An Experimental Performance Analysis of the Cryptographic Database ZeroDB

Michael Mitterer
Technical University of Munich
mitterem@in.tum.de

Heiko Niedermayer
Technical University of Munich
niedermayer@net.in.tum.de

Marcel von Maltitz
Technical University of Munich
maltitz@net.in.tum.de

Georg Carle
Technical University of Munich
carle@net.in.tum.de

ABSTRACT

Cryptographic databases aim to protect the user's data from cloud servers that hold the data and operate the server-side of the database. They are privacy-by-design solutions. ZeroDB belongs to the family of cryptographic databases that solves this by off-loading the main database functionality to the client and many rounds of communication. We study ZeroDB performance by crafting specific queries and observing the operation. Furthermore, we compare ZeroDB to MySQL, SQLite, and MongoDB in TPC-C measurements. All measurements were performed in a testbed where network parameters were edited. The results show that the communication overhead of ZeroDB leads to significant drops in performance compared to the non-cryptographic databases, in particular for write and update operations. With respect to the trade-off of security and performance, ZeroDB is only recommendable when security requirements outweigh the performance impact. The reasoning shows that performance can be a limiting factor for the usability of a privacy-by-design solution.

KEYWORDS

Cryptographic Database, ZeroDB, Performance

ACM Reference Format:
Michael Mitterer, Heiko Niedermayer, Marcel von Maltitz, and Georg Carle. 2018. An Experimental Performance Analysis of the Cryptographic Database ZeroDB. In *W-P2DS'18: 1st Workshop on Privacy by Design in Distributed Systems , April 23–26, 2018, Porto, Portugal,* Michael Mitterer, Heiko Niedermayer, Marcel von Maltitz, and Georg Carle (Eds.). ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3195258.3195264

1 INTRODUCTION

With the advent of apps and the mobile revolution the ratio of user data stored in the cloud instead of just the user's devices is increasing. This enables many new applications. However, the cloud knows the user's data. This can be used for profiling and information could be sold to 3rd parties with whatever intents. Even if the company is not in any way acting maliciously, an attack may leak information to outsiders and all promises of data protection from the cloud provider become invalidated by this event.

Similar developments are seen in the world of businesses. In order to focus on the core competence of a company, IT operations are outsourced to specialized companies. This can still happen on-site, but the use of Cloud Computing is on the rise for businesses as well. It helps businesses to reduce their costs and become more competitive [7]. While this sounds very good at first, one may wonder about the consequences for company secrets. Privacy of their users as well as data protection regulation may also raise concerns.

If we remember the NSA scandal which was made public by Edward Snowden, outsourcing IT and data to international companies may not be desirable if it should not be leaked. Companies have to be very careful. Privacy-by-design solutions are needed, yet it is unclear if they can provide the required functionality with good-enough user experience.

An interesting solution would be to have these cloud benefits, but not leak potentially sensitive data to cloud servers. Disk encryption may seem promising, but it is not sufficient [9]. Encrypting all data directly solves the privacy problem. The cloud cannot read the plaintext. Thus, nothing is leaked. However, the cloud cannot process the data anymore, which is an important part of its functionality.

Cryptographic databases come in various forms. Some, such as of ZeroDB, require the client-side to run all operations that need decrypted data. Others use ciphers or cryptographic schemes that allow some more processing on server-side. The result is that neither the server nor a hacker who infiltrated the server will now be able to see the plaintext of the data. Thus, a certain level of privacy is achieved and the privacy issue is resolved. Performance and scalability could be a problem, however.

Our contribution is as follows. We set up a general testbed for performance measurements. By adding delay, bandwidth limitations, and packet loss to the network of the testbed, we studied the influence of these parameters on the performance of the analyzed databases, in particular ZeroDB. We updated pytpcc in order to run TPC-C benchmark tests in the environment as well.

In the following we will briefly introduce ZeroDB in Section 2, then discuss related work in Section 3. Our experimental setup and experiment results are shown in Section 4. Finally, we conclude in Section 5.

