQL DBaaS. To support CryptDB's system, we had to install a library [21] to support some encryption schemes as per CryptDB's specification.

4 Experimental evaluation

In this section we present the experimental evaluation conducted to test and assess the performance of our proposed system. In this evaluation we considered different metrics, namely: latency, execution of SQL queries, homomorphic encryption overhead, CBIR's throughput. For these observations we used different test bench environments, from which we then compared the obtained results. The used test bench configurations are the following: local environment (both server and proxy on the same machine), with an encrypted database and then an unencrypted one; cloud environment, with a remote server, both with encrypted and unencrypted databases too.

We then assessed the latency of overall image operations on extSQL queries. Finally, we present CBIR's algorithm throughput when adding images to the server.

4.1 Experimental test bench

To create the experimental test bench we implemented an environment with the following reference architecture: a laptop running Ubuntu 12.04 with a 2-core Intel i5-3230M running at 2.60GHz with 8GB of RAM. On local tests both the

proxy and the server components ran on the laptop, while only the proxy was local on cloud tests. Clients were ran both on the laptop and on an Asus Fonepad 7 (Android).

The cloud provided selected for remote tests was Google Cloud, with a DBaaS provided by Bitnami. In our research, we found that CryptDB requires cryptographic user-defined functions to be present in the DBMS, and this was the only provider found allowing for such functions to be defined in a DBaaS environment.

To set up the cloud environment, a MySQL DBaaS was deployed and the cryptographic libraries were added to the DBMS. To accomodate both the CBIR and the server dispatcher for attestation, a Ubuntu 12.04 virtual machine was also deployed. Due to different architectures, a TPM emulator could not be installed in the DBMS virtual machine. However, its usage would be trivially the same as in the CBIR virtual machine.

The client ran on the local machine logging all the relevant metrics. From our tests, running the client on a mobile environment bore no additional overheads, as there is almost no processing involved in this component.

4.2 Experimental results

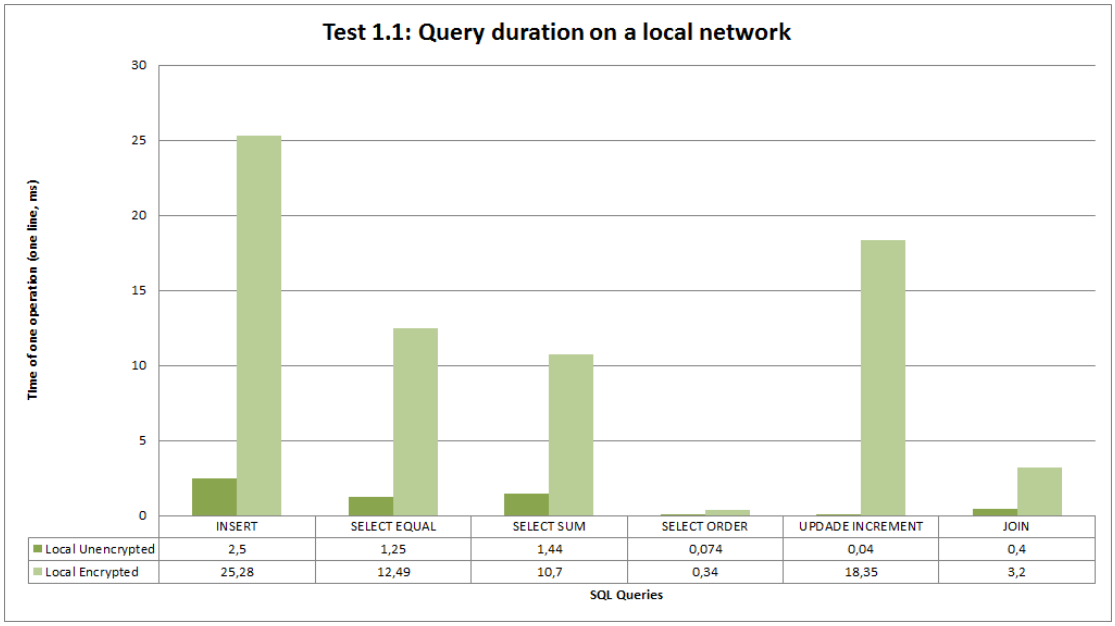

Fig. 4. In this graph we compare the performance (time between issuing a query and receiving the answer) of issuing different types of SQL queries on the system. In this case, all the system's components were located in the local network.

