Micro-Benchmarking Property Preserving Encryption: Balancing Performance, Security and Functionality

Daniel Becker
School of Computer Science
University College Dublin
Dublin, Ireland
daniel@becker.ie

Andrew Hines
School of Computer Science
University College Dublin
Dublin, Ireland
andrew.hines@ucd.ie

***Abstract*—Practical encryption systems with new and more flexible capabilities have been enabled by recent advances in computing hardware performance and Property Preserving Encryption (PPE) schemes. PPE schemes allow limited and pre-selected operations to be performed on encrypted data allowing system designers to trade-off between performance, security and functionality. This paper uses micro-benchmark to evaluate three interdependent factors of PPE: performance, security and functionality. The findings validate the efficacy of this technique and provide guidance to application designers and technology evaluators seeking to understand these interdependent relationships for PPE database applications. Experiments were performed using the *CryptDB* research system. Results validate the previous assessments of *CryptDB* and provide supplemental detail on performance, security and functionality.**

***Index Terms*—Encryption, Performance, Security, CryptDB, Micro-Benchmark, Property Preserving Encryption**

I. Introduction

Systems using Property Preserving Encryption (PPE) [1] schemes can provide broad functionality, with alternative schemes allowing differing subsets of operations to be performed on encrypted data. While these schemes permit a range of operations, they have different performance and security characteristics and cannot be used interchangeably.

Previous evaluations have used blended and full stack macro-benchmarks to provide useful overviews of the aggregate performance and functional characteristics of systems but they do not provide sufficient detail to inform design decisions. End-to-end performance measures can be dominated by factors other than data retrieval; for example, the measured total client-server round-trip time for a typical web application can include: proxy negotiation, DNS lookups, connection establishment, SSL negotiation, request sending, server pre-processing, multiple database calls, server post-processing and content transfer. Conversely caching layers in application software stacks can improve measured performance by reducing database calls. Benchmarks that synthetically represent the intended workload of a system such as the `SQL TPC-C` [2] or workloads based on execution of specific applications make it difficult to isolate system performance factors, and the distribution of sample query types may not be representative for the target application.

In contrast, in this paper we use a *Micro-Benchmarking* evaluation approach. Micro-benchmarks provide a mechanism for specific feature evaluations in isolation; as well as full system characterisation. They are also useful for application and system designers as a regression tool to evaluate the performance of system updates. In this study, a micro-benchmark will refer to the *minimal viable test* of a combination of a specific data-type, payload and SQL operation across the various encryption schemes. A viable test exercises the system as a user would and must not cause unexpected side effects. Unlike aggregated benchmarking, this approach restricts evaluation to the system under test and allows the performance of each operation mode to be explored in isolation. The study presents an micro-benchmarking approach to PPE system evaluation where each unique viable combination of datatype, operation and encryption mode are exercised. Micro-Benchmarking is a mature technique that has been applied in other performance evaluations, e.g.[3], [4], [5].

Section II of this paper gives an introduction to the system under test (*CryptDB*) and Section II-A reviews the existing evaluations in the literature. Section III presents the technique used with the results in Section IV. Sections V and VI present our conclusions and future research opportunities. From a terminology perspective, it should be noted that the phrase "Primary Key" is used in this paper to refer to the database column that uniquely identifies a row in a table; it should not be confused with a cryptographic key.

II. *CryptDB*

CryptDB [6] is a research database proxy that encrypts data before it is stored in an unmodified MySQL or PostGres database. The threat model for *CryptDB* is the Honest-but-Curious [7] database administrator. In this threat model the database administrator can view the data in the database but not interfere with the data contents or structures. Figure 1 shows a typical usage and the experimental setup for this paper. *CryptDB* can use multiple/different encryption schemes

for each database column to allow a range of SQL operations on encrypted data. Application designers can specify the scheme(s) or allow *CryptDB* to determine the appropriate scheme via a training mode. If multiple schemes are used for a single column the number of encrypted columns is increased.

