Extending CryptDB to Operate an ERP System on Encrypted Data

Kevin Foltz and William R. Simpson
Institute for Defense Analyses, 4850 Mark Center Drive, Alexandria, VA 22311, U.S.A.

Keywords: Enterprise, Enterprise Resource Planning, Database, System Design, Confidentiality, Integrity, Homomorphic Encryption, Partial Homomorphic Encryption, Application Security, Security, Cloud Services, End-to-End Encryption, Key Management, Database Security.

Abstract: Prior work demonstrated the feasibility of using partial homomorphic encryption as part of a database encryption scheme in which standard SQL queries are performed on encrypted data. However, this work involved only translating raw SQL queries to the database through the CryptDB proxy. Our work extends the prior work to an Oracle application. The goal for this work was to determine feasibility for a full-scale implementation on a real Oracle Enterprise Resource Planning (ERP) system. This requires accommodating extra features such as stored procedures, views, and multi-user access controls. Our work shows that these additional functionalities can be practically implemented using encrypted data, and they can be implemented in a way that requires no code changes to the ERP application code. The overall request latency and computational resource requirements for operating on encrypted data are under one order of magnitude and within a small factor of those for unencrypted data. These results demonstrate the feasibility of operating an Oracle ERP on encrypted data.

1 INTRODUCTION

Homomorphic encryption provides a way to manipulate encrypted data to perform computations on the underlying plaintext without decrypting the data. It provides added security for hosting in a cloud, where sensitive data may be accessible to an untrusted third party. By homomorphically encrypting data prior to storing it in the cloud, the data can be used for computations while remaining protected. This stops threats to confidentiality posed by the cloud provider, its employees, and any external entity that may compromise the cloud provider.

Full homomorphic encryption (FHE) allows any sequence of operations to be performed on the encrypted data, so any computation can be done encrypted (Gentry, 2009). However, FHE is prohibitively slow for all but the simplest of computations (Gligor, 2014). Partial homomorphic encryption (PHE) has higher performance, but it allows only a single type of operation, such as addition or multiplication. An extensive survey of homomorphic encryption methods is provided in (Acar et al., 2017).

Our prior work suggested that FHE was not practical, but methods using PHE showed promise (Foltz and Simpson, 2017). Research involving PHE has shown that a SQL database (DB) can be encrypted such that standard SQL queries can be run against this encrypted database (Popa et al., 2011). The encryption is performed by the CryptDB proxy, which is located between the database requester and the database. The CryptDB proxy translates queries on unencrypted data into queries on encrypted data, allowing a user to access the encrypted database as if it were not encrypted. The CryptDB proxy also translates encrypted responses to unencrypted responses.

Real systems are not as simple as a single database. A typical Enterprise Resource Planning (ERP) system has the following additional complications:

- Proprietary ERP code that cannot be changed,
- Primary and foreign key reference integrity,
- Stored procedures,
- Views, and
- Multiple accounts with different permissions.

The original CryptDB implementation used MySQL and did not account for these complications.

Foltz, K. and Simpson, W.
Extending CryptDB to Operate an ERP System on Encrypted Data.
DOI: 10.5220/0006661701030110
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 103-110
ISBN: 978-989-758-298-1

Our work ports CryptDB from MySQL to Oracle's SQL database and addresses the items listed above. This demonstrates that CryptDB can be integrated with existing operational Oracle ERP systems and is not restricted to the research laboratory or custom-built systems.

2 RELATED WORK

There are many approaches to securing database information. Monomi, another CryptDB-based approach, allows more complex queries by splitting computation between server and client (Tu et al., 2013). The L-EncDB system uses techniques to preserve the data formats and lengths between unencrypted and encrypted data (Li et al., 2015).

The BigSecret system secures NoSQL databases using property-preserving encryption on indices of data encrypted with standard techniques (Pattuk et al., 2013). A modular framework for providing varying degrees of privacy and performance for NoSQL databases is provided in (Macedo et al., 2017).

