INSTITUTE FOR DEFENSE ANALYSES

ERP Homomorphic Encryption Performance Evaluation

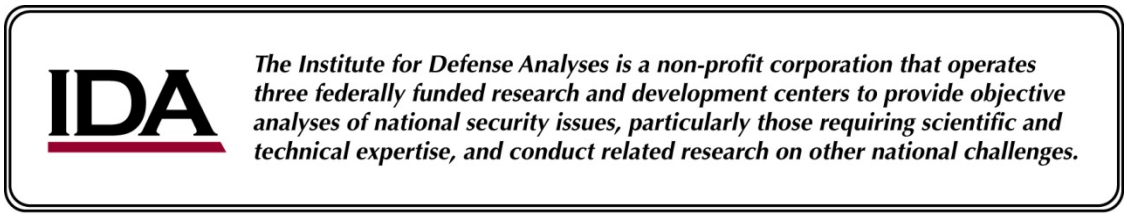
Kevin Foltz, *Project Leader*

William R. Simpson

May 2019

IDA Non-Standard
NS D-10634

INSTITUTE FOR DEFENSE ANALYSES
4850 Mark Center Drive
Alexandria, Virginia 22311-1882



About This Publication

This work was conducted by the Institute for Defense Analyses (IDA) under contract HQ0034-14-D-0001, Task BC-5-2283, "Architecture, Design of Services for Air Force Wide Distributed Systems," for USAF HQ USAF SAF/CIO A6. The views, opinions, and findings should not be construed as representing the official position of either the Department of Defense or the sponsoring organization.

For more information:

Kevin Foltz, Project Leader
kfoltz@ida.org, 703-845-6625

Margaret E. Myers, Director, Information Technology and Systems Division
mmyers@ida.org, 703-578-2782

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2019) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN/HCist 2019

ERP Homomorphic Encryption Performance Evaluation

Kevin Foltz[a,*], William R. Simpson[a]

[a]*Institute for Defense Analyses, Alexandria, Virginia, USA*

Abstract

Homomorphic encryption provides a way to keep sensitive data encrypted while operations are performed on it. It offers the possibility of hosting and processing sensitive data in an untrusted environment, such as a public cloud. However, the additional encryption affects performance, and it may cause degradation to latency or throughput. Special considerations are required when evaluating performance of an Enterprise Resource Planning (ERP) system with a homomorphically encrypted database. These result from different encryption types with different performance characteristics, combinations of encryption with non-linear performance behavior, and disparities between start-up and steady-state performance. In this paper, we describe the challenges of homomorphic encryption performance evaluation and some methods to obtain accurate and reliable results, and we present the application of these methods to the evaluation of a CryptDB-based system.

Keywords: homomorphic encryption; performance; latency; throughput; test; CryptDB

* Corresponding author. Tel.: +1-703-845-6625; fax: +1-703-845-6848.
E-mail address: kfoltz@ida.org

1. Introduction

Homomorphic encryption is a way to encrypt data such that the encrypted data can be directly manipulated without a decryption key. This differs from traditional encryption, where manipulation of data requires decryption, computation, and re-encryption. The ability to operate on data without having the decryption key means that a third party can perform the operations without knowing the underlying data. This security property can be used as part of a system that allows an untrusted party, such as a public cloud vendor, to process sensitive encrypted data.

Fully homomorphic encryption refers to encryption methods that permit any logical operation on the data. It is theoretically possible, but practically infeasible. *Partial homomorphic encryption* refers to encryption that permits a single type of operation on data, such as addition. *CryptDB* is an implementation that uses partial homomorphic encryption as part of an overall scheme to encrypt the contents of a database and permit a set of standard database queries on the encrypted data.

The figure below shows the basic architecture for using CryptDB. The proxy sits between the application and database. It encrypts content going into the database and decrypts content coming out. It stores encryption keys in encrypted form in the database itself. The user passwords are used to decrypt the appropriate encryption keys. Information about the encrypted schema and which parts of it each user is allowed to access are stored in the annotated schema. CryptDB user defined functions (UDFs) are stored in the database to help the proxy manipulate the database cryptographic state.

