Developing a secure SQL/key-value translation service

Davi Boberg
Universidade Tecnológica Federal do Paraná
Curitiba, Brazil
daviboberg@alunos.utfpr.edu.br

Luiz Gomes-Jr
Universidade Tecnológica Federal do Paraná
gomesjr@dainf.ct.utfpr.edu.br

Marcelo Rosa
Universidade Tecnológica Federal do Paraná
Curitiba, Brazil
mrosa@utfpr.edu.br

Keiko Fonseca
Universidade Tecnológica Federal do Paraná
Curitiba, Brazil
keiko@utfpr.edu.br

RESUMO

As cloud services are becoming an alternative for IT infrastructures in many organizations, data privacy guarantees become a priority. This paper presents the development of a secure database system using Intel's SGX trusted platform. SGX provides hardware-based processing privacy offering protection for a wide range of sophisticated attacks. We propose a modular, multi-service architecture that is well suited to the advantages and limitations of the SGX platform. This paper focuses on the ongoing implementation of our SQL translation service.

Palvras-chave

Secure Databases; Cloud; SGX

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 [3]. These products have become very attractive given the potential to reduce costs when contrasted with in-house development and support of IT infrastructures.

Cloud services, however, can introduce security risks for their clients: private data need to be transfered to the cloud and become vulnerable to internal and external attacks in the adopted platform. Cloud service providers often offer a high level of security from external attacks, but internal deliberate attacks are harder to predict and prevent. An internal personnel (e.g. a network administrator) could have full privileges to access the sensitive information. Encrypting sensitive information would provide some protection, but attackers with physical access to the servers could still perform a physical attack to access decrypted information in the memory

To address these shortcomings, we propose a secure database architecture based on the Intel's SGX[1] technology [2]. SGX, a kind of trusted platform module, provides secure enclaves where code runs in a protected and encrypted region of the computer memory, even protected from attacks related to physical access to that memory area. Unfortunately, the current SGX standard has memory size limitations that makes it impossible to implement a full-blown cloud and BigData-ready database system inside a single enclave. Our strategy is therefore to modularize the architecture and provide different levels of security and query capabilities. Here we describe our architecture for a secure SQL-based cloud database based on a secure key-value store, namely ChocolateCloud [2] (Section 2). We also describe ongoing efforts for the implementation of the system as well as preliminary experiments (Section 3).

1.1 Related Work

There has been several proposals for secure databases employing traditional encryption strategies [7, 1]. While these proposals offer a reasonable level of security for data stored in secondary memory, the systems are still vulnerable to physical attacks where the opponent has direct access to the computer's primary memory. Homomorphic encryption [4], a theoretical solution for this problem, has a performance penalty of orders of magnitude, making it unsuitable for cloud and BigData workloads.

Hardware support for encrypted in-memory processing is an answer for the aforementioned performance issues. Intel's SGX [2] is currently the most comprehensive and commercially available implementation of a trusted platform module (TPM). As a TPM, it has abilities of (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, or DHKE, allowing them to exchange cryptographic keys), (iii) sealing data (which me-

WPCCG '17 Outubro, 2017, Ponta Grossa, Paraná, Brasil

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

ans once a data is sealed by the module, it can only unsealed by that module). To the developer's perspective, it ensures that nothing can tamper with the code running inside such modules. It also changes the way software are developed in order to run safely from external attacks.

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 [8]. The service offers secure processing and communication through the use of the SGX architecture and guarantees confidentially 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.

In our proposal we are using the Chocolate Cloud service to provide secure data storage and retrieval. We are then building more expressive modules to provide more flexibility while constraining intramodule complexity and system vulnerability.

There has been other proposals to bridge the relational model with simpler NoSQL models. For example, [6] proposes mappings for key-value stores and document databases. Another approach is the new class of databases labeled as NewSQL [5], which focus on providing BigData capable systems that retain SQL and ACID capabilities. These proposals however do not address the security issues described here.

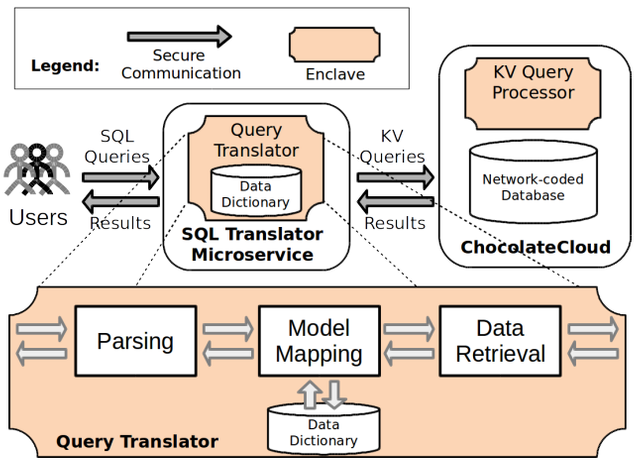

Figure 1: Service architecture

2. ARCHITECTURE AND SQL TRANSLATION

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. The focus of this paper is the *SQL translation engine*, which handles SQL queries and interacts with the underlying key-value store module to answer requests. We are also developing a data anonymization module for access control and on-the-fly data anonymization (not covered in this paper).

```
a) INSERT INTO reading
      (   <clientID>, <timestamp>, <read 1> ... <read 30>);
b) key: reading|<clientId>|<timestamp>
   value: [<read 1> ... <read 30>]
```

Figure 2: 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.

Figure 1 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, 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 2a shows an insert query for a table named *reading* that has the two first attributes as primary keys. Figure 2b shows the resulting key-value pair generated. The key/value pair is then submitted to the Chocolate Cloud store. For selection queries the key is assembled with 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.