W-P2DS'18, April 23–26, 2018, Porto, Portugal
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5654-1/18/04…$15.00
https://doi.org/10.1145/3195258.3195264

2 ZERODB

ZeroDB [2] encrypts data and indices on client-side. It uses standard non-deterministic symmetric encryption for this task. The server will, thus, only see strongly-encrypted ciphertext and queries for encrypted nodes. This means that the client has to do the work, since the server cannot decrypt or otherwise process the data. A lot of data has to be transferred over the network to the client. The server stores the data in data blocks, and it stores a B-Tree for each index.

To process a query, the client has to traverse the B-Tree and request the relevant encrypted nodes of the tree in order to find the data items of interest. Each of these steps involves the client requesting a node, the server providing it, the client decrypting the node, and proceeding according to the information found. This process is called *remote traversal* and visualized in Figure 1.

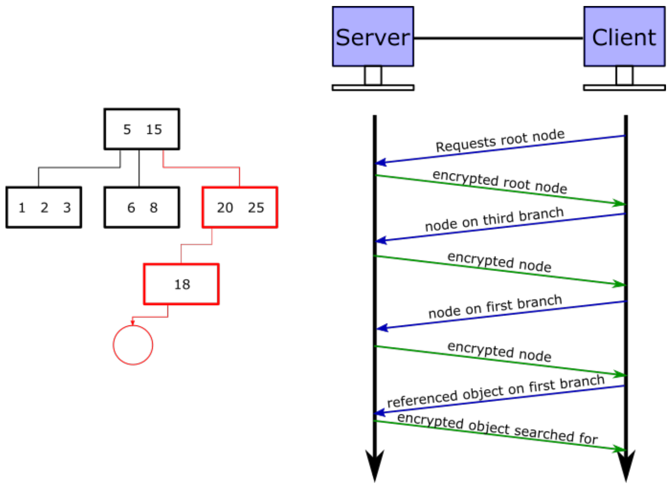

Figure 1: ZeroDB operation

Figure 1 shows the request for all entries where a specific attribute equals 18. It will start at the root and proceed along the red nodes of the tree. The leaf nodes of the B-Tree have reference to the data objects, which are also encrypted client-side.

3 RELATED WORK

First of all, there are cryptographic database proposals other than ZeroDB. CryptoDB[6] and Arx[4] are examples. Important for performance-security trade-offs is the concept to use weaker encryption schemes like Order-Preserving Encryption [3]. These allow the server to do some of the processing, e.g. range queries.

Closer to our work is a large number of work on performance analysis of databases with benchmarks. Most of the authors test the influence of parameters on server-side like the configuration or the hardware setup to different database systems. Popa et al. [5] analyzed in their paper the performance of the encrypted database CryptDB as well as different encryption schemes. [8] determined the HTTP throughput of CryptDB, which was only 14.5 % lower than MySQL. CryptDB is a database using weaker encryption schemes. Thus, it performs better than ZeroDB, but the security level is not as high as with the database system evaluated in this paper. Because of this disadvantage we tested only ZeroDB in this paper.

Other benchmarking works like [1], Brendel looked at the impact of server-side factors like I/O and RAID on the database performance. In contrast, we consider them as given and modify network parameters.

Furthermore, Yao et al. [11] compares the performance of MySQL, SQLite and Redis, a key-value store [10]. Depending on the query mix, each system had its benefits.

4 EXPERIMENTS

4.1 Experiment Setup

The experiments are performed in a setup where two computers form a client-server infrastructure. The machines in the testbed had the following configuration. They had Intel Xeon CPU E31230 processors at 3.20 GHz with 16 GB memory, 1 TB RAID 1 storage and an Intel 10 Gbit Ethernet Network Adapter X540-T2 network card.

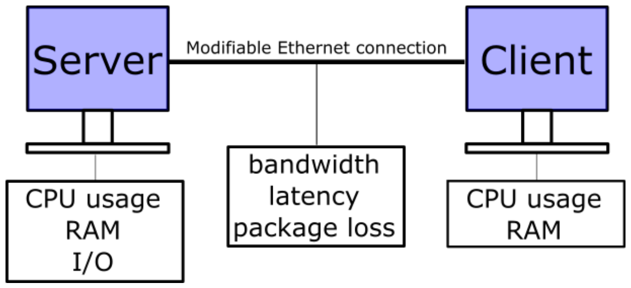

Figure 2: ZeroDB operation

Figure 2 shows the details of our setup. PC1 acts as server holding the database and managing the experiments, PC2 behaves like a client who is sending requests to the server to query records from the database and store new entries. Both machines are connected directly by a copper RJ45 cable providing a bandwidth of 10 Gigabit per second. The operating system on both machines was Debian Jessie. All necessary software and libraries were pre-installed on the machines.