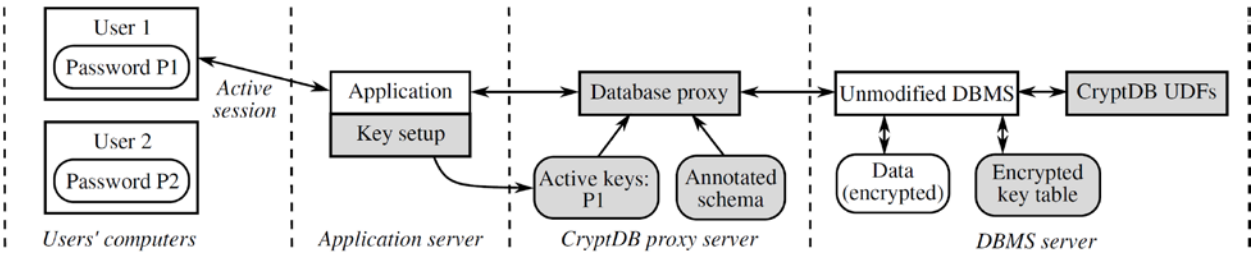


Fig. 1. CryptDB architecture.

Prior related work examined the feasibility and performance of such a setup [1]. It extended the original implementation of CryptDB [4] to include stored procedures, views, and referential integrity of primary/foreign key pairs. There has been some work to test performance of both homomorphic and partial homomorphic encryption. [2] describes a framework for homomorphic encryption performance testing, and [3] tests the raw cryptographic operations for fully homomorphic and partial homomorphic encryption methods. Our work looks at the techniques needed to test a homomorphically encrypted database and applies these to a test Enterprise Resource Planning (ERP) system with the extended CryptDB implementation.

Section 2 discusses the areas covered by evaluation, Section 3 describes the test setup, Section 4 presents the evaluation approach, and Section 5 provides selected test results to illustrate the evaluation approach and provide sample results.

2. Evaluation Areas

Performance evaluation covers two main areas: bulk encryption of an existing database and queries against the encrypted database.

2.1. Bulk Encryption

Bulk encryption is the process of converting an existing unencrypted database to an encrypted database. This process performs multiple encryptions of each data element according to the operations that will be performed on the data and the associated encryption types that support these operations. Evaluation of bulk encryption includes

assessment of the amount of time and computation resources required for a baseline data set and an evaluation of scalability to larger data sets.

The time and computation resources are important for determining the practical impact of bulk encryption. A bulk encryption is often performed offline. For operational systems, this requires shutting the system down while the encryption takes place to ensure data is not modified during the bulk encryption process. The bulk encryption time should not exceed a reasonable downtime for the system that uses the data.

Scalability is important for predicting performance with larger data sets. With a fixed set of computation resources, the total computation, and hence the total time, should scale nearly linearly with database size. However, the allowed downtime is often fixed, so another important area of scalability is parallelizability. If this also scales nearly linearly with data size, it means the bulk encryption can be divided up and completed in parallel in the same amount of time regardless of the data size.

2.2. Encrypted Queries

Query evaluation involves sending different types of queries to the encrypted database to determine performance characteristics. The primary performance measures are latency and throughput.

Latency includes both absolute and relative latency. *Absolute latency* is the amount of time a standard query takes to complete. This is important because longer delays from the encrypted queries can negatively impact the user experience. *Relative latency* is the percent increase in latency compared to unencrypted queries. A user might not notice an increase in latency from 10 ms to 20 ms for an individual query, but composite operations that aggregate many such queries would see significant latency increases.

Throughput is an issue as a system scales up. Even if latency is low for nominal request rates, decreased throughput will create scaling problems that eventually cause latency problems. Contention for critical central resources, such as cryptographic functions that are not parallelizable, throttles throughput as request rates increase. Synchronization causes delays when distributed resources are available but must be locked to maintain data integrity and consistency. The computation resources available provide a limit on overall throughput.

Homomorphic encryption methods that rely on shared internal state information face potential problems with critical central resources and synchronization. Increased resource requirements for performing complex encryption operations reduce the throughput for a fixed set of resources. Increasing the available resources would alleviate this problem, but it would increase costs.