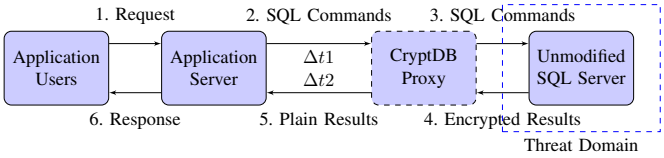

Fig. 1. CryptDB usage. CryptDB is implemented as a SQL proxy for a unmodified SQL database. CryptDB addresses the threat of an adversary with access to the SQL server. In this paper micro-benchmarks are executed with and without the CryptDB proxy. Timings were measured for $\Delta t1$ and $\Delta t2$.

The encryption schemes used by CryptDB are:

- Random (`RND`) – Cipher-text Indistinguishable (IND-CPA); equal plaintexts map to different cipher values.
- Deterministic (`DET`) – Always generates same Cipher-text for same plaintext; i.e allows equality comparisons implemented using AES after [8].
- Order-preserving Encryption (`OPE`) – allows encrypted values to be compared/ordered after [9].
- Homomorphic Encryption (`HOM`) – Partial Homomorphic Encryption that supports addition after [10].

Additional schemes for joins `JOIN`, `OPE-JOIN` and a `SEARCH` scheme (after [11]) were not evaluated as the optimised version of CryptDB used in [6] is unavailable and the published version does not support these schemes.

These schemes can be added, changed and re-encrypted automatically so that additional SQL functions can be performed; although this mode of operation is not desirable for production scenarios as it reduces security while incurring an additional-overhead. An example of a steady-state (post-training) configuration is illustrated in Table I, the encrypted data is stored in the table structure shown in the right column. From inspection of this example it is apparent that a single plaintext (unencrypted) 4 byte integer may consume up to 270 bytes of storage when encrypted with three encryption schemes. When the application inserts or updates a value in the encrypted column the three encrypted values must be recalculated.

Table I
Comparison of plain and encrypted table structures with multiple encryption schemes in use. The 'Eq' suffix on the column denotes that the column is used for equality operations; 'Order' for operations that require comparison operations (e.g `RANGE`) and 'Add' for homomorphic additions. The cdb_salt column stores a salt (i.e a random vector used as an initialization input to strength the encryption).

	Plain Table Structure		Encrypted Table Structure	
Table Name	t1		table_VAOQVRZWMW	
Column Name, Type	intcol1	int	QEARUHZBWBoEq	bigint
			ZRBNJYGKWUoOrder	bigint
			HPTOPMPZZJoADD	varbinary(256)
			cdb_saltTXHARYIMSW	bigint

A. Prior CryptDB Evaluations

The developers of *CryptDB* presented an evaluation of the system in their paper outlining the system [6]. Results for the standardised TPC-C benchmark query mix indicating an overall throughput reduction with *CryptDB* of 21–26% and a latency increase of up to 11 ms depending on configuration. This measured the throughput for the underlying MySQL database with a MySQL Proxy deployed, compared to a *CryptDB* proxy and MySQL. The use of MySQL Proxy is not a generally required component for MySQL and an increase in latency of 600% is seen when a *CryptDB* proxy system is compared with a MySQL system without such a proxy. (This configuration is used in this paper, Section IV-A explores the overhead of this architecture). With the TPC-C benchmark there was a storage factor penalty of 3.76 i.e. the encrypted database consumes 3.76 times the storage space compared to a standard plaintext MySQL database.

Benchmarks [6] of multi-user web applications showed an addition of 7–18 ms (6–20%) per HTTP request and throughput reduction of 14.5%. with a storage factor penalty of 1.2. In this test less than 5% of the total columns in the applications tested were encrypted; while the performance workload includes queries that do not require the encryption/decryption of data.

A separate study [12] presents storage factors of 2 and 2.85 in two differing scenarios. The variance in reported storage factors in the above research is due to the different number of encrypted columns and encryption schemes in effect during the benchmarks; `HOM` encrypted columns in particular consumes 64 times the storage of the plaintext.

III. Micro-Benchmarking Approach

Following the *minimal viable test* criteria a series of tests were developed and executed against an encrypted database and a plaintext control database covering a range of encryption schemes. Each micro-benchmark test is run sequentially on the Control 1 database and then the encrypted database. Each consists consists of: creation, initialisation and workload scripts. The creation script creates the database table, primary key (for indexed retrieval) and payload columns appropriate to the benchmark, the initialisation script is used to set the encryption state of the key or payload (i.e. the schemes and columns used) and the workload script runs the test.

