Inout Secure DB: Maximizing security for data INside and OUTside the database

Luiz Gomes-Jr[1], Isabella Mika Taninaka[1], Marcelo Rosa[1], Keiko Fonseca[1], Daniel Lucani[2]

[1] Universidade Tecnológica Federal do Paraná
Curitiba – PR – Brazil
taninaka@alunos.utfpr.edu.br, {lcjunior,mrosa,keiko}@utfpr.edu.br
[2] Chocolate Cloud
daniel@chocolate-cloud.cc

Abstract. As cloud services are becoming an alternative for internal IT infrastructures in many organizations, guarantees of data privacy become a priority. This article presents a secure database system that takes privacy as a design principle. The proposed system offers improved privacy guarantees for data in primary and secondary memory as well as for data that is served to users as a result of SQL queries. Data in working memory is protected using Intel's SGX platform for trusted execution, while data in secondary memory uses network coding for secure storage. SGX provides hardware-based processing privacy offering protection for a wide range of sophisticated attacks. Network coding provides inter and intra-cloud privacy for stored data (by means of storage provided by Chocolate Cloud). For privacy of data served to the outside world, we propose a flexible role-based access control mechanism that anonymizes data at query-time. We have implemented a modular, multi-service architecture that is well suited to the advantages and limitations of the SGX platform. We present the architecture of the system, its components and performance evaluation.

Categories and Subject Descriptors: H.2 [**Database Management**]: Miscellaneous; H.3 [**Information Storage and Retrieval**]: Miscellaneous; I.7 [**Document and Text Processing**]: Miscellaneous

Keywords: Data Privacy, Privacy by Design, Secure DBMS

1. INTRODUCTION

The continuous development and growing availability of cloud services are changing the relationship between organizations and their technological infrastructures. Several commercial cloud providers currently offer products that provide flexible scalability, high availability, and security to small and large companies [Cusumano 2010]. These products have become very attractive given the potential to reduce costs when contrasted with in-house development and support of IT infrastructures.

However, cloud services can introduce security risks to their clients: although transferring private data to data centers can be secured by TLS/SSL protocols, and cloud service providers often offer a high level of security from external attacks, internal deliberate attacks are harder to predict and prevent. Internal personnel (e.g. a network administrator) could have full privileges to access sensitive information. Although encrypting sensitive information would provide some protection, attackers with physical access to the servers can still try to gain access to decrypted information in computer memories. This problem is more evident in databases deployed in cloud service providers: even when data is stored encrypted, during the query processing the data are decrypted in memory and are vulnerable to attackers with physical access to the servers.

To tackle these internal security aspects we propose a secure database architecture based on the

Intel's SGX[1] technology [Costan and Devadas 2016]. SGX, a kind of trusted execution environment (TEE), provides secure enclaves where software code runs in an encrypted region of the memory (thus protecting data from physical attacks). As for data stored on secondary memory (disks), we take advantage of a related project on a secure key-value store, namely Chocolate Cloud Storage Service[2]. The CC Storage Service uses network coding algorithms to spread data across distinct servers, which makes it difficult for attackers to reconstitute the encrypted data even when some servers are compromised.

Securing data internally is critical in a cloud environment but becomes useless when data is served to users and applications in plain form. Therefore, we consider that the design of security mechanisms for external data access is an indispensable part of a secure database system project. To protect sensitive information and guarantee data privacy, it is important to use data anonymization techniques and make them an integral part of the system. Data anonymization changes the data in order to obfuscate details that can be used to identify sensitive information. In this article we present our rule-based access control mechanism that streamlines the management of anonymized access to the database. In our solution, the administrator defines the anonymicity level allowed for each role. This fine-grained access control allows the data to be anonymized when required, serving users with either the raw, non-modified data or with a layer of anonymization – adjusted based on the user's role profile. The technique to anonymize the data is registered in the database as a user defined function (UDF), providing flexibility for administrators to match roles with diverse anonymization options.

This article describes the implementation of our SQL-based database system with improved privacy guarantees for data. The goal is to provide a unifying solution that preserves privacy both internally (secure processing of data) and externally (automatic data anonymization when users query the database). The system's architecture is based on multiple, complementary services. This strategy simplifies the development using the SGX technologies (given current memory allocation limitations) but also makes the system more flexible and scalable.

The remainder of the article is organized as follows: Section 2 presents important concepts and research related to this proposal. Section 3 describes the architecture of the system and the implementation of the internal mechanisms. Section 4 details the experiments developed to test the system's performance. Finally, Section 5 concludes the article with final considerations and future work.

2. BACKGROUND AND RELATED WORK

Regarding the system's internal security, there have been several proposals for databases employing traditional encryption strategies [Shmueli et al. 2009; Basharat et al. 2012]. While these proposals offer a reasonable level of security for data stored in secondary memory, the systems are still vulnerable to physical attacks in which the opponent has direct access to the computer's primary memory. Homomorphic encryption [Gentry 2009], a theoretical solution for this problem, has a performance penalty of orders of magnitude, making it unsuitable for database workloads.