3 RESEARCH METHODS

This section presents the experimentation and test process. Maintaining correct functionality was required. Experiments determined relative performance in encrypted searches. This work consists of the following steps:

1) Select database,
2) Determine the selected database schema,
3) Develop partial homomorphic encryption schemes,
4) Perform credential mapping,
5) Develop SQL translation schemes,
6) Develop a web application test harness,
7) Establish a set of nominal work queries, and
8) Measure performance.

The implementation was performed in a lab with commodity hardware and software, with the exception of the CryptDB proxy and other code and scripts written specifically for this work.

3.1 HR Database Selection

This work used the Oracle 12c database system and the sample HR database that is provided with it by Oracle. This database was used as the starting point for development and testing. It provided an adequate test system for many of the complexities of a full-scale HR ERP system.

3.2 HR Database Schema

The HR DB schema consists of seven tables:

- Employees,
- Jobs,
- Job_History,
- Departments,
- Locations,
- Countries,
- Regions.

The Employees table schema is shown in Figure 1. The HR schema and the data for the HR DB were used for unencrypted operation of the ERP and served as the baseline for performance measurements.

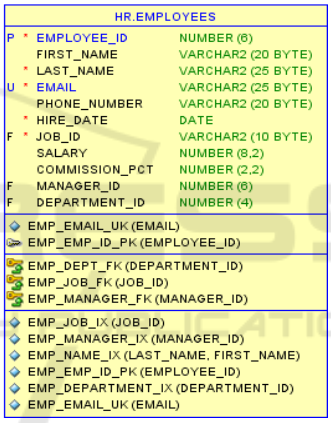

Figure 1: HR Schema for Employees Table.

For encrypted operation, the HR DB schema had to be modified for encrypted data. This involved modification of data types and sizes to accommodate encrypted data. It also included additional columns for multiple encryption methods where required. The names of tables and columns were encrypted as well. The ERP was not modified for these changes. It communicated through the CryptDB proxy, which mapped requests from the unencrypted schema to the encrypted schema.

3.3 Encryption Schemes

Different encryption methods offer different properties that are useful for operations on encrypted data. The proper encryption scheme for data depends on its intended use. The encryption methods and supported SQL operations are shown in Table 1.

Table 1: Encryption Methods.

Type	Encryption Method	SQL Operation
RND	AES in CBC mode	None
HOM	Paillier	Addition
SEARCH	*	Word Search
DET_EQ	AES in CBC mode	Equality
OPE	*	Order
EQ_JOIN	*	Join
OPE_JOIN	*	Range Join

* custom encryption methods

These encryption methods are listed in approximate order of security protection, from highest to lowest. Security protection decreases as more information is revealed about the data. For example, assuming the cryptographic methods are known, random encryption reveals essentially no information about the plaintext. Deterministic encryption (DET) allows determination of whether two encrypted values have the same plaintext values without revealing what this plaintext is. Order-preserving encryption (OPE) leaks the relative size of the plaintext values without revealing the values themselves.

Generally speaking, the encryption methods reveal the amount of information needed to perform the associated SQL operation. A determination of whether this amount is acceptable must be made at design time (Akin and Berk, 2015; Naveed et al., 2015). For this work, we assume this is acceptable for our intended use cases. In cases in which this is not acceptable, there are methods to reduce it, and these are briefly mentioned as future work.

Because different encryption schemes may reveal different information when the cryptographic methods are known, it is possible to layer them. Plaintext can be encrypted with EQ_JOIN, and this can again be encrypted with DET. We refer to a data element that has undergone this multi-layer encryption as an "onion," with each different encryption method being a layer of the onion. The four encryption onions listed below are used for this work. The EQ onion is illustrated in Figure 2.

1) Search: SEARCH
2) Add: HOM
3) EQ: RND, DET, EQ_JOIN
4) Ord: RND, OPE, OPE_JOIN

Figure 2: EQ Onion Layering.

For situations in which these innermost onion layers are rarely used, the onions-and-layers approach provides increased security by revealing the least secure inner layers only when needed. For simplicity and performance, it is also possible to use only the innermost layer.