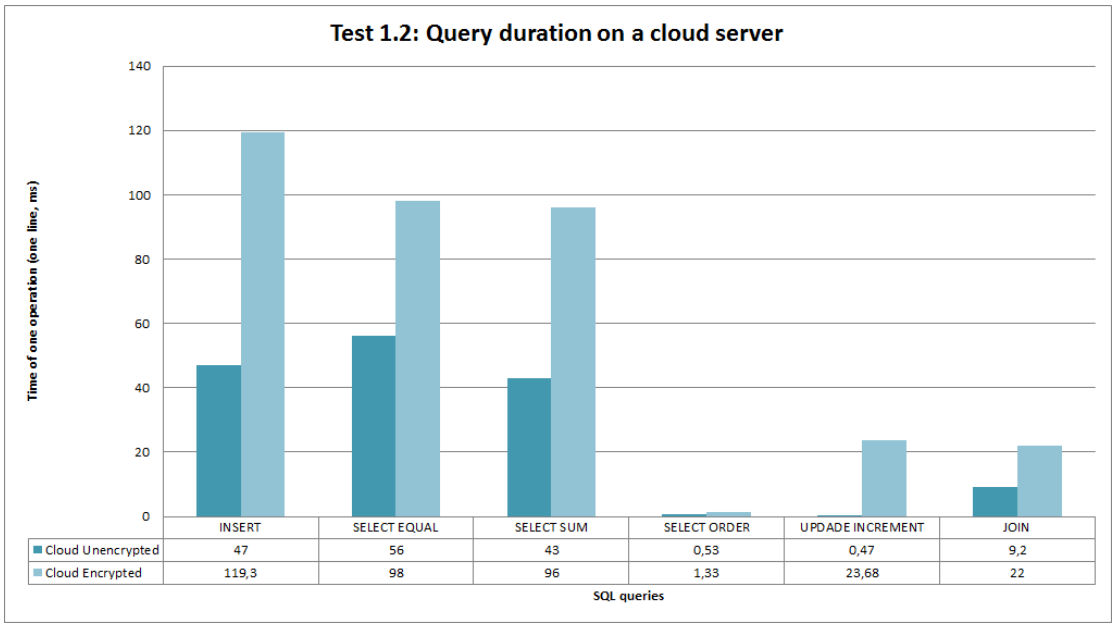

Fig. 5. In this graph we compare the performance (time between issuing a query and receiving the answer) of issuing different types of SQL queries on the system. In this case, the proxy and clients were located in the local network, while the server was located on a cloud provider.

Test 1: local and cloud databases Our first batch of tests focused on the performance of the encrypted MySQL database. As previously stated, we took three variables into account: whether the server is located on a local network or on the cloud, whether encryption is being used and which SQL operations were tested. We focused on the following operations: INSERT, SELECT (with no restrictions), SELECT with SUM, ordered SELECT, SELECT with JOIN, and UPDATE to increment a column.

These operations were chosen so we could observe some particular aspects of the CryptDB implementation (in addition to the expected latency increase), specifically the overhead introduced by the Paillier cryptosystem in INCREMENT queries, and the performance penalty resulting from changes in onion layers, which are described in the CryptDB paper [7].

Tests comprised operations on a thousand rows of data. Some queries, like JOIN, involved a smaller number of rows. To present a comparable metric, we averaged all results and, for each query, we divided the time each query took by the number of rows it focused on. The results are presented in graphs in Figs. 4 and 5. We separated the local network and cloud test-benches, as not all conditions were the same on both, eg. our hardware was different from the cloud's. We verified that encryption adds some overheard to the operations on both environments. However, we considered this overhead is not significant, taking into account that all each query's time is in the same order of magnitude (or similar), both for its encrypted and unencrypted form. In a cloud environment, this overhead is diluted, as we observed in Fig. 5. We attributed this difference mainly to internet latency, as a part of the time each query takes is due to travel time inherent to the internet. We measured latency both on the local network and on the cloud. We observed that, on average, a query request and respective answer had a RTT of ∼10ms on the local network, and a unreliable RTT on the