For each data layout and encryption scenario the following operations are evaluated: `INSERT`, `SELECT EQUAL`: Select the entire row by the key column; `SELECT RANGE`: Select a number of rows based on the integer key within the specified range; `SELECT SUM`: Select a sum of integer columns; `SELECT ALL`: Select all rows from all columns. Additional possible benchmarks are discussed in section VI.

The benchmarks are arranged in four test suites, summarised in Table II. They first test if a database operation is supported by the encryption schemes in use and then evaluate the performance and storage characteristics. To complement existing research this paper only presents results that combine multiple schemes.

The workload script was repeatedly executed (against both encrypted and plaintext databases) and timed using the MySQLSlap tool [13] and the total storage requirement was also recorded. The `INSERT` and `SELECT ALL` results were combined to form a *Read/Write Ratio*. Re-encryption is computationally expensive and would significantly distort benchmark results and changing encryption schemes would invalidate the benchmark. Consequently, for valid benchmarking purposes, we forced the encryption of columns to a particular state and verified that the columns remained in this state after the workload was executed (benchmarks that cause side effects are excluded). The evaluations use two suites of controls:

- Control Suite 1: This test set is run against the plaintext MySQL control database for all test conditions. The results for each suite are individually indexed to these results, i.e. each micro-benchmark is the corresponding control result.
- Control Suite 2: This test set is run against the *CryptDB* proxy without any encryption enabled. This captures the intrinsic performance overhead of the *CryptDB* proxy implementation, equipment configuration and architecture without any cryptographic operations executing. Control Suite 2 is used by [6] as an index value and is included here as a basis of comparison with these prior evaluations.

Table II
Summary of the micro-benchmark Suites used to evaluate *CryptDB*. In Suites 1 and 2 the encrypted payload (an Integer and a Text Field respectively) is retrieved using a primary key that is not encrypted. This measures the simple encryption and decryption overhead of the various schemes without exercising any operations on encrypted data. Suites 3 and 4 use the encryption field as the primary key and carry no additional payload but perform operations directly on the encrypted data. Benchmarks that were are non-functional or that result in side effects are excluded from this table.

Suite	Key	Payload	Encryption Variations	SQL Operations
Controls	-	-	None	5
1.	Integer	**E. Text**	4 - DET or RND; OPE or RND	5
2.	Integer	**E. Integer**	4 - DET or RND; OPE or RND; HOM	4
3.	**E. Integer**	None	4 -DET or RND; OPE or RND; HOM	5
4.	**E. Text**	None	4 -DET or RND; OPE or RND	4

A. Security Categorisation

This paper uses three security categorisations in the context of the passive *Honest-but-Curious* administrator Threat Model; *unsecure*, *partially secure* and *secure*. Of the operations exercised here `RND` and `HOM` are categorised as *secure* as they provide (distinct-) semantic security [14]. `OPE` is categorised as *partially secure* as it has been shown by [15] and others to leak information under certain conditions. Deterministic (`DET`) encryption will also leak information if the fields encrypted are non-unique (without a `UNIQUE` constraint); in these executions when `DET` is used as a primary key, we treat it as *secure*, while categorising it as *partially secure* when used to encrypt a payload field.

B. Evaluation Environment

The latest publicly available version of *CryptDB* [16] was used for all benchmarks. This code includes notes that it is not code used in [6] as discussed above. Two Amazon's EC2 Ubuntu 14.04.1 instances were deployed. The client instance ran the MySQLSlap tool to execute the workloads and the *CryptDB* proxy which encrypts the data. The server instance ran an unmodified MySQL 5.5.44. The Client and *CryptDB* proxy instance was a m4.10xlarge (40 vCPU 160 GB Ram) and the MySQL database ran on a m4.4xlarge instance (16 vCPU 64GB Ram). These environments were deliberately oversized for micro-benchmarks to avoid resource starvation, As this is a virtual execution environment there was a risk that other virtual resources may consume capacity and distort results. To minimise this risk all outliers with a z-score greater than 3 were discarded and the benchmarks re-run.