In the subsequent subsections, we will first look at experiments with ZeroDB. Then, we will compare ZeroDB with other database systems using a Python implementation of the TPC-C benchmark.

4.2 ZeroDB Experiments

In this section, we present experiments that try to understand the behavior of ZeroDB given particular types of queries. Table 1 provides an overview of the setup. To evaluate the influence of the network parameters we simulate a Ethernet link with low latency and a bandwidth of 1000 Mbps as basic configuration . Initially, the database is filled with 50000 records representing measurements in a smart building with the following attributes.

(1) roomID: any random integer between 1111 and 9999
(2) nodeID: any of 900 random integer between 11111 and 99999
(3) value: any random integer between 1 and 9999

(4) date: a timestamp calculated from actual time and a running number (not randomly generated)
(5) desc: the string "door" concatenated with a randomly generated string of characters of the size which is given in the recordsize
(6) state: any random integer between 0 and 3 inclusive, where each number represents a state of the node

We mainly change the query type. Each time 1000 records are affected with one query.

Latency	10 milliseconds
Bandwidth	1000 Mbps
Database Size	50000 records
Query Size	1000 records
Data Distribution	uniform
Record Size	10 Byte
Package Loss	0%

Table 1: Default values of experiment parameters

After each experiment, the server and the client were restarted.

ZeroDB relies heavily on client-server communication. So, the first aspect we are interested in is the number of roundtrips between client and server. For comparison, we also include the values for MySQL given the same data and query. The results of this experiment (Figure 3) show that in line with our description, ZeroDB client and server communicate a lot in both directions. Insert and Delete are expensive in ZeroDB and Update is extremely expensive. The number of roundtrips for the select statement is moderate, considering that 1,000 data records are to be found and returned.

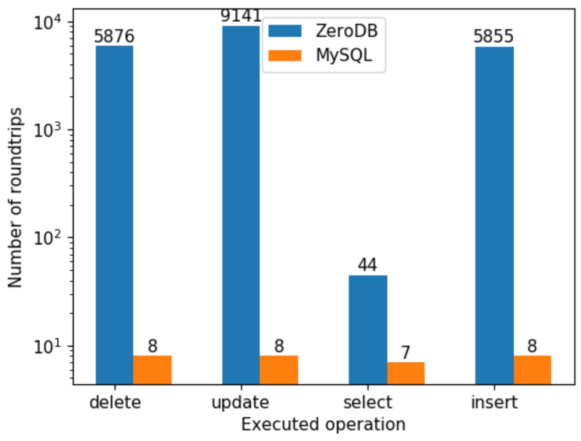

Figure 3: ZeroDB roundtrips per query type

Figure 4 shows a similar behavior of ZeroDB for the overall execution time of the queries. Execution time is, of course, more important than the number of roundtrips. The pattern in the results remains the same. The Update operation with ZeroDB takes 209 seconds, while the same operation with MySQL needs 0.1 seconds. The Update is the slowest operation of ZeroDB. Insert took 158 seconds, Delete 78 seconds, and the Select 0.5 seconds. If only the performance of select statements matter in a given scenario, ZeroDB might provide a good-enough performance as its performance is at least almost in the same order of magnitude as MySQL's performance.

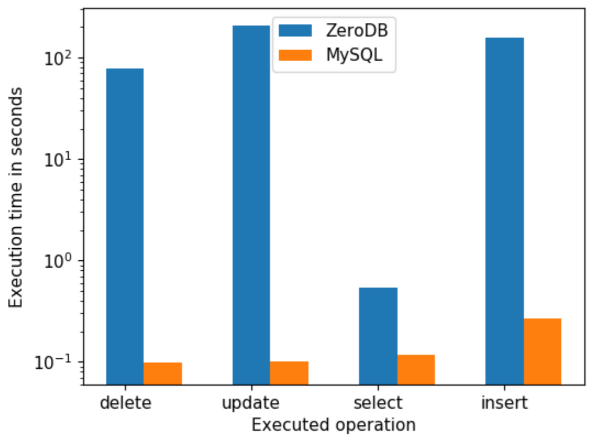

Figure 4: ZeroDB execution time per query type

Figure 5 shows how ZeroDB reacted to network latency in our experiments. While the increase seems to be linear, only the select query remains within a reasonable execution time when the latency is increased. Operations like Insert and Delete took too long for reasonable execution time over even a modest delay that could occur within a country.