3. Setup Considerations

The following are important considerations in setting up homomorphic encryption performance evaluation:

- Different queries invoke different cryptography with different performance characteristics,
- Combinations of queries with different cryptography are not linear, and
- Initial and steady-state behavior of cryptography can vary significantly.

Standard database testing looks at the database queries and other properties of the database. Homomorphic encryption adds another layer of consideration. It is now also necessary to invoke queries that stress particular types of encryption. Just as certain types of database queries are optimized for performance, the encryption can be optimized for performance based on how different data sets are used. Stressing particular types of encryption can highlight what these tradeoffs are so that appropriate measures can be taken operationally.

Due to the different ways the different types of encryption operate, some are more compute-intensive and others are more disk or memory-intensive. Some rely on the proxy to do most of the work, and others push the majority of the work to the database. As a result, stressing a single type of encryption is not sufficient to evaluate overall system performance under a diverse request set. Queries that use encryption types that stress the same resources may combine linearly, such that the total achievable request rate for a combination of requests is just an average of the achievable request rates for the individual requests.

However, if requests use different resources or distribute the resource requirements to different components, it is possible for the requests to receive more responses than the simple average would indicate. This assumes the throughput is bounded primarily by resources. With contention and synchronization, highly non-linear behavior is possible. Thus, it is important to test not just individual queries, but various combinations of queries in order to determine the performance for a system.

Testing different queries and combinations of queries results in a large number of test runs. Because the system state can become corrupted, inconsistent, or unknown after a test run, it may be necessary to reset the system state between runs. However, some of the homomorphic encryption implementations use shared state information that is generated as needed during encryption and decryption. This state information is cached and reused for future queries. This state information is not the data itself, but the supporting cryptographic structures that are used for computing encryptions and decryptions. These optimizations play an important role in making homomorphic encryption feasible, but we must consider the start-up performance as separate from the steady-state performance.

In a real system, it is likely that the steady-state performance is most important. For performance evaluation, it is easy to test start-up performance by issuing a small number of queries. Over time, the performance will converge to a steady-state value of latency and throughput. The challenge for testing is to determine the point at which steady-state performance is achieved. Averaging performance across the start-up and steady state provides unreliable numbers. The simple solution is to monitor performance until performance remains steady and then record data starting after that point in time. If the performance remains stable for a time period comparable to the initial transient performance, it is likely that it will remain stable. Accounting for longer-term performance issues is a separate problem. Garbage collection and other infrequent but performance-affecting events should be tested as well, but a longer-term test would be necessary.

In order to run the tests above, a method to generate requests and record results is needed. To generate requests, one approach is a simple web application that takes requests from a browser, translates them into database queries, and returns results in HTML pages. The application approach mimics a full ERP by translating user requests to database queries. A full ERP would have more complicated queries and sequences of queries where the relative performance is more important due to the additional latency of queries. To test such performance, queries can be issued repeatedly to generate higher system utilization that resembles that of an operational ERP.

Although requests are initiated by a browser, it is possible to use a tool to generate many such requests that appear to come from multiple browsers. This approach emulates the aggregated requests from many individual web browsers to the application. With the tool, it is possible to prepare collections of requests that test different types of queries and cryptography. Requests are assigned different weightings to form an overall request profile, where each request is repeated at a rate proportional to its weighting.

4. Evaluation Method

Performance evaluation consists of sending requests to the database through the application, waiting for responses, and recording timing information of the requests and responses in order to generate latency, throughput, and other metrics. Because many requests can be handled in parallel by the database, it is necessary to run multiple requester threads in parallel in order to stress the system and determine maximum performance. With a multi-core desktop, it is possible to achieve true parallelism up to the number of physical cores and virtual parallelism far beyond this limit.

One challenge is that the ramp-up period for multiple users can have a significant effect on performance. If all users start at the same time, they tend to cluster their requests together so that the system receives a group of the same requests, then a group of requests of another type, and so on. This degrades performance by focusing the resource utilization rather than spreading it out. To address this, ramp-up times must be chosen so that the different threads are spread across the request sequence. Sudden spikes and drops in CPU utilization, disk access, and network traffic indicate a failure to adequately space threads. Although such thrashing of resources is a good worst-case test, it is not likely to represent real-world behavior, where requests are made by independent requesters.