Hardware support for encrypted in-memory processing is an answer for the aforementioned performance issues. Intel's SGX [Costan and Devadas 2016] is currently the most comprehensive and commercially available implementation of a trusted execution environment (TEE). As a TEE, it has the following features: (i) containing a hardware encoded cryptographic key (known only by the module); (ii) remote attestation (which means two enclaves can trust each other through a modified Sigma protocol over a Diffie-Helman Key Exchange - DHKE - allowing them to exchange cryptographic keys), (iii) sealing data (which means once a data is sealed by the module, it can only be unsealed by that module). From the developer's perspective, it ensures that nothing can tamper with

[1]https://software.intel.com/en-us/isa-extensions/intel-sgx
[2]http://www.chocolate-cloud.cc/

the code running inside such modules. It also changes the way software are developed in order to run safely from external attacks.

Intel's SGX has been employed in several high profile projects, such as Microsoft's Haven [A. Baumann and Hunt 2015] and SecureCloud [Brito and Fetzer 2018]. The interest from the industry and research communities contributes to the evaluation and improvement of the security and performance guarantees.

A practical approach to build a secure database is to use simpler models for querying and storage, which reduces the amount of code to be protected. The most basic model of NoSQL database is the key-value, which uses arbitrary keys to identify values stored as uninterpreted byte arrays. This was the approach taken by the Chocolate Cloud project [Sipos et al. 2017]. The service offers secure processing and communication through the use of the SGX architecture and guarantees confidentiality for the storage using network coding algorithms.

Network coding is based on splitting the information of an item into parts stored at different locations. The reconstruction of the original item can only be done by retrieving multiple parts. This strategy guarantees confidentiality even when some of the storage servers are compromised [Sipos et al. 2017].

In our proposal, we are using the Chocolate Cloud Storage Service to provide secure data storage and retrieval. We are then building more expressive modules to provide more flexibility while constraining intra-module complexity and system vulnerability. To provide more flexibility to users, we implement a service that translates SQL queries into Chocolate Cloud's key-value query API.

There have been other proposals to bridge the relational model with simpler NoSQL models. For example, [Schreiner et al. 2015] proposes mappings for key-value stores and document databases. Another approach is the new class of databases labeled as NewSQL [Pavlo and Aslett 2016], which focus on providing BigData capable systems that retain SQL and integrity guarantees. However these proposals do not address the security issues described here.

The other aspect of security addressed in this work regards data that is served to the outside world through queries. The usual strategy in this case is to anonymize the data before delivery. There are several strategies to provide guarantees of anonymity to data, with some of the most used approaches

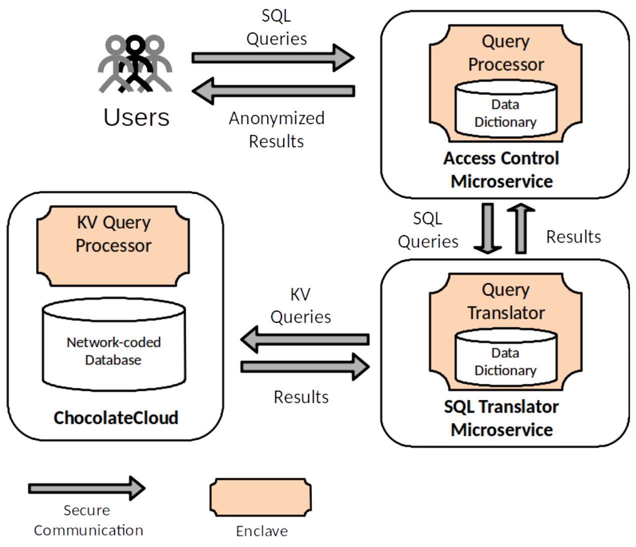

Fig. 1. Inout DB architecture

being based on k-anonymity and entropy [Kelly et al. 2008]. Anonymization can also be applied to unstructured data, a notable example being blurring of people's faces in images and videos based on machine learning algorithms for face detection [Garcia and Delakis 2004]. Given the wide range of anonymization techniques, we defined, as a design principle, to being agnostic in relation to the specific strategy to anonymize the data being delivered to users. In our proposal, the database administrator controls which anonymization strategy is more adequate for a given subschema of data. We provide fine-grained configuration of strategies based on a Role-based Access Control [Ferraiolo et al. 1995] mechanism.

3. SYSTEM DESIGN

The main principles considered when designing the proposed system are:

—Provide the highest level of privacy guarantees for data in all stages of storage and processing

—Simplified integration of data anonymization options for administrators

—Offer a SQL-based query language for users

—Modularization of code base

The following sections describe how these design principles were met and detail our implementation strategies.