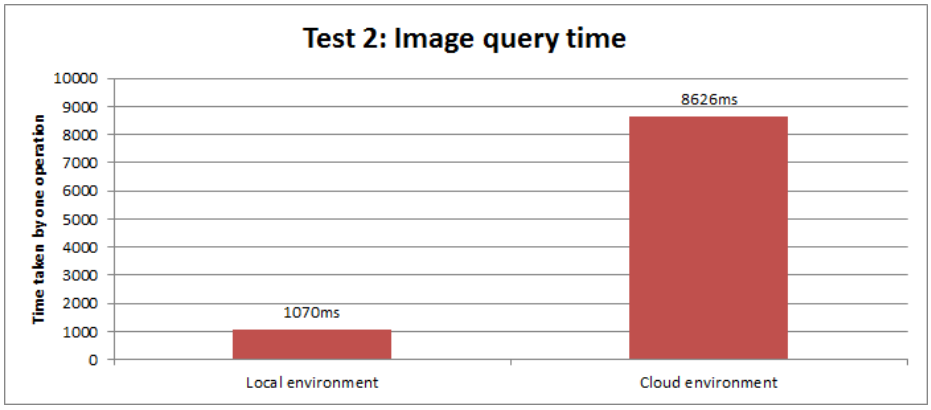

Fig. 6. In this graph for test 2 we assess the time taken for image operations, both on local and cloud environments. We concluded that an image query's time is independent on the repository size.

cloud, comprised in the range of 40∼100ms. Latency could not be subtracted from the times presented in the cloud servers, as those results were not obtained directly, instead we measured the time each operation took and divided it by the number of rows it focused on. Therefore, latency is distributed across all rows accounted for in each operation in the graph of Fig. 5.

Test 2: image queries Our second batch focused on image queries, which use both the DBMS and the CBIR server. These queries send an image file from the client to the proxy. The proxy searches for that particular image ID in the repository's index, contained in the DMBS. After that, a search for similarity is requested to the CBIR server. It then returns the four most similar images, which are then sent to the client. In this test we used the encrypted environments only, as we had no comparable searchable image-proximity algorithms to compare to. Our system was configured to return, for each image search issued by the client, the four most similar images as deemed by CBIR.

In a local environment, the duration of a query was, on average, 1070ms. In a cloud environment, an average query took 8626ms. We tested queries with repositories of different sizes (1, 5, 10, 50, 100, 500 and 1000 images), assessing that search time is independent from the size of the repository.

Test 3: throughput of CBIR Our third batch focused only on assessing the throughput of adding new images to the CBIR repository on the cloud environment. We issued consecutive requests to the server, with 1, 5, 10, 50, 100, 500 and 1000 images. On average, our images had 120kB in size. We then divided the time each query took by the size of the images sent, thus assessing its throughput, as presented in Fig. 7. We concluded that time increases linearly with the size of each request, thus making the throughput constant and independent of the size of both the request and the repository.

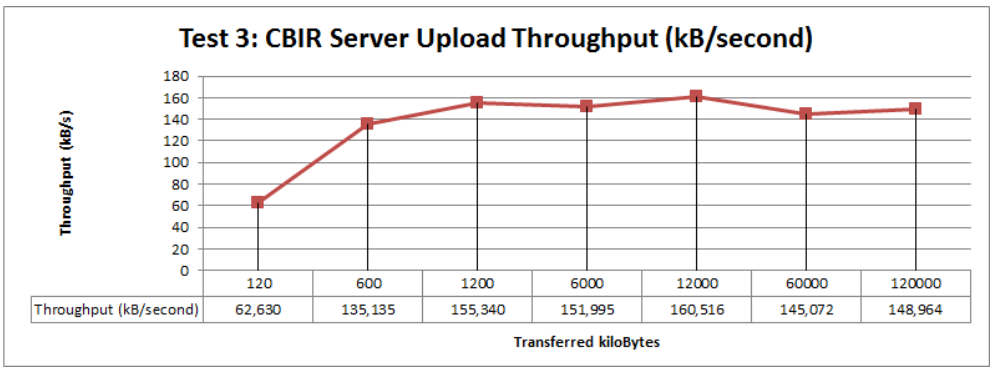

Fig. 7. In this graph for test 3 we assess CBIR's throughput in regards to image upload to the repository. We concluded that its throughput is virtually constant, and therefore independent from the size of the request and of the repository.

4.3 Critical analysis

In regards to the first test, we observed that on a cloud environment the overhead of encryption is not as high as in the local network (an increase of, on average, about 2 times, compared to increases of 5-10 times on a local environment). To obtain comparable metrics, we would need to have the same machines both on the local network and on the cloud, and we would have to measure latency in all tests. As these variables were unaccountable for, we did not compare directly the local and cloud tests. However, we considered the overhead introduced on both situations is acceptable.

The lack of an unencrypted version of a similar image-proximity search algorithm did not allow us to have a point of comparison in the performance of CBIR and while we can assess its performance, we can not comment on the overhead its encryption algorithms introduce in the processing time.