For throughput testing, the goal is to see how many requests the system can sustainably complete over a period of time. There are two approaches for this: adaptive and non-adaptive loads. For an adaptive load, multiple threads repeatedly send queries on each thread. New requests are only issued after previous requests are completed. This

keeps the number of parallel requests constant, but the request rate varies with the server's ability to send responses. As the number of threads rises, the throughput initially goes up as more parallelism is achieved. However, when the system is at maximum capacity, the extra threads queue up and wait, and the throughput stagnates while the latency rises. By incrementally increasing the thread count, a performance curve is generated for latency versus throughput; the "knee" in the curve indicates where maximum sustainable capacity is reached. Before the knee, throughput is still increasing with more users; after the knee, latency rises rapidly with little or no corresponding increase in throughput. With this adaptive load, the request rate is determined by the server's response rate. As the server completes more requests, it allows users to send more requests to the server. This is useful for simulating a situation where a few users are making long sequences of requests.

The non-adaptive load is comparable to a large pool of users making short requests. In this case, the request rate is fixed without regard to the server's ability to complete them. The queue length in this case can grow very large, unlike the adaptive load where the number of users, and hence the maximal queue length, is fixed. Non-adaptive loads better test system stability. The non-adaptive load will continue to send requests even as the old ones queue up, and sometimes a large queue of incoming requests can cause additional problems at the server. Thus, a temporary slow-down at the server, such as garbage collection, can cascade into a complete failure as queues form and then grow. Adaptive loads would simply wait until the server responses and hide such problems behind a lower throughput, but non-adaptive loads will maintain request rates and make such problems apparent.

For latency testing, timing information is collected and average latency values for each type of request are computed. It is helpful to first identify the range of achievable throughputs. Latency values are most meaningful when the system is not overloaded. By looking at the latency-throughput performance curves, it can be determined whether a latency measurement is due to real system latency or just queuing of requests due to throughput saturation.

To minimize the warm-up time, queries using the same cryptographic methods are grouped together, and tests are run sequentially on the warmed-up machine instead of restarting. This requires that the state of the system remains clean between tests. Some tests cause the server to fail, and this requires a clean restart and a new warm-up period before resuming tests.

To compute an average performance for a collection of requests, the individual queries are tested first. This shows the extreme cases where particular parts of the system are stressed. Then, combinations of queries show which queries combine well or poorly.

5. Test Results

This section presents selected results of applying the above evaluation approach to a CryptDB-based system. Testing of bulk encryption was performed on different data sets to assess scalability. The database was based on the Oracle human resources database provided with Oracle 12c. The employees table contained information about employees (one employee per row). To test different sizes of data sets, the number of employees in this table was varied: the smallest data set contained 10,000 employees, the next largest contained 100,000 employees, and the largest contained 1 million employees. Additional employees were generated randomly. Employee IDs were assigned sequentially. Salaries and other data were assigned randomly within ranges. Managers, locations, and other data that links to other tables were assigned randomly from available options. This provides a set of databases of different sizes that are functional and consistent.

Performance testing of bulk encryption computes the time to do the conversion from the original files to the encrypted files. The loading itself is generally much quicker than the encryption, so the encryption dominates the overall loading time for the encrypted data. In our test using dual 20-core processors in a Dell Precision desktop, the total encryption time for the database with 1 million employees was just under one hour. We deemed this to be feasible for our intended use case, but for other ERPs, a determination would depend on the hardware available, the data set size, and the desired execution time of the encryption.

For scalability, the three data sets were run, and the time for each increased roughly by a factor of 10 as the size increased. This shows that the times scale linearly with the data set size. For parallelization scalability, the tests were repeated on a desktop with 40 cores and a laptop with 4 cores. The speeds of the processors were comparable. The times for the laptop were roughly 10 times longer than those for the desktop, again showing linear scalability.