3.1 Architecture

We are building our secure database service in a modular architecture. Independent modules are easier to fit in the enclave's limited memory and can be replicated for scalability. Figure 1 shows the components of the architecture. Users issue queries to the *Access Control Microservice*, responsible for role-based access and privacy control. This microservice offers regular access control, matching roles to schema definitions, and adds privacy options, allowing access to anonymized elements of the database. The *SQL Translation Engine* handles SQL queries and interacts with the underlying key-value store module to answer requests. The translation engine receives queries from the Access Control Microservice, parses the contents and makes the necessary calls to the underlying data store to compose the results.

To simplify the deployment of our algorithms in SGX-enabled environments, we used SCONE

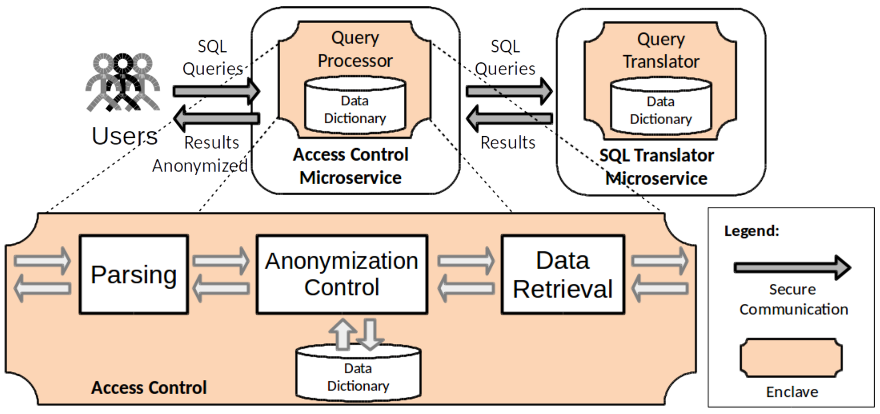

Fig. 2. Access Control microservice - Internal modules

[Arnautov et al. 2016]. SCONE is a secure container mechanism for Docker[3] that uses the SGX trusted execution to protect container processes from outside attacks.

3.2 Attack model

We consider that both the hardware and the operational system running in the cloud machines can be tampered with: this includes side-channel attacks to the processors [Oleksenko et al. 2018] and unauthorized root access by cloud providers [Richter et al. 2016], for example.

All communications with our secure database service - running inside enclaves - are performed over TLS/SSL protocols. All private keys used by such a service are not hardcoded: they are stored and managed by SCONE CAS/LAS services. Such services also run inside enclaves and provide keys to our software after a remote attestation – a process where two different enclaves running in different machines check if they trust each other.

Therefore, only trusted applications outside the cloud can securely interact with our secure database service, and the data processed by such a service can not be accessed by unauthorized users: this feature is granted by Intel SGX technology.

3.3 Automatic Data Anonymization

```
CREATE USER maria; GRANT fullaccess TO maria;
CREATE USER joao; GRANT noaccess TO joao;
```

Fig. 3. An example of user creation and role assignment

```
CREATE TABLE teachers(name, age, department);
CREATE ROLE fullaccess; CREATE ROLE noaccess;
GRANT ALL PRIVILEGES ON teachers TO fullaccess;
GRANT SELECT(anon(name), age, department) ON teachers TO noaccess;
```

Fig. 4. An example of role creation and privilege granting

The Access Control microservice is the interface between users and the underlying data services. Figure 2 shows the internal modules of the service. Database administrators use the service to assign access privileges to users, which are stored in the data dictionary. Users issue SQL queries, which are parsed and checked for compatibility with the user's privileges. Accepted queries are redirected to the SQL Translator microservice, which retrieves the data and compiles the result set. Before returning the results to the user, the Access Control microservice may need to process the data to comply with the anonymity requirements.

In the proposed Role-based Access Control mechanism, the database administrator defines roles to be assigned to users. When a new user is created, s/he must be given a role so s/he can access the information stored in the database. This role might have restricted access to several tables and attributes, enabling or disabling certain actions by the user. Figure 3 shows the SQL-inspired commands for the creation of users and granting each one of them different roles. Since each user has a different role, the queries that they can send to the database are different and the results will also be different.