3.4 Credential Mapping

To provide proper access to different users, different accounts are set up within the ERP and database. The ERP uses its accounts and credentials to make queries to the database. To use these accounts through the CryptDB proxy, separate accounts must be created for the encrypted database. The CryptDB proxy maps the existing accounts for the unencrypted database to accounts for the encrypted database.

Along with the accounts are keys and other information that a user must have to encrypt requests and decrypt responses. These keys are themselves encrypted using information from the user account so that only the appropriate accounts have access to the proper cryptographic keying material. Other accounts may be able to access the database of ERP cryptographic keys, but they will not be able to decrypt and use keys for which they are not authorized.

3.5 SQL Translation Schemes

With the encryption methods and accounts set up, we need a way to translate an unencrypted request to an encrypted request. This translation depends on the type of data and the type of encryption. For normal requests, the column names and data are encrypted. For other requests, such as addition, the query itself must be modified. For example, with Paillier encryption addition of plaintext corresponds to multiplication of ciphertext.

Other translations are similar. For searching, text strings must be converted to binary values. If a numerical value is part of a query, it may need to be encrypted in multiple ways if it is used with data in different columns with different encryption methods or keys.

With multiple users, which user will be calling a stored procedure is not known, so encrypted values in stored procedure queries cannot be pre-determined. In this case, these values must be inserted at execution time by determining the user and updating the query based on the appropriate user-specific information.

3.6 Web Application

To test an encrypted versus unencrypted database, a web application was implemented to act as a frontend for both the encrypted and unencrypted databases. This web application runs natively on the unencrypted database, and it can send requests to the CryptDB proxy for operation on the encrypted database. No code changes are needed in the application itself. The switch to operate on encrypted data is simply a configuration change.

In addition to the web application frontend, various tools allowed more in-depth testing of the encrypted database. Direct queries of the encrypted database are possible using the sqlplus tool, but due to the encrypted values, it is difficult to create the requests or interpret the results. A script, cryptdb-sqlplus was developed to make a subset of such queries on the encrypted database. It uses the CryptDB keys and mappings to translate and send user requests to the encrypted DB and translate responses.

3.7 Assessments

Assessment of the encrypted database consists of three parts. The first part validates that each implemented capability works as expected. A sequence of requests is made to the unencrypted and encrypted databases, and the results are compared. Identical results confirm that the capabilities are implemented correctly.

The second part consists of performance tests in which the same sequence of queries is sent to the same application, first using the unencrypted database and then using the encrypted database through CryptDB. Latency and throughput of the encrypted database queries are compared to the values for the unencrypted database.

The third part consists of multiple users with simultaneous access, to confirm that the multi-user access controls are performing properly and not negatively affecting performance.

3.8 Lab Setup

The development work for the encrypted database was done on a single machine. This allowed functional testing and very limited performance testing. For full performance testing, a dedicated lab was set up with the following equipment:

- Database server laptop;
- Application server laptop;
- Multi-use desktop for application server, loader, and client;
- Additional client laptop; and
- Ethernet switch.

The application server was a Dell Mobile Precision 7710 laptop with four cores @ 3 GHz, 64 GB memory, 2 TB SSD, and 1 Gb/s Ethernet running Windows 10.

The database server was identical to the application server, except storage was two 1 TB PCIe drives in a RAID 0 configuration.

A Dell Precision Tower 7910 served as multi-use desktop. It had dual 20 core processors @ 2.2 GHz, 512 GB memory, a 2 TB SSD, and 1 Gb/s Ethernet running Windows 10.

The additional client laptop had two cores @ 2.8 GHz, 8 GB memory, and 100 Mb/s Ethernet running Windows 10.

Connecting these machines was an eight-port gigabit Ethernet switch.

Software included the following:

- Oracle 12c database – database to store both unencrypted and encrypted data;
- H2 database – database to store the CryptDB keys and other cryptographic and mapping data;
- CryptDB proxy – ported to Oracle from original MySQL implementation;
- WebLogic 12.1.3 – web server for the test application;
- Java JDK 7 update 80 for Windows x64 – platform for various applications;
- JMeter – test tool to execute test scripts and capture performance data;
- Bouncycastle – library for cryptographic operations;
- Cygwin – application that enables Linux-style scripting within a Windows operating system;
- Various scripts and tools developed for automating the setup of the encrypted database and loading the encrypted data into the encrypted database.