The combination of these results shows that the workload scales linearly with the size of the database, and the rate of work scales with the number of processors.

For query testing, a number of queries were used. Queries Q1 through Q6 returned large data sets, and performance depended primarily on highly variable network speeds, so these are not included. The remaining queries returned smaller result sets and include the following:

Q7: search for employees with particular department name, job ID, and manager ID values.
Q8: search for employees with particular job ID and manager ID values, and salary between two values.
Q9: search for employees with particular city, job ID and manager ID values.
Q10: search for employees with particular job ID and department name values, and salary between two values.
Q11: search for employees with particular department name, job ID, and manager ID values.
Q12: search for employees with particular job ID and manager ID values, and salary between two values.
Q13: search for employees with particular first name and job ID values, and phone number that contains a particular substring.
Q14: search for employees with particular last name and department name values, and phone number that contains a particular substring.
Q15: return salary information plus a raise value for employees with particular job ID, manager ID, and salary between two values.
Q16: Insert an employee
Q17: Update an employee
Q18: Delete an employee

For the queries under test, measurements of latency and throughput were made for different adaptive user loads, and requests were generated by the test tool JMeter. A sample of the results is shown in Figure 2.

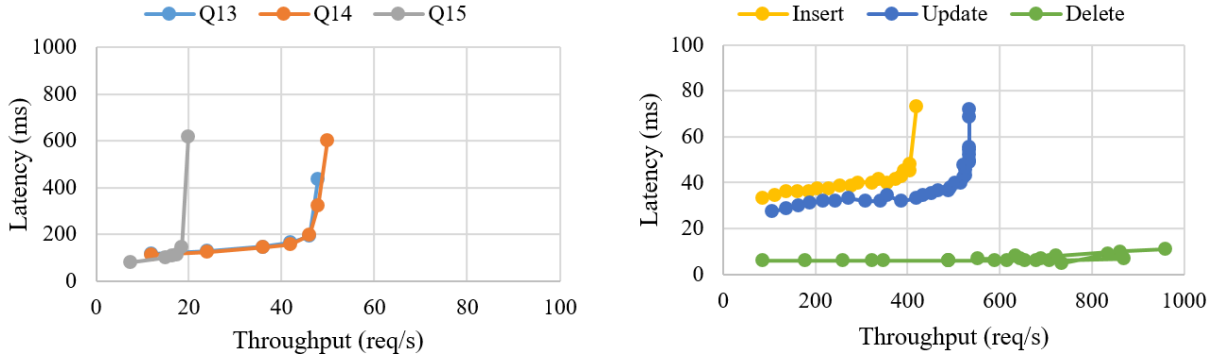


Fig. 2. Latency vs. throughput for individual queries.

Several key features are visible. First, as the load initially increases, the throughput increases with little effect on latency. Then, at a critical point, the latency rapidly increases while the throughput remains nearly unchanged. This was observed to be the saturation point, where the system resources are consumed and additional requests simply queue up and wait longer. Performance for individual queries is shown below. The throughput measurements are the values at the “knee” in the curve, which is the maximum sustainable throughput. Based on the initial adaptive load results, non-adaptive loads were generated around the knee value to find the highest stable request rate achievable. These are displayed in Table 1.

Table 1. Individual query throughput.

Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
600 req/s	308	587	172	592	663	43	45	18

To move from a single query to a mix, we first examined combinations of two queries. With baseline numbers for individual queries, it is possible to predict the aggregate throughput of combinations of requests. If the throughputs of queries A and B are T_A and T_B, then the amount of resources needed for a request of type A or B is $1/T_A$ or $1/T_B$, respectively. So, n of A and m of B would require resources $R = m/T_A + n/T_B$. Thus, this sequence would be repeated $1/R$ times per second, with total throughput of $n + m$ requests per iteration, which is $(n + m) / R = (n + m) / (n/T_A + m/T_B)$.

We computed these expected values and compared them to the actual numbers to determine which queries mesh well together. It is expected that actual performance will always improve compared to the predicted performance, because similar queries stress the system in the same way, and different queries have the potential to stress it in different ways, which improves internal parallelization and aggregate performance.