[3]https://www.docker.com/

```
query : create | grant | select | insert;
create : create_table | create_role | create_user;
grant : grant_role | grant_privilege;
create_role : CREATE ROLE role_name_create;
create_user : CREATE USER user_name_create;
grant_privilege : GRANT privilege_type
          ON TABLE? table_name_grant TO role_name_grant_priv;
grant_role : GRANT role_name_grant_role TO user_name_grant_role;
privilege_type : ALL PRIVILEGES #Priv_All
           | privilege_list #Priv_Others;
privilege_list : table_privilege (COMMA table_privilege)*;
table_privilege : DELETE                  #Priv_Delete
           | INSERT                       nyy#Priv_Insert
           | REFERENCES attribute_list?    #Priv_References
           | SELECT select_item_list?      #Priv_Select
           | UPDATE attribute_list?        #Priv_Update;
attribute_list : [ attr_name_grant (COMMA attr_name_grant)* ];
select_item_list : [ select_item (COMMA select_item)* ];
select_item : user_function cache? [ attr_name_grant_select ]
           | attr_name_grant_select;
```

Fig. 5. Simplified grammar definitions

When creating the database schema (tables/attributes), the administrator can specify several levels of access that can be granted to a role, varying from full access to a table's attributes, anonymized versions of attributes or no access at all. To specify that a given role has restricted access to an attribute, the administrator chooses the anonymization function that the system has to apply at query-time.

Figure 4 shows an example of table and role creation as well as the operations that associate them. In the example, the role *fullaccess* can see and edit the whole table *teachers*, while *noaccess* can only see an anonymized version of the attribute *name* derived from the function *anon()*.

Figure 5 shows a simplified definition of our grammar based on Backus–Naur notation (as used by the ANTLR[4] parse generator). The *grant_privilege* rule is used to assign an access level to a role. The *select_ item* rule encompasses the most important options available for the administrators when granting privileges to a privacy-sensitive table attribute. The administrator defines the attribute name for the grant and chooses (i) the anonymization function to be applied, and (ii) whether the anonymized contents of the attribute must be cached (details below).

The normal behavior of the *select* query, when retrieving content for a user whose role is at the anonymized access level, is to anonymize the content as the query is being processed. However, if the cache option is used when granting access, the first time a *select* query is issued by a user with this role, the anonymization will happen during the processing of the query. The result will also be stored together with the original content. If subsequent queries request the same content with the same anonymization level, the content delivered will be the cached one and the anonymization during the processing will not be needed. This mechanism is especially important for computationally heavy anonymization procedures such as blurring of faces in images or videos (examples are provided in Section 4).

[4]http://www.antlr.org/

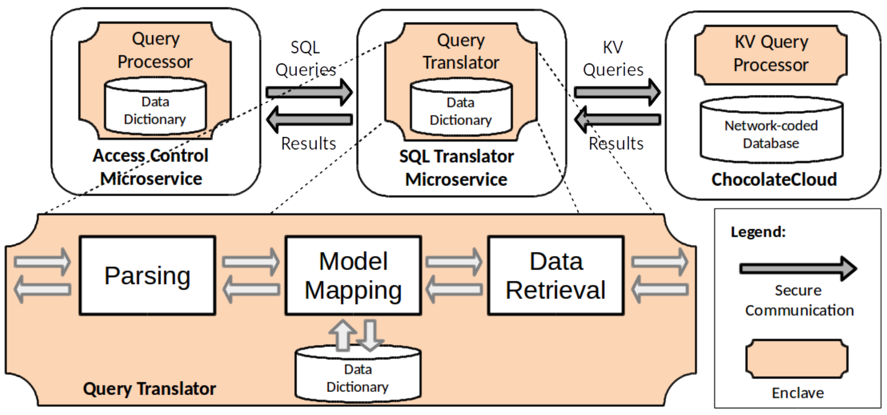

Fig. 6. SQL Translation service - Internal modules

```
a) INSERT INTO reading
       (    clientID, timestamp, read_1 ... read_30);
b) key: reading | <clientId> |  <timestamp>
   value: [<read_1> ... <read_30>]
```

Fig. 7. Example an insert query (a) mapping into a key-value pair (b). ClientID and timestamp compose the primary key of the table and become part of the mapped key. The table has 30 attributes (omitted for clarity).

3.4 SQL Translation

Figure 6 shows the internal parts of the SQL translation module and its interactions with the key-value store. Data definition (DDL) queries are handled by the parser and have their settings stored in the data dictionary. The schema and keys define how each tuple will later be mapped to the key-value model.

When a data manipulation (DML) query is received, a dictionary lookup is performed to determine the mappings between the models. For insertion queries, the specified row has to be transformed into a key and a value to be inserted in the key-value store. In our approach, a key is composed by a concatenation of the table name and the primary key(s) (a predefined separator is used for readability). Tuple attributes are also concatenated to be stored as the value. Figure 7a shows an insert query for a table named *reading* that has the two first attributes as primary keys. Figure 7b shows the resulting key-value pair generated. The key/value pair is then submitted to the Chocolate Cloud Storage Service. For selection queries, the key is assembled with both the table name and the keys presented in the query, potentially with a wildcard character (*) to retrieve all keys matching the prefix. Currently only queries with conditions over the primary keys are supported. More flexible selects and joins are upcoming developments.