IV. Results

After execution a small number outliers were removed (less than 0.5%).

The detailed results in Table III show the wide ranges of performance and storage factors recorded across the different operations and encryption options. From high level inspection of the results it can be observed that the storage factor penalty is fixed based on the encryption options chosen while the performance varies with both the encryption options and the SQL operation. These variances are discussed in IV-B and IV-C and illustrated in Figures 2–5. It should be noted when reviewing these figures that the SQL operations should not be compared to each other as the results in each sub-figure are indexed back to the plaintext/unencrypted result.

Generally `INSERT` operations show the lowest performance degradation when using encryption, with the bulk of the additional latency being explained by the presence of the proxy rather than the performance of the encryption.

As expected the storage factor penalty is heavily influenced by the presence of a `HOM` encrypted field which requires 64 times the storage. The variance in the read/write ratio indicates that care should be exercised when applying assumptions about the relative cost of these operations. Decisions on how long to cache reads and writes can be influenced by this ratio. The storage factor of 3.76 measured in earlier Table III also includes three results reproduced from prior work on evaluating *CryptDB*. It should be noted that these results are not directly comparable with the micro-benchmarks experimental results reported here due to differences in the evaluation approach and software versions available.

A. Controls

The storage factor penalty and performance figures are indexed to Control 1 (standalone MySQL without *CryptDB*). Control 2 introduces the *CryptDB* Proxy but encrypts no fields. The factor of 5 performance reduction observed between Control 1 and 2 is inline with the latency increase (from 0.10 ms to 0.60 ms) reported in similar experiments in [6, Figure 12]. The increase in the Read/Write ratio between Control 1 and 2 suggests that the proxy adds more overhead while returning results. This is intuitively expected as even just parsing (with out processing) the larger result set will be

Table III
Summary of Benchmark suites 1–4 presenting an overview of relative benchmark execution time on encrypted database on suites with varying encryption layouts. *Benchmark*: The Benchmark Suite, or Control name. *Encryption*: The Encryption in place for the security categorisation; Unsecure, Partially Secure and Secure *Encryption*. The *Storage Factor* and *Performance* figures are indexed to Control 1 for each benchmark individually. *Read/Write Ratio* figure is the ratio of performance of the SELECT EQUAL to INSERT operations for that benchmark suite (if available). *Operations* shows the available SQL operations successfully measured for that benchmark/onion combination; absence of an operation implies an operation is not supported by the encryption setup (or that the operation caused CryptDB to reduce the security to facilitate it). The figures presented for prior work were can't be directly compared due to differences of the underlying experimental setups and method.