5 Related work and critical analysis

CryptDB [7] was the main contribution to the work we presented. CryptDB acts as a middleware between a user application and a database management system. Their system intercepts any query from a regular application with its proxy, encrypting and only then sending it to the DBMS, hence never revealing the plain text data to a third-party server, in our case a cloud DBaaS. Our system was designed in a very similar vein to this. Honest-but-curious attackers are also the main kind of attackers CryptDB protects against. Although CryptDB's contributions proved to be very noteworthy, they add no further functionality to that of regular SQL, providing only homomorphic encryption to current systems.

Cipherbase [8] is the second work in which we based our research. Similarly to CryptDB, it provides a solution with always-encrypted data on a remote server, also addressing the issue of trustability. This system uses FPGAs to execute crucial computations and attest that the system is not under an active attack. However, the usage of their system implies expensive changes on servers (addition of FPGAs to current hardware), and whose performance is lower than that of specialized hardware. We considered the use of TPM modules, as they provide more functionality, are more adequate and easier to implement for the reasons we will now describe. First, all computations are still made on current hardware, and the only change needed is the addition of the TPM chip to current machines. This leads to minimal changes on software also - while Cipherbase needs deeper changes to current software, in order for the DBMS to accommodate computations made on the FPGA - our system needs no changes to the DBMS software whatsoever. We only need the addition of a small software to handle quote requests for the TPM, which operates independently from the DBMS. Finally, usage of current unspecialised hardware allows for cloud providers to update their machines without needing to redesign the FPGA's software, thus allowing for greater performance.

Other considered systems, like AWS GovCloud [6], assume security on the server-side, thus relying on the users' trust of the provider to work. As stated

before, we assume that the provider can both snoop and tamper users' data on the server, allowing for a secure deployment of servers in any kind of untrusted provider. Systems like Monomi [9] or TrustedDB [10], while implementing homomorphic encryption, do not provide further security (like attestation) to existing database management systems, besides only providing the same set of functionalities as current SQL databases.

We would also like to mention the work around the MIE/CBIR algorithm ([15] and [16]). The work on image similarity was integrated in our system as part of an extended SQL, which allows for users to transparently search both normal databases and image repositories, which is non-existing in current DBMS implementations. Further developments both on homomorphic encryption and search over encrypted multimodal data would allow for enhancements in our system to also be made.

5.1 Our contributions

In our system we also added some state-of-the-art features, not present in the related work mentioned, like remote attestation capabilities. While Cipherbase [8] incorporates trusted hardware, it does not provide versatile attestation capabilities. With a TPM chip, we can attest any component in our system and infer its integrity, without the need of specialized hardware other than the TPM chip, which can integrated on motherboards and mobile devices. Our focus on cloud-based services is also an enriching aspect of our system: the architecture we conceived was centred on these new services, and the components developed were designed to be easily deployed on these environments.

Other feature included in our work was the addition of image-based queries, not supported on SQL. While our related work focused solely on SQL operations ([7], [9], [8] and [10]), we included a new feature, implemented as a Java JCE by [16], thus enabling its use in different systems, as any other encryption algorithm in Java.

We also tried to provide a lightweight as possible solution on the client side. With the use of a proxy, we implemented a cross-platform end-user application with no encryption processing involved. All the systems we referred previously do not provide a separation between the cryptographic engines and user applications, which we tried to implement, as to provide a solution as adaptable as possible, ie. whose components are decoupled and easily integrated with other platforms and existing applications.

6 Conclusions

In this paper we analysed, proposed and implemented a system that allows querying of encrypted data on cloud-provided services. This system allows to store encrypted data in cloud servers, making it insusceptible to data leaks, while allowing computations on this data, therefore providing better performance than a security on the rest approach.

Our system also provides the ability to search over multimodal data, namely text and photos, while being based on a normal DBMS. Furthermore, we provided a way of guaranteeing trustability of the overall system, thus detecting and preventing active attacks. In the following subsections we debate some issues left open and future research ideas that could be taken for further development on our system.

6.1 Open issues

In our implementation, we left some of the aspects of the designed system to be implemented. The trusted computing base to be introduced in the proxy was left out due to compatibility issues described in section 3.

We also did not implement a communication protocol as described in the system architecture. Due to technical constraints of the CryptDB proxy, we separated communication with the server in two moments: database access and CBIR search. Therefore, image queries have to travel twice between the proxy and server, and the respective answers have to be aggregated in the proxy, instead of the server's dispatcher. We believe that the reference architecture would be more efficient, as processing would be more split between components, also reducing latency penalties due to communications.