The translation module was implemented in ANSI C. The module issues REST requests to a Chocolate Cloud Storage Service. A wider coverage of the SQL language will be implemented with extra indexes for non-key attributes – stored in the key-value database. Joins on the primary keys can be implemented efficiently since the Chocolate Cloud Storage Service returns keys in alphabetical order, enabling the use of a merge-join algorithm. Other joins can be implemented using the aforementioned extra indexes or full table scans when indexes are not available.

```
CREATE USER mika; CREATE ROLE admin;GRANT admin TO mika;
CREATE USER maria; CREATE ROLE nocache; GRANT nocache TO maria;
CREATE USER joao; CREATE ROLE withcache; GRANT withcache TO joao;
```

Fig. 8. Basic configuration of users for all tests

3.5 Network Coded Storage

This service focuses on managing reliable distributed storage and data privacy of critical or confidential data. It also provides support for standard processing and search operations on data as well as a flexible storage capability for various data types. Our main goal is to deliver a storage system that enables users and applications to trade-off data availability (for efficient processing) and data redundancy/reliability (for efficient storage), but without compromising privacy during communications, processing, or storage.

This service creates *datastores* to store *objects* (key-value pairs) in them. The datastores allow us to separate objects into different namespaces, which can rely on the same level of data protection and overall storage policy at the time of storing the data. The system supports configurable settings, including: (a) number of nodes used to store members of the datastore in a distributed fashion, which is related to the reliability provided by the system; (b) number of nodes that can fail before data loss for the members of the datastore, which determines the amount of redundancy and storage cost in the system; and (c) data locality support, which can speed up retrieval of data for processing.

At its core, this service has a number of micro-services to handle incoming/outgoing data, reliability management, storage in the system, and repairs of node or device losses. A critical component for reliability management is the network coding micro-service, which carries out encoding, decoding and recoding operations within the Intel SGX enclave. The use of Random Linear Network Coding (RLNC) [Ho et al. 2006] is key to reducing the cost of storage of the data, compared to standard replication strategies, while also maintaining a reasonable use of network resources during the repair of lost devices in the system. Network coding allows the system to take an original data of size qL bits and stripe it into g fragments of L bits before starting the encoding process. To encode the data and generate $g+r$ coded fragments (r represents the added redundant coded fragments), the micro-service performs a linear combination over a finite field of $GF(2^8)$ using coding coefficients drawn uniformly at random from the elements of the field. This micro-service also carries out a decoding check to guarantee that any g coded fragments will be sufficient to decode the data. The decoding method is implemented using Gaussian elimination essentially solving a system of linear equations considering the coding coefficients of the specific coded fragments collected at the time of decoding. Although encoding and decoding are used for storing and accessing data, i.e., operations triggered by the system's users, our network coded service also supports a repair functionality triggered by the service itself upon detection of node or device losses. This repair is carried out using the ability of network codes to recode data without the need to decode, which translates in a distributed recombination of coded fragments and reduced traffic between racks and nodes in the network when repairing losses [Dimakis et al. 2010]. We refer the reader to [Paramanathan et al. 2013] for a tutorial on network coding. For more performance details on the network coded storage service, we refer the reader to [Lucani et al. 2018].

4. EXPERIMENTS

To test the performance of our proposed microservices, we set up tests based on two real-world scenarios. The first regards video footage from security cameras in public locations. The videos present a clear privacy concern since unauthorized persons should not have access to the raw footage. The second scenario is that of an electric power company that collects energy usage summaries several

```
CREATE TABLE VIDEO(file);
GRANT ALL PRIVILEGES ON VIDEO TO admin;
GRANT SELECT(faceBlur(file)) ON VIDEO TO nocache;
GRANT SELECT(faceBlur(file) WITH CACHE) ON VIDEO TO withcache;
```

Fig. 9. Simplified commands for configuration of access levels for the videos

```
CREATE TABLE fasor(client, current A, current B, ...);
GRANT ALL PRIVILEGES ON fasor TO admin;
GRANT SELECT(anon(name)) ON fasor TO nocache;
GRANT SELECT(anon(name) WITH CACHE) ON fasor TO withcache;
```

Fig. 10. Simplified commands for configuration of roles access for fasor (real table has 32 attributes)

times per day for thousands of customers. The data are aggregated in our database service and consumed by operational and analysis applications such as billing, fraud detection, and quality of service evaluation.

4.1 Data Anonymization Module

For the experiments of the data anonymization module, we set up users and access roles as shown in Figure 8. The main goal of these tests is to assess the cache mechanism for the anonymization procedures. The role *nocache* is intended to represent access without the caching mechanism, while the role *withcache* represents the use of the mechanism.

The commands for table creation and privilege assignment for the video use case are shown in Figure 9. The user *admin* has unrestricted access to the footage while the roles *withcache* and *nochache* can only access the video processes by the anonymization function *faceBlur()*.