4 RESULTS

This section presents the results for tests of proper functionality and performance.

4.1 Proper Functionality

Porting the MySQL-based CryptDB implementation to an Oracle-based implementation was tested first.

Direct queries on the encrypted database showed that the tables, columns, and values were properly encrypted. Queries through the CryptDB proxy on the encrypted database provided results that matched results from the unencrypted database. This confirmed the proper functioning of the encrypted database and CryptDB.

4.2 Enhancements

Next, certain improvements were made to the CryptDB implementation, including referential constraints, views, and stored procedure capabilities. These features, which are required for an ERP instantiation, were not part of the original MySQL implementation.

Stored procedures in the unencrypted database are converted to stored procedures that perform equivalent operations on the encrypted database. This involves encrypting raw data in the procedures, mapping table and column names, changing operations (such as addition to multiplication for Paillier encryption), and changing commands where appropriate. Stored procedures use the PL/SQL language, and for this work, a core subset of PL/SQL was implemented to enable testing of stored procedures.

Referential constraints are important for primary key and foreign key references. When the primary and foreign keys are encrypted with different encryption keys, the database cannot guarantee integrity across these two columns. CryptDB has been modified to recognize these references and add two additional columns that contain keyed hashes of the primary and foreign keys. The keyed hash of the foreign key is set to reference the keyed hash of the primary key. CryptDB uses the same key to generate the two columns of hashes, which allows the database to maintain referential integrity of the primary and foreign keys through these extra columns.

Views are like tables, but they have some important differences that require special consideration for implementation. A table has one column for random initialization vectors (IVs). However, a view must have multiple such columns because it may be constructed as a join across multiple tables, each with its own IVs. CryptDB must also maintain the mappings from each encrypted column to its IV column.

Like tables, views can operate at different onion layers of encryption. If these layers are unwrapped or rewrapped on the underlying tables, the views must keep track of these changes. A query on a view that requires adjusting onion layers also requires changes to the base tables. As a result, the query translation for views is more complicated, and the resulting query may contain additional queries to change onion layers on the view and associated tables, and changes to internal state to keep track of the view-to-table mappings.

Stored procedures are groups of SQL statements and control statements. They offer many benefits, including performance, security, scalability, and maintainability. However, they require CryptDB to be extended to support creating, dropping, and calling these stored procedures.

Additional complications arise in stored procedures. The same encrypted value in a query may be used across multiple columns and onions. Instead of simply encrypting a value once, it must be encrypted for each possible use. With multiple users, a method for determining which principal is making a request is required to select and use the proper key.

An additional challenge is that a result may be from one or another column, and which column the value comes from is determined at runtime. In this case, CryptDB must keep track of which column, onion, and principal a response came from, and use this additional information for proper decryption.

The issues of multi-user stored procedures and dynamic column determination have not been addressed yet, and these present challenges for future work. However, the solution concepts are generally understood, so the challenge is simply implementing them.

4.3 Performance

Performance testing was conducted to assess how well suited this new CryptDB implementation is for real-world implementations.

4.3.1 Operational Latency and Throughput

Operational performance testing was conducted through the web application using a web browser. For normal queries that completed quickly, there was no noticeable difference in latency. For longer queries that returned hundreds of thousands of results, the difference was noticeable. However, upon closer inspection, much of the time was actually due to the larger data sizes for encrypted data and the correspondingly longer time to transmit them over Ethernet. Although this larger transmission time is a valid concern for real-world systems, it is only observed in queries with very large result sets.

For a more comprehensive analysis, JMeter was used as a test tool with a set of queries that represented a normal mix for typical business. The mix of queries included at least one use of each of the following:

- Insert, Update, and Delete,
- DET-based queries,
- ORD-based queries,
- HOM-based queries.