Another line of work that could be followed to improve the current system would be to add proxy replication. In that way, clients from different networks could share the same data without using the same proxy - which was designed to be mainly local. To achieve this, a key-sharing mechanism would have to be developed, while also providing data consistency across the system.

6.2 Future research trends

One of the aspects overlooked is the possibility of database inference, i.e. studying the patterns of data and queries executed on the server. While no plain text data is ever available on the DBMS, some information, like order, can be inferred from the database schema by an attacker. Some mechanisms, like splitting data across servers (also providing redundancy), may be proposed to address this issue, and further integration with our system would be one of the possible lines of work. This leads to a research direction in the domain of *oblivious storage systems*, to reduce the possible inference correlations between operations over cloud data repositories.

Further extensions to SQL could also be devised. With the development of new searchable encryption schemes, we can expand extSQL operations, in order to provide a common pattern for searches across many types of multimodal data, e.g. work on searches over text also developed in CBIR.

Another line of research would consist in providing a Docked [22] solution of the server, ready to be deployed on the cloud, integrating both the DBMS and the CBIR server along with the dispatcher for attestation, therefore allowing for better portability and easier changes to components.

Furthermore, a possible extension to our system would consider using a DBMS other than MySQL, like MongoDB [23]. Adoption of another system would require changes to the CryptDB's proxy, as MongoDB uses a different communication protocol than that supported. However, this work would allow for future integration with most DBMS available with fewer changes, and further integration with different systems, like Google BigQuery [24].

References

1. *Amazon Web Services*. Provided by Amazon, Available from https://aws.amazon.com/, accessed on 26/09/2016.
2. *Microsoft Azure Platform*. Provided by Microsoft, Available from https://azure.microsoft.com/, accessed on 10/10/2016.
3. *Google Docs*. Provided by Google, Available from https://www.google.com/docs/, accessed on 10/10/2016.
4. Directorate-General for Internal Policies. *Study on Cloud Computing*, European Parliament, 2012.
5. Shvets, Eugene et al. *Indexing Documents in Azure Blob Storage with Azure Search*. Microsoft Research.
6. *AWS GovCloud*. Provided by Amazon Web Services, Available from https://aws.amazon.com/govcloud-us/, accessed on 12/10/2016.
7. Popa, Raluca et al. *CryptDB: Protecting Confidentiality with Encrypted Query Processing*. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles* (SOSP), Cascais, Portugal, October 2011.
8. Arasu, Arvind et al. *Orthogonal Security With Cipherbase*. Microsoft Research, January 2013.
9. Tu, Stephen et al. *Processing Analytical Queries over Encrypted Data*. MIT CSAIL, 2013.
10. Bajaj, Sumeet et al. *TrustedDB: A Trusted Hardware based Database with Privacy and Data Confidentiality*. Stony Brook Computer Science, 2011.
11. *Amazon RDS*. Provided by Amazon Web Services, Available from https://aws.amazon.com/rds/, accessed on 12/10/2016.
12. *MySQL DBaaS Deployment*. Provided by Bitnami and Google Cloud, Available from https://docs.bitnami.com/google/components/mysql/, accessed on 16/10/2016.
13. Trusted Computing Group. *TPM 2.0 Specification*. 2015.
14. Dierks, T. and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246, DOI 10.17487/RFC5246, August 2008. Available from http://www.rfc-editor.org/info/rfc5246, accessed on 11/10/2016.
15. Ferreira, Bernardo. *Privacy-Enhanced Cloud-Storage and Multimodal Data-Searching*, 2016.
16. Librio, Joo. *Privacy-Enhanced Searchable Encryption in Multi-Cloud Storage Repositories*, 2016.
17. Popa, R. et al. *CryptDB Prototype*. Available from http://css.csail.mit.edu/cryptdb/, accessed on 03/03/2016.
18. Stallings, William. *Computer Security: Principles and Practice*, Third Edition, Pearson, 2016.
19. *TPM Quote Tools*, version 1.0.2. Available from https://sourceforge.net/projects/tpmquotetools/, accessed on 17/09/2016.

20. *TrouSerS TSS 1.2*. Available from http://trousers.sourceforge.net/, accessed on 17/09/2016.
21. Shoup, Victor. *NTL: A Library for doing Number Theory*, version 5.4.2. Available from http://www.shoup.net/ntl/, accessed on 24/09/2016.
22. *Docker Platform*, Available from https://www.docker.com/, accessed on 11/10/2016.
23. *MongoDB*, Available from https://www.mongodb.com/, accessed on 12/10/2016.
24. *Google BigQuery*, Available from https://cloud.google.com/bigquery/, accessed on 12/10/2016.