Figure 10 shows the configuration of the tests using a table called *fasor* that contains the energy consumption readings for the tests of the second use case. In this case, we use a simple word scrambling function *anon()* to represent a simple anonymization function. Other settings are similar to those of Figure 9.

The results of the experiments with the video anonymization are in Figure 11. For this computationally heavy anonymization, the benefits of the caching mechanism are clear. A query that requires the execution of the anonymization function takes several times longer to complete. The figure also shows that the overhead of the cache mechanism is minimal when compared to a query that has full access to the data.

For the experiments of our second scenario, we populated the table *fasor* and issued a total of 100 selects for each test (considering that the processing time for the anonymization function is minimal). In this experiment we also assessed the impact of running the microservice inside an SGX enclave (using SCONE). The results of the experiments are shown in Figure 12. The chart confirms the benefits of the caching mechanism but, most importantly, gives insight into the performance degradation under SGX. When the data are retrieved without the anonymization (i.e. the user has full access or data have been already cached), query processing under SGX/Scone is slower. However, when the anonymization procedure is called, the performance under SGX/Scone is better for the procedures tested (word scrambling, md5 hashing). This unexpected behavior is probably due to Scone using PyPy[5], a version of the Python language with a just-in-time code compiler.

[5]https://pypy.org/

4.2 SQL Translation Module

Number of queries	Time without SGX		Time with SGX	
	Total	Per Query	Total	Per Query
10	0.19	0.0191	0.35	0.0354
50	0.25	0.0050	0.58	0.0117
100	0.49	0.0049	1.16	0.0116
1000	5.10	0.0051	11.67	0.0117
10000	51.00	0.0051	117.86	0.0118
100000	523.35	0.0052	1178.59	0.0118

Table I. Performance tests for select queries (in milliseconds)

Since the Query Translator microservice is a potential performance bottleneck in our infrastructure, we developed extensive tests based on a real scenario and real data. Here we focus on tests of the translation engine with and without the use of the SGX environment (SCONE). For the tests, we used queries for insertion (Figure 7) and selection of power usage summaries (for the table fasor in our second use case). The goal is to show the impact of using SGX in the execution of the query translation algorithm.

Table I shows performance tests for select queries over our service. We show results for increasing numbers of elements in the queries. The column *Time without SGX* represents the time taken in the translation of the queries, including identifying query type, parsing, retrieving metadata and building the respective key-value queries, without the use of an SGX enclave. The column *Time with SGX* represents the time to process the queries inside an enclave (using Scone). The tests show a performance penalty of around 100% when using SGX. For the dataset tested, the performance degrades linearly with the increase in the number of queries.

Figure 13 shows histograms for Insert query times with and without SGX. Although queries running on SGX in general take longer to complete, the fastest SGX queries are comparable with the slowest non-SGX queries. Figure 14 shows results for total Select query processing times. The graph shows the time taken for processing queries with 10, 100, 1000, and 10,000 elements. The lines, drawn for

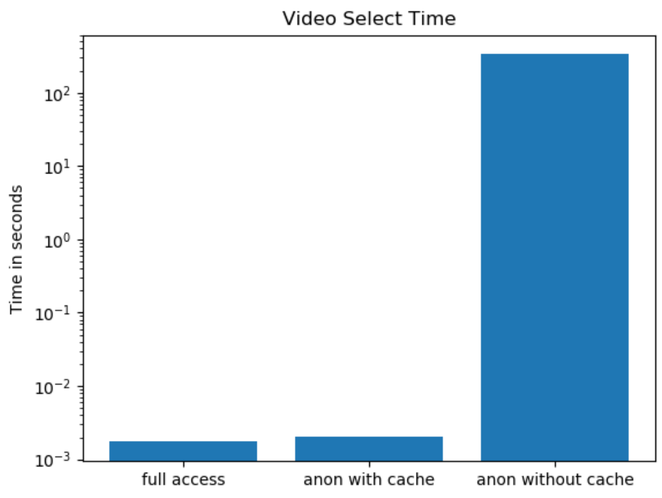

Fig. 11. Evaluation of performance (total time) for `SELECT` statement using cache for video select