In general, HOM, which is implemented using Paillier encryption, is the most costly. It imposes a significant computational burden on the Application/ CryptDB server beyond not only the unencrypted queries but also the other encrypted queries. ORD is also computationally expensive, but less so. The others impose little additional overhead. The mix of queries used for testing consisted of the following:

1) 3% Get all managers (35% of all employees returned),
2) 2% Search on employees (3% returned),
3) 1% Search on employees (.15% returned),
4) 1% Search with salary range (1% returned),
5) 1% Search on employees (85% returned),
6) 3% Search on employees (5% returned),
7) 10% Search on employees (few returned),
8) 5% Search with salary range (ORD) (few returned),
9) 5% Search on employees (few returned),
10) 10% Search with salary range (ORD) (few returned),
11) 10% Search on employees (few returned),
12) 5% Search for employees with salary range (ORD) (0 returned),
13) 2% Search for partial match in phone number (1 returned),
14) 2% Search with partial match (1 returned),
15) 5% Search with addition of a constant to results (HOM) (.2% returned),
16) 15% Insert an employee (Ins),
17) 15% Update an employee (Upd),
18) 5% Delete an employee (Del).

Queries 1 through 6 return large data sets, and it is anticipated that these types of queries would be executed occasionally only by users with special privileges. Queries 7 through 18 represent standard application queries that would be expected during ongoing business operations.

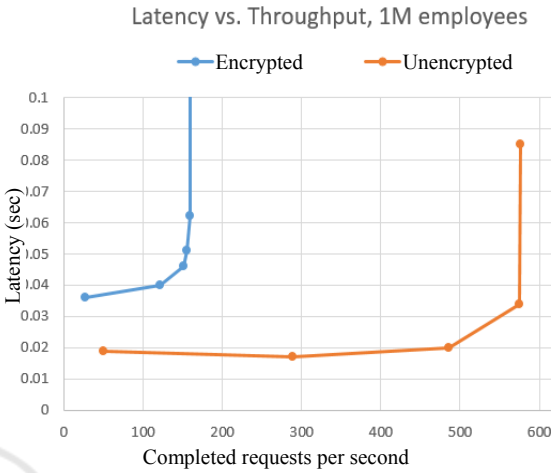

Figure 3: Performance Summary, Queries 7–15.

To compare the user experience and resource requirements for encrypted operation versus unencrypted operation, increasing numbers of active threads repeatedly cycled through queries 7 through 15, with the relative ratios as listed. This excludes the queries with large result sets as well as the insert, update, and delete operations, and it is intended to represent normal user queries.

Figure 3 shows the observed latencies and throughputs on the databases with one million users. For this mix, the baseline latency was approximately double with the encrypted database (36 ms vs. 19 ms), and achievable throughput was about 25% (155 req/s vs. 575 req/s). This suggests that encrypted operation for these queries would require four times the computational resources as unencrypted operation to support the same request rates, and latency would be approximately double under these conditions.

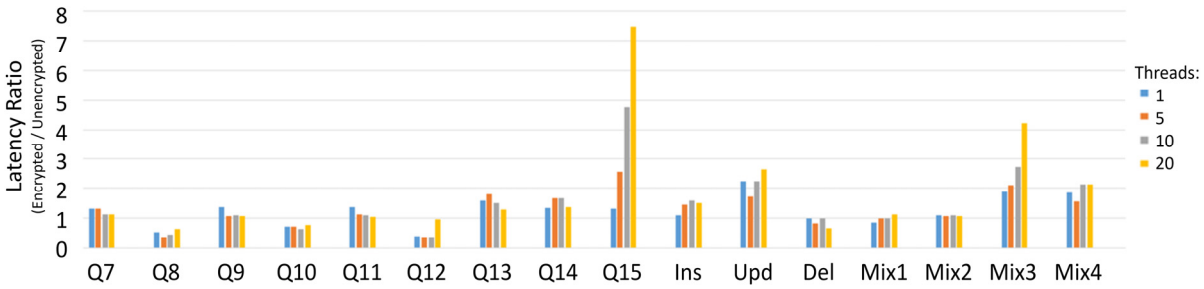

Figure 4: Latency Ratios for Individual Queries and Mixes of Queries.