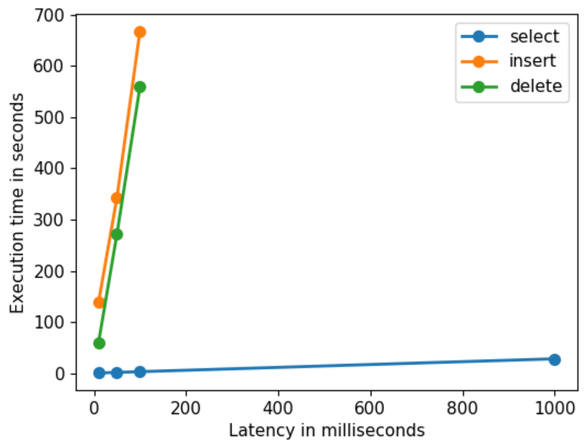

Figure 5: ZeroDB latency

Finally, in Figure 6 we show how ZeroDB reacts to packet loss l in the network. While the impact of the 50 % loss is huge, it is important to note that even the lower loss rates of 5 % or 10 % increase execution time out of proportion. It can be concluded that packet loss badly affects ZeroDB.

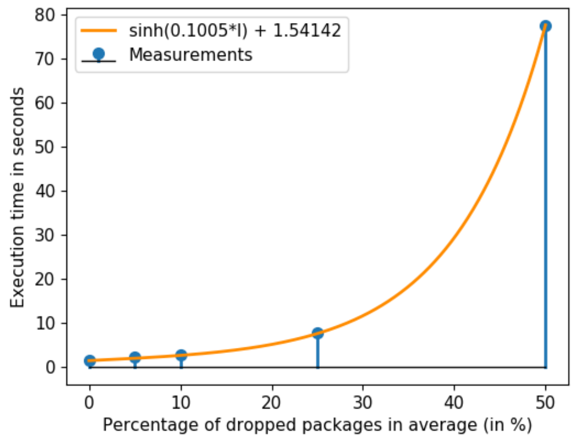

Figure 6: ZeroDB operation and packet loss

4.3 TPC-C Benchmark

We used pytpcc to run TPC-C benchmarks on ZeroDB and other databases. We had to write drivers for ZeroDB and MySQL in order to include them in pytpcc. SQLite is run over an encrypted network file system where the data resides on the server. We then let the benchmark run in our testbed with varying network conditions.

Figure 7 shows the average execution time of the TPC-C delivery query mix for ZeroDB, MySQL, SQLite, and MongoDB. ZeroDB is orders of magnitude slower than the other database systems. It is also heavily influenced by the network latency between client and server. One of the reasons for this slow performance could be that ZeroDB is slow for inserts and studies with mainly select queries could be a bit more favorable. For the other database systems there is only a small increase with latency as data transfer in the beginning and end of queries takes a bit more time. The query itself is not influenced by the network latency in these systems.

Table 2 shows the results when we restrict the available bandwidth down to the order of a DSL connection. In this operational state the bandwidth has little impact on MySQL and MongoDB. The switch from 10Mbps to 1 Mbps showed a 30-40 % increase in execution time for SQLite running over the network file system as well as for ZeroDB.

Our experiments with ZeroDB showed that it consumes far more storage than expected. Table 3 shows the allocated storage space of the TPC-C database after an initial population run with pytpcc. The value for ZeroDB is measured after running the garbage collection which removes all outdated indices and data from the database file. Without the call of the garbage collection the value would be significantly larger. Even after this, it is four times the size of the database size on disc of MongoDB, the worst of the other database systems with respect to size on disk. According to the ZeroDB white paper [2], this can be led back to the indices created by ZeroDB. The encrypted database stores an index for each attribute by default while conventional database systems only calculate indices for the attributes specified by the TPC-C standard.