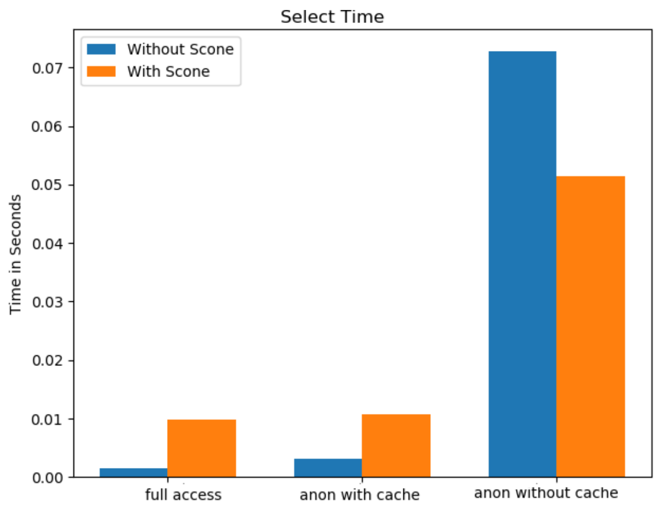

Fig. 12. Evaluation of performance (total time) for `SELECT` statements

reference, show how queries on SGX take longer to complete, reaching more than twice the running time for the larger queries. These tests also demonstrate that the impact of SGX on performance is linear.

5. CONCLUSION

This article described the development of a secure database system with improved privacy guarantees. Our goal is to combine solutions for internal and external data security in a unified framework. Our main contributions are: (i) a modularized architecture intended to simplify SGX integration and improve scalability; (ii) SGX-based implementations of the Access Control and SQL translation modules; (iii) an integrated role-based access control mechanism that streamlines data anonymization in SQL queries.

To assess the performance of our system, we developed experiments based on two real world sce-

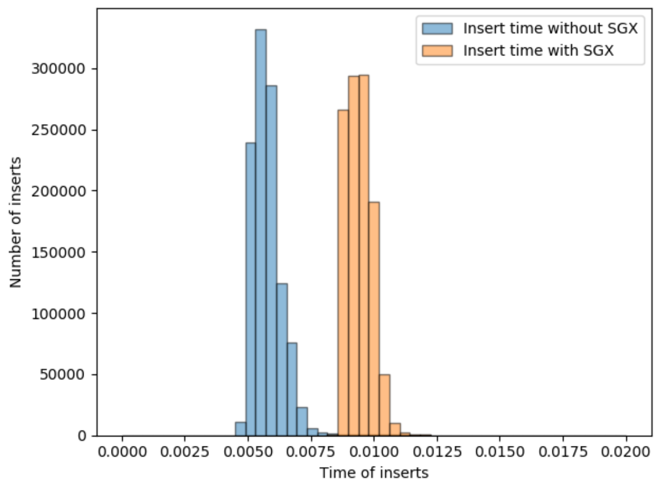

Fig. 13. Evaluation of performance for `INSERT` statements

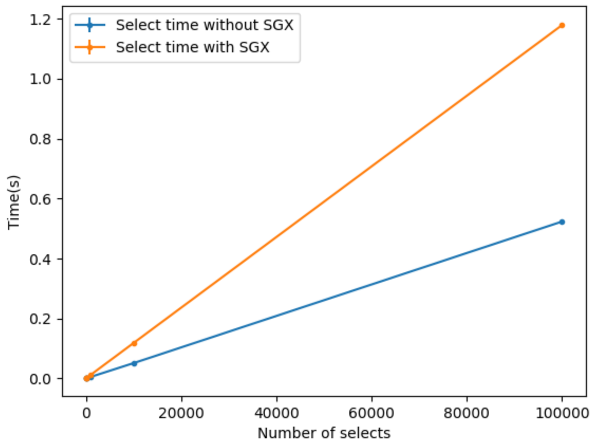

Fig. 14. Evaluation of performance for `SELECT` statements

narios. Our experiments show the performance impact associated with data anonymization and SGX-based execution. While the impact is considerable, we believe that it is tolerable for scenarios with sensitive data. We also demonstrated how our transparent caching mechanism can help curb performance degradation for heavy anonymization functions.

Many of the security features implemented in this research rely on SGX, a proprietary technology from Intel. As such, the proposed system inherits SGX's offered attack surface, present and future, which might pose security threats if the technology is compromised. With the increasing demand for privacy solutions and interest from the industry and research communities, we believe that any design issues will be addressed with urgency, especially considering that currently there are no other technologies that provide such level of privacy with reasonable performance penalties.

Another possible shortcoming is the dependency of a single vendor (Intel). While this might be a limiting factor in the future, it is important to notice that other processor manufacturers are implementing similar concepts in their architectures. ARM has explored TEEs for several years, releasing in 2002 the specification of TrustZone [ARM 2009] for the ARMv6Z sub-architecture. AMD is also developing its AMD Memory Encryption [Kaplan and Powell 2013]. As the technologies mature, we expect industry standards to enable our solution to be implemented across vendors.

Future work in the system include supporting more elaborate SQL queries, query processing optimization, and more performance and security tests. We also plan to assess parallel execution of our microservices to improve scalability.