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 store 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.

Table 1: Performance tests for select and insert queries (in seconds)

Test	Processing time	Total time	Total time/query
10 `Selects`	0.000171	3.0767	0.308
100 `Selects`	0.000189	28.5287	0.285
1000 `Selects`	0.000172	286.5795	0.286
10 `Inserts`	0.001626	4.1538	0.415
100 `Inserts`	0.013217	41.3288	0.413
1000 `Inserts`	0.13250	418.5262	0.418

3. EXPERIMENTS

We are developing and testing our solution based on a real scenario and real data. The scenario is that of an electric

power company that collects usage summaries several 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.

In the first prototype of our translation module, the code was implemented in ANSI C. The module issues REST quests to a ChocolateCloud database hosted in a cloud service in Europe. Here we focus on initial tests of the translation engine without the use of the SGX environment.

For these preliminary tests we used simple queries for insertion (Figure 2) and selection of usage summaries. The goal is to show that (i) our translation module has a small impact in the overall query execution and (ii) that performance scales linearly with the number of queries.

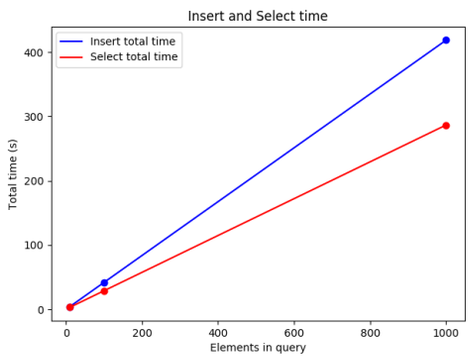

Figure 3: Evaluation of performance (total time) for `INSERT` and `SELECT` statements

Table 1 shows performance tests for select and insert queries over our service. We show results for increasing numbers of elements in the queries (10, 100, 1000). The column *processing time* represents the time taken in the translation of the queries, including identifying query type, parsing, retrieving metadata and building the respective key-value queries. The column *total time* represents the time to process the query, retrieve and assemble the results. This time is much higher because currently the key-value database is located in a cloud server in Europe. In the future we plan to have all modules running on the same cloud infrastructure and expect significant reductions in query times. Finally, the last column shows the total time divided by the number of queries. All values are averages over 10 runs of the experiments. The numbers show that the translation has little impact on query performance. These results were expected since our implementation is in C and the algorithms for processing simple queries are straightforward. The results will, however, serve as the baseline for the next iterations of our implementation, especially when we embed the processing in the SGX enclave and enable more complex queries.

Figure 3 shows results for total query processing times. The graph shows the time taken for processing queries with 10, 100 and 1000 elements. The lines, drawn for reference, shows the linear trend for scalability. Figure 4 shows results for processing time (excluding time to query the key-value store). Again, time for inserts grows linearly. Processing time for selects remain constant because there is only one query to be processed (only the size of the results change).

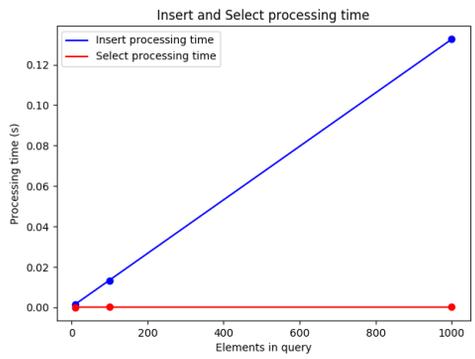

Figure 4: Evaluation of performance (processing time) for `INSERT` and `SELECT` statements

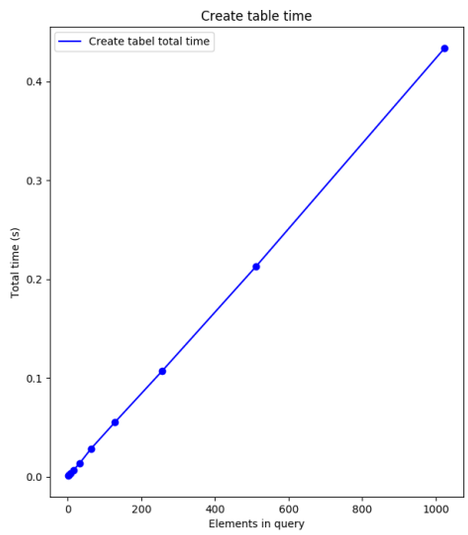

Figure 5: Performance of `CREATE TABLE` statements

Finally, Figure 5 shows processing times for create table queries (which currently only affect the internal data dictionary). Again, these results were expected given the nature of the algorithms and will serve as baseline for future tests.

4. CONCLUSION

This paper described our ongoing work towards the development of a secure service to enable SQL queries to a key-value store. The service is being integrated in Intel's SGX platform, which provides hardware-based privacy guarantees. Our service translates SQL queries into key-value calls to a Chocolate Cloud storage service. Chocolate cloud also offers increased levels of security employing both SGX and network coding algorithms.

We are currently working on the integration of our translation service with the SGX library developed by our research group. The next steps include supporting more elaborate SQL queries, query processing optimization, and per-

formance tests. We are also working on another service to complement the SQL translation that will be responsible for role-based anonymization and access control.

5. 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).

6. REFERÊNCIAS

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

[2] V. Costan and S. Devadas. Intel sgx explained. Technical Report 2016/086, Cryptology ePrint Archive, 2016.

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

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

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

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

[7] E. Shmueli, R. Vaisenberg, Y. Elovici, and C. Glezer. 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.

[8] M. Sipos, P. J. Braun, D. E. Lucani, F. H. P. Fitzek, and H. Charaf. 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.