The expected performance is shown in the table below. The baseline numbers appear in gray along the diagonal. The values in the table are computed as above with $n = m = 1$, indicating an equal number of each query.

Table 2. Combined query expected throughput (requests per second).

	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
Q7	600	407	593	267	596	629	80	83	35
Q8	407	308	404	220	405	420	75	78	34
Q9	593	404	587	266	589	622	80	83	35
Q10	267	220	266	172	266	273	68	71	33
Q11	596	405	589	266	592	625	80	83	35
Q12	629	420	622	273	625	663	80	84	36
Q13	80	75	80	68	80	80	43	44	25
Q14	83	78	83	71	83	84	44	45	26
Q15	35	34	35	33	35	36	25	26	18

The actual performance of each combination of two queries was very close to the predicted values in most cases. To help identify areas of significant deviation, the table below lists the percent improvement in throughput of the actual above the predicted values. Improvements are green and reductions are red, with the darkness of color reflecting the magnitude of change from the baseline.

Table 3.Deviation from expected throughput.

	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
Q7		-1%	-2%	5%	-2%	3%	6%	5%	6%
Q8	-1%		-1%	16%	-2%	26%	11%	7%	9%
Q9	-2%	-1%		5%	-2%	2%	6%	5%	6%
Q10	5%	16%	5%		6%	4%	3%	0%	14%
Q11	-2%	-2%	-2%	6%		4%	7%	5%	6%
Q12	3%	26%	2%	4%	4%		5%	1%	8%
Q13	6%	11%	6%	3%	7%	5%		2%	51%
Q14	5%	7%	5%	0%	5%	1%	2%		45%
Q15	6%	9%	6%	14%	6%	8%	51%	45%	

The most gains are seen when Q15 is combined with Q13 or Q14. This indicates that Q15 has very different resource requirements than Q13 and Q14. Upon closer examination, it was observed that Q15 created a very high computational load on the CryptDB proxy due to the complicated encryption and decryption associated with additions, while Q13 and Q14 produced a higher load on the database server due to the invocation of a UDF to perform searching of character strings.

Due to the test tool (JMeter) and its ease of use with static requests, all testing was done with the same sequence of queries that had the same values for all queries. This raised the question of whether the query and response values were being cached in the database. In order to test this, a variation of Q10, called Q10R, was implemented that used random values for the salary range.

The random values created one million possible queries. The test was run first sequentially, where 300 iterations of Q10 were followed by 1,300 iterations of Q10R. Results showed slightly higher latency for the random queries,

suggesting some internal caching of results. Then, queries alternated between Q10 and Q10R for 1,600 queries. The same overall performance improvement was observed, with a quick drop from a high latency to around 150 ms and a second drop to around 50 ms. These were observed to be synchronous for the alternating queries, showing that the cryptographic state is the largest component of the performance improvement, and the improvement for deterministic over random values is persistent but small.

The first 10 points represent individual values. Points between 10 and 100 represent the average of 10 trials; points at 100 and more represent the average of 100 trials. Iteration numbers represent the average of the trials (e.g., trials 201–300 are plotted at 250.5).

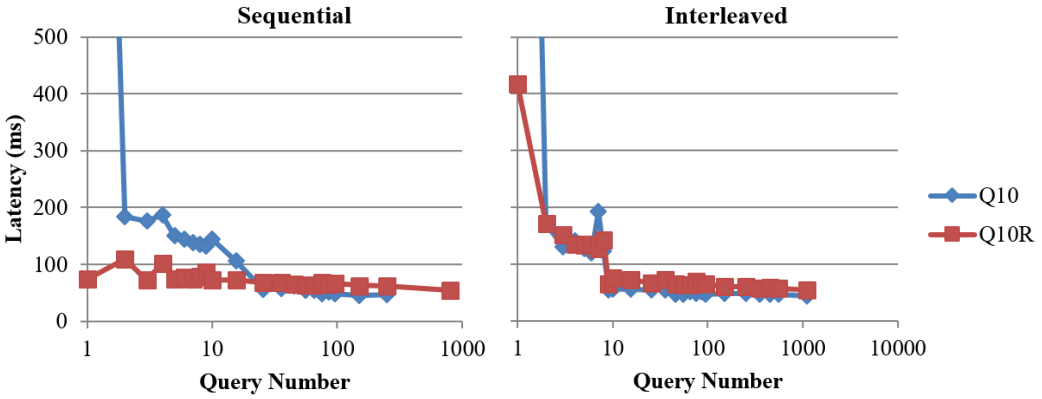


Fig. 6. Initial latencies for Q10 and Q10R – sequential and interleaved.