Benchmark	Encryption	Storage Factor	Performance	Read/Write Ratio	Operations
Control 1: Benchmark Suites 1 - 4 MySQL	(Unsecure)	1.00 (Index)	1.00 (Index)	0.26	ALL
Control 2 - PK: Plaintext Int, No Payload	(Unsecure)	1.00	5.18 - 14.42	0.43	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE, SELECT SUM
1 - PK: Plaintext Int, Payload: 1 Encrypted Text	DET,OPE (P. Secure)	1.49	8.19 - 13.06	0.22	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE
1 - PK: Plaintext Int, Payload: 1 Encrypted Text	DET,RND (P. Secure)	1.49	8.4 - 13.05	0.23	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE
1 - PK: Plaintext Int, Payload: 1 Encrypted Text	RND,OPE (P. Secure)	1.49	8.49 - 14.8	0.23	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE
1 - PK: Plaintext Int, Payload: 1 Encrypted Text	RND,RND (Secure)	1.49	5.44 - 14.82	0.19	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE
2 - PK: Plaintext Int, Payload: 1 Encrypted Int	DET,OPE,HOM (P. Secure)	40.43	11.12 - 17.27	0.18	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE, SELECT SUM
2 - PK: Plaintext Int, Payload: 1 Encrypted Int	DET,RND,HOM (P. Secure)	40.43	11.15 - 17.63	0.19	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE, SELECT SUM
2 - PK: Plaintext Int, Payload: 1 Encrypted Int	RND,OPE,HOM (P. Secure)	40.43	10.84 - 18.95	0.19	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE, SELECT SUM
2 - PK: Plaintext Int, Payload: 1 Encrypted Int	RND,RND,HOM (Secure)	40.43	9.07 - 19.07	0.21	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE, SELECT SUM
3 - PK: Encrypted Int, No Payload	DET,OPE,HOM (P. Secure)	70.00	10.34 - 18.26	0.28	INSERT, SELECT ALL, SELECT EQUAL, SELECT RANGE, SELECT SUM
3 - PK: Encrypted Int, No Payload	DET,RND,HOM (Secure)	70.00	10.41 - 16.13	0.28	INSERT, SELECT ALL, SELECT EQUAL, SELECT SUM
3 - PK: Encrypted Int, No Payload	RND,OPE,HOM (P. Secure)	70.00	12.4 - 19.0	0.29	INSERT, SELECT ALL, SELECT RANGE, SELECT SUM
3 - PK: Encrypted Int, No Payload	RND,RND,HOM (Secure)	70.00	9.05 - 17.57	0.33	INSERT, SELECT ALL, SELECT SUM
4 - PK: Encrypted Char, No Payload	DET,OPE (P. Secure)	1.50	8.12 - 12.12	0.35	INSERT, SELECT ALL, SELECT EQUAL
4 - PK: Encrypted Char, No Payload	DET,RND (Secure)	1.50	8.19 - 12.35	0.36	INSERT, SELECT ALL, SELECT EQUAL
4 - PK: Encrypted Char, No Payload	RND,OPE (P. Secure)	1.50	8.31 - 14.04	0.36	INSERT, SELECT ALL
4 - PK: Encrypted Char, No Payload	RND,RND (Secure)	1.50	5.53 - 14.35	0.46	INSERT, SELECT ALL
Previous work: Multi-User Web Application from [Popa et al.,2011]	23/563 Encrypted RND,DET,OPE	1.2	1.06 - 1.2	Unavailable	ALL +
Previous work: TPC-C from [Popa et al.,2011] fig. 12	92/92 Encrypted RND,DET,OPE,HOM	3.76	3 - 9	Unavailable	HTTP End to End Measurement
Previous work: Scenario a from [Skiba et al, 2015]	Unavailable	2.85	40	Unavailable	ALL +
Previous work: Scenario b from [Skiba et al, 2015]	Unavailable	2.85	69	Unavailable	ALL +

more expensive than parsing the relatively small SQL queries. As expected the storage factor penalty is unchanged as there are no encrypted fields in these Controls Suites.

B. Suites 1 &2 (Plaintext Primary Key, Encrypted Payload)

The relative execution time results for Suites 1 and 2 are presented in Figures 2 and 3 respectively. These suites contain a single encrypted field as a payload and a plaintext Primary Key. As expected these test suites supported execution of all tested operations as the Primary Key was not encrypted and the encrypted payload is just stored and retrieved by the database transparently.

Encrypted `INSERT` operations are slower than plaintext `INSERT` operations although comparisons of these figures to Control Suite 2 in Table III shows that the actual encryption overhead is modest compared to the overhead incurred by the presence of the proxy.

C. Suites 3 & 4 (Encrypted Primary Key, no Payload)

In contrast to the first two test suites, suites 3 and 4 performed SQL operations on the encrypted payloads rather than merely retrieving them. These benchmarks exercise the usage of Property Preserving Encryption; i.e allowing operations on encrypted data. Figure 4 (Suite 3) highlights that some operations are not supported by certain configurations, i.e. the `SELECT RANGE` operation is not available when `OPE` is not used and `SELECT EQUAL` requires that `DET` or `OPE` be used. The `SELECT ALL` operation is faster when `DET` *onion layer* is used.

Figure 5 presents the results for test suite 4, where the Primary Key was encrypted text. As with suite 3, the performance advantage of `SELECT ALL` operations with DET encryption can be seen.

V. Conclusions

This empirical data presented across the performance, security and functionality dimensions are broadly in line with