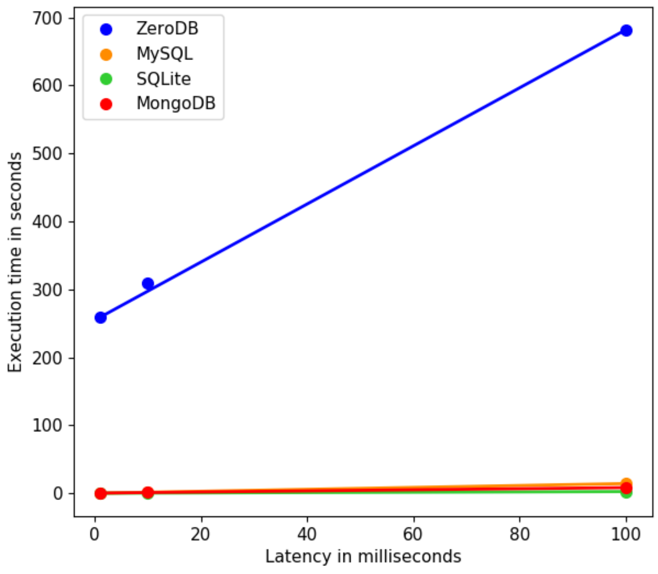

Figure 7: Pytpcc benchmark performance vs latency

	1Mbps	10Mbps	1000Mbps
ZeroDB	38.9	29.4	28.7
MySQL	0.96	0.98	0.97
SQLite	0.70	0.51	0.49
MongoDB	0.770	0.761	0.761

Table 2: The influence of bandwidth to the execution time (in seconds) of the query mix "New Order" to the given databases

	ZeroDB	MySQL	SQLite	MongoDB
pytpcc	2,4 GB	526 MB	372 MB	645 MB

Table 3: Database size on disc after the initial database population of pytpcc

5 CONCLUSIONS

ZeroDB provides confidentiality for databases where the data is stored on cloud servers. It follows an honest but curious model where the server will provide the data as needed, yet it will not learn about the details of the data. Compared to traditional databases or modern NoSQL databases, ZeroDB is remarkable slower. The large amount of communication makes ZeroDB performance decline significantly with bad network conditions. Databases with less communication overhead are less affected by this. ZeroDB performs better for select operations. However, the performance penalty of write and update operations is huge.

There are solutions like Order-Preserving Encryption where the server-side can provide more functionality and queries can be retrieved faster. In this security-performance trade-off ZeroDB is on the slow yet considerably secure side.

ACKNOWLEDGMENTS

This work was supported by the European Commission through the project SafeCloud (H2020-653884).

REFERENCES

[1] Jens-Christoph Brendel. 2008. MySQL-Benchmark hilft Konfiguration optimieren. *Linux-Magazin* 2008, 12 (2008), 36–38.

[2] Michael Egorov and MacLane Wilkison. 2016. ZeroDB white paper. *CoRR* abs/1602.07168 (2016). http://arxiv.org/abs/1602.07168

[3] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over Encrypted Data in the Database-Service-Provider Model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 216–227.

[4] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2017. Arx: A DBMS with Semantically Secure Encryption. (2017).

[5] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 85–100.

[6] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: A Practical Encrypted Rrelational DBMS. (2011).

[7] Gwynne Richards. 2014. *Warehouse Management: A complete guide to improving efficiency and minimizing costs in the modern warehouse*. Kogan Page Publishers.

[8] Stoyan Stefanov. 2005. *Building Online Communities with phpBB 2*. Packt Publishing Ltd.

[9] Dan Suciu. 2012. SQL on an Encrypted Database: Technical Perspective. *Commun. ACM* 55, 9 (2012), 102–102.

[10] Lemmy Marco Tauer. [n. d.]. Key Value Datenspeicher am Beispiel Redis. ([n. d.]).

[11] Xing Yao, Wei Su, and Shuai Gao. 2017. Performance Analysis of Different Database in New Internet Mapping System. In *AIP Conference Proceedings*, Vol. 1820. AIP Publishing, 090017.