The latency for Q10R is about 23% higher. However, even on unencrypted data, the latency is about 9% higher for random values. This means the effect of randomization is really only a 13% (1.23/1.09 – 1) penalty due to CryptDB for random data versus deterministic data. This is not negligible, but small enough to conclude that deterministic queries are an adequate representation of random queries. The situation for any real-world system is likely to lie somewhere between these values depending on the type of application and the degree of user influence on the values in the database queries from the application.

6. Conclusions

With the increasing concern over privacy of data and the concurrent increase in public cloud computing, homomorphic encryption of databases is a promising component of a solution to preserve privacy while leveraging the power of the cloud model. An accurate assessment of performance is an important step in implementing such solutions. This paper describes potential challenges when evaluating performance of a homomorphically encrypted database as part of an ERP. It describes evaluation approaches to provide accurate and reliable results. It applies these to an implementation of CryptDB and presents the performance results.

References

[1] K. Foltz and W. R. Simpson, “Extending CryptDB to Operate an ERP System on Encrypted Data,” *Proceedings of 20th International Conference on Enterprise Information Systems* (ICEIS 2017), Funchal, Madeira, Portugal, March 21–24, 2017, pp. 103–110.

[2] M. Varia, S. Yakoubov, and Y. Yang, *HEtest: A Homomorphic Encryption Testing Framework*, presented at 3rd Workshop on Encrypted Computing and Applied Homomorphic Cryptography, January 2015.

[3] R. A. Al-Shibib, *Performance Analysis for Fully and Partially Homomorphic Encryption Techniques*, Masters thesis, Middle East University, Amman, Jordan, May 2016.

[4] R. A. Popa et al. *CryptDB: Protecting Confidentiality With Encrypted Query Processing*. ACM Press, 2011. 85.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE (DD-MM-YY)	2. REPORT TYPE	3. DATES COVERED (From – To)
00-04-19	Non-Standard	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
ERP Homomorphic Encryption Performance Evaluation	HQ0034-14-D-0001
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBERS

6. AUTHOR(S)	5d. PROJECT NUMBER
Kevin Foltz, William R. Simpson	BC-5-2283
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESSES	8. PERFORMING ORGANIZATION REPORT NUMBER
Institute for Defense Analyses 4850 Mark Center Drive Alexandria, VA 22311-1882	NS D-10634

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR'S / MONITOR'S ACRONYM
Frank P. Konieczny United States Air Force SAF-CIO A6, 1800 Air Force Pentagon, Washington DC 20330-0001	USAF HQ USAF SAF/CIO A6
	11. SPONSOR'S / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

Project Leader: Kevin Foltz

14. ABSTRACT

Homomorphic encryption provides a way to keep sensitive data encrypted while operations are performed on it. It offers the possibility of hosting and processing sensitive data in an untrusted environment, such as a public cloud. However, the additional encryption affects performance, and it may cause degradation to latency or throughput. Special considerations are required when evaluating performance of an Enterprise Resource Planning (ERP) system with a homomorphically encrypted database. These result from different encryption types with different performance characteristics, combinations of encryption with non-linear performance behavior, and disparities between start-up and steady-state performance. In this paper, we describe the challenges of homomorphic encryption performance evaluation and some methods to obtain accurate and reliable results, and we present the application of these methods to the evaluation of a CryptDB-based system.

15. SUBJECT TERMS

homomorphic encryption; performance; latency; throughput; test; CryptDB

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Frank P. Konieczny
Unclassified	Unclassified	Unclassified	Unlimited	8	19b. TELEPHONE NUMBER (Include Area Code) 703-697-1308