ACKNOWLEDGMENTS

This research is being performed in the context of the SecureCloud project. The SecureCloud project has received funding from the European Union's Horizon 2020 research and innovation programme and was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under grant agreement number 690111. This work was partially funded by the EU-BR SecureCloud project (MCTI/RNP 3rd Coordinated Call). This work has been supported in part by the INCOM project (No. 8057-00059B) financed by the Innovation Fund Denmark.

REFERENCES

A. Baumann, M. P. and Hunt, G. Shielding applications from an untrusted cloud with haven. *ACM Trans. Comput. Syst.*, 2015.

ARM. Security technology - building a secure system using trustzone technology. Tech. rep., ARM Technical White Paper, 2009.

Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M., Goltzsche, D., Eyers, D. M., Kapitza, R., Pietzuch, P. R., and Fetzer, C. Scone: Secure linux containers with intel sgx. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, K. Keeton and T. Roscoe (Eds.). USENIX Association, pp. 689–703, 2016.

Basharat, I., Azam, F., and Muzaffar, A. W. Database security and encryption: A survey study. *International Journal of Computer Applications* 47 (12), 2012.

Brito, A. and Fetzer, C. Securecloud: Secure big data processing in untrusted clouds. In *DSN Workshops*. IEEE Computer Society, pp. 53–54, 2018.

Costan, V. and Devadas, S. Intel sgx explained. Tech. Rep. 2016/086, Cryptology ePrint Archive, 2016.

Cusumano, M. Cloud computing and saas as new computing platforms. *Commun. ACM* 53 (4): 27–29, Apr., 2010.

Dimakis, A. G., Godfrey, P. B., Wu, Y., Wainwright, M. J., and Ramchandran, K. Network coding for distributed storage systems. *IEEE Transactions on Information Theory* 56 (9): 4539–4551, Sept, 2010.

Ferraiolo, D., Cugini, J., and Kuhn, R. Role based access control (RBAC): Features and motivations. In *Annual Computer Security Applications Conference*. IEEE Computer Society Press, 1995.

Garcia, C. and Delakis, M. Convolutional face finder: a neural architecture for fast and robust face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26 (11): 1408–1423, Nov, 2004.

Gentry, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*. STOC '09. ACM, New York, NY, USA, pp. 169–178, 2009.

Ho, T., Medard, M., Koetter, R., Karger, D. R., Effros, M., Shi, J., and Leong, B. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory* 52 (10): 4413–4430, Oct, 2006.

Kaplan, D. and Powell, J. AMD memory encryption. Tech. rep., AMD, 2013. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf.

Kelly, D. J., Raines, R. A., Grimaila, M. R., Baldwin, R. O., and Mullins, B. E. A survey of state-of-the-art in anonymity metrics. In *Proceedings of the 1st ACM workshop on Network Data Anonymization, NDA 2008, Alexandria, VA, USA, October 31, 2008*, S. Antonatos, M. Bezzi, E. Boschi, B. Trammell, and W. Yurcik (Eds.). ACM, pp. 31–40, 2008.

Lucani, D. E., Feher, M., Fonseca, K., Rosa, M., and Despotov, B. Secure and scalable key value storage for managing big data in smart cities using intel sgx. In *2018 IEEE International Conference on Smart Cloud (SmartCloud)*. pp. 70–76, 2018.

Oleksenko, O., Trach, B., Krahn, R., Silberstein, M., and Fetzer, C. Varys: Protecting SGX enclaves from practical side-channel attacks. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, pp. 227–240, 2018.

Paramanathan, A., Pedersen, M. V., Lucani, D. E., Fitzek, F. H. P., and Katz, M. Lean and mean: network coding for commercial devices. *IEEE Wireless Communications* 20 (5): 54–61, October, 2013.

Pavlo, A. and Aslett, M. What's really new with newsql? *SIGMOD Record* 45 (2): 45–55, 2016.

Richter, L., Götzfried, J., and Müller, T. Isolating operating system components with intel sgx. In *SysTEX' 16*. Trento, Italy, 2016.

Schreiner, G. A., Duarte, D., and dos Santos Mello, R. Sqltokeynosql: a layer for relational to key-based nosql database mapping. In *iiWAS*, G. Anderst-Kotsis and M. Indrawan-Santiago (Eds.). ACM, pp. 74:1–74:9, 2015.

Shmueli, E., Vaisenberg, R., Elovici, Y., and Glezer, C. Database encryption: an overview of contemporary challenges and design considerations. *SIGMOD Record (ACM Special Interest Group on Management of Data)* 38 (3): 29–34, Sept., 2009.

Sipos, M., Braun, P. J., Lucani, D. E., Fitzek, F. H. P., and Charaf, H. On the effectiveness of recoding-based repair in network coded distributed storage. *Periodica Polytechnica.Electrical Engineering and Computer Science* 61 (1): 12–21, 2017. Copyright - Copyright Periodica Polytechnica, Budapest University of Technology and Economics 2017; Last updated - 2017-03-09.