Additional testing was performed for each individual query and for some combinations of queries. Latency results are shown in Figure 4. Queries 7 through 15 (Q7 – Q15), inserts (Ins), updates (Upd), and deletes (Del) were tested individually. Mixes 1 through 4 were as follows:

1) Q7-15, Ins, Upd, Del;
2) Q8, Q10, Q12;
3) Q7, Q9, Q11, Q13-15, Ins, Upd, Del;
4) Q7, Q9, Q11, Q13, Q14, Ins, Upd, Del.

Mix 1 is the baseline experience, representing a normal mix of queries. Mix 2 examines just Queries 8, 10, and 12, which individually show improved performance on encrypted data. In combination, they show no change compared to unencrypted data. Mix 3 excludes Queries 8, 10, and 12 from the baseline and shows degraded performance compared to the baseline. Mix 4 is similar to Mix 3, but it excludes the computationally intensive Query 15, revealing that the degradation with increasing users was due primarily to Query 15.

Query 15 performance degrades roughly linearly with increasing request loads beyond five users. This is because throughput is already saturated on the encrypted system and additional queries simply queue up and wait longer. For these tests the CPU approached 100% utilization at seven users.

The Paillier encryption (HOM) of Query 15 is the biggest bottleneck for performance and is the most likely target for performance improvements in a real system. The simple solution is to allocate more resources to this type of encryption, either through more servers, higher-performance servers, or offloading the Paillier computations to more efficient and parallelized GPUs rather than faster but less parallel CPUs.

Although full multi-user functionality that includes the use of stored procedures is not yet implemented, performance testing was conducted for multiple users on queries that do not use stored procedures. For such queries the test tool JMeter was launched with up to 100 parallel threads of execution on a single machine. This simulates a high load from many different users. This leveraged the powerful desktop's ability to run many threads. Tests were also run from multiple requester machines using multiple requester threads on each machine.

Performance for multiple users showed no significant degradation compared to the single user case until CPU and other resources were nearly consumed. No problems with encryption-related locking of database resources were observed.

4.3.2 Data Load Time

The main question to answer for loading is whether encryption adds significantly to the loading time. Even when query performance is acceptable, excessive time to encrypt and load the data may create complications when a database must be reloaded. Testing with a high-performance commodity machine showed that encrypting the database with one million employee records can be completed in about one hour.

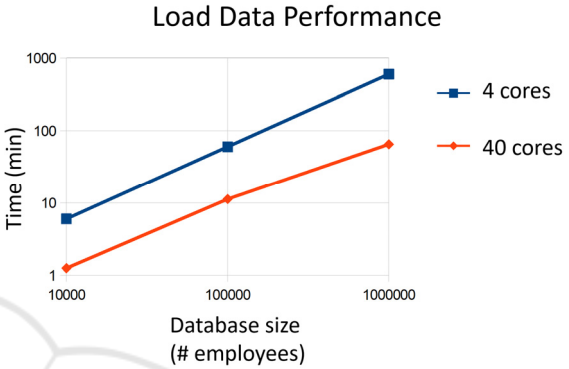

Figure 5: Database Loading Times.

The times are nearly linear in the size of the database to be loaded. The speed of encryption is also nearly linear in the number of processors. Thus, the encryption time can be reduced by increasing parallelism. Figure 5 shows results for single machines with 4 and 40 cores.

5 CONCLUSIONS

This work assesses whether running an ERP on an encrypted database is feasible. Prior work showed that basic queries without the full complexities of a real system were possible, but it was not clear whether the additional complexities of a real system could be accommodated and if so, whether the performance would be acceptable.

The primary conclusion is that the Oracle-based CryptDB system shows strong feasibility for real-world implementations. It supports referential integrity, views, and stored procedures, which are three major areas of concern for real systems that were not previously implemented.

The performance measured across individual queries and in aggregate showed a small degradation compared to the same queries on unencrypted data. This degradation varies among the individual

queries, so a careful consideration of the mix of queries is important.

Overall computational resource requirements are increased due to the need for encryption, extra logic, and processing, with an overall increase of a small factor over the unencrypted computing resources. This factor may vary depending on the particular application and mix of queries.

Performance with multiple users showed good scalability, with no observable encryption-related latency.

Using CryptDB with an encrypted database is feasible for moving a sensitive database to an untrusted cloud hosting environment. The latency performance is comparable to the use of an unencrypted database, and comparable throughput can be achieved with additional resources to support the encryption-related computation.

6 FUTURE WORK

Follow-on work to this study includes testing on an operational Oracle ERP system under normal use cases and workflows.

Additional extensions and improvements are planned for CryptDB, and PL/SQL support is to be expanded. Performance improvement for Paillier encryption may be possible using GPUs, which should improve performance and reduce the load on the CPU. This will provide the benefits of improved scalability for Paillier encryption and reduced CPU contention for other queries.

It was noted earlier that there are possible leakages of information about plaintext through some of the encryption schemes. For example, relative sizes and distributions of numbers can be calculated for OPE encryption, which could lead to a few known values revealing other encrypted values.

This leakage cannot be completely eliminated, but it can be reduced by various methods. First, additional entries can be added to the database to smooth out the distribution of values. Additional queries would be inserted periodically to access these otherwise unused values. Second, existing entries with the same values can be split into different categories by CryptDB so that they appear different in the database. Third, encryption keys can be changed periodically. These all impose a resource burden on the system through additional storage and computation.

ACKNOWLEDGEMENTS

The authors wish to acknowledge Virgil Gligor for his deep insights and broad knowledge in homomorphic encryption and related areas.

REFERENCES

Acar, A., Aksu, H., Uluagac, A. S., and Conti, M. A. Survey on Homomorphic Encryption Schemes: Theory and Implementation. arXiv preprint arXiv:1704.03578v1, April 12, 2017.

Akin, I. H., and Berk, S. 2015. “On the Difficulty of Securing Web Applications using CryptDB,” International Association for Cryptologic Research. Available at https://eprint.iacr.org/2015/082.

Foltz, K. and Simpson, W. Enterprise Level Security with Homomorphic Encryption. In Proceedings of 19th International Conference on Enterprise Information Systems (ICEIS 2017), Porto, Portugal, April 26–29, 2017.

Gentry, C. 2009. “A Fully Homomorphic Encryption Scheme.” Doctoral thesis. Stanford University. Available at https://crypto.stanford.edu/craig/craig-thesis.pdf.

Gligor, V. 2014. “Homomorphic Computations in Secure System Design,” Final Report. Pittsburgh, PA: Carnegie Mellon University.

Li, J., Liu, Z., Chen, X., Xhafa, F., Tan, X., and Wong, D. S. “L-encdb: A lightweight framework for privacy-preserving data queries in cloud computing,” Knowl.-Based Syst., vol. 79, pp. 18–26, 2015.

Macedo, R. et al., “A Practical Framework for Privacy-Preserving NoSQL Databases,” 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS), Hong Kong, 2017, pp. 11–20.

Naveed, M., Kamara, S., and Wright, C. V. “Inference Attacks on Property-Preserving Encrypted Databases.” In: CCS’15, Denver, CO, 2015.

Pattuk, E., Kantarcioglu, M., Khadilkar, V., Ulusoy, H., and Mehrotra, S. “Bigsecret: A secure data management framework for key-value stores,” International Conference on Cloud Computing, 2013.

Popa, R. A., Redfield, C. M.S., Zeldovich, N., and Balakrishnan, H. 2012 “CryptDB: Processing Queries on an Encrypted Database,” Comm. ACM, vol. 55, no 9, Sept. 2012 (also Proc. of 23rd ACM SoSP, Sept. 2011).

Tu, S., Kaashoek, M. F., Madden, S., and Zeldovich, N. “Processing analytical queries over encrypted data,” Proc. VLDB Endow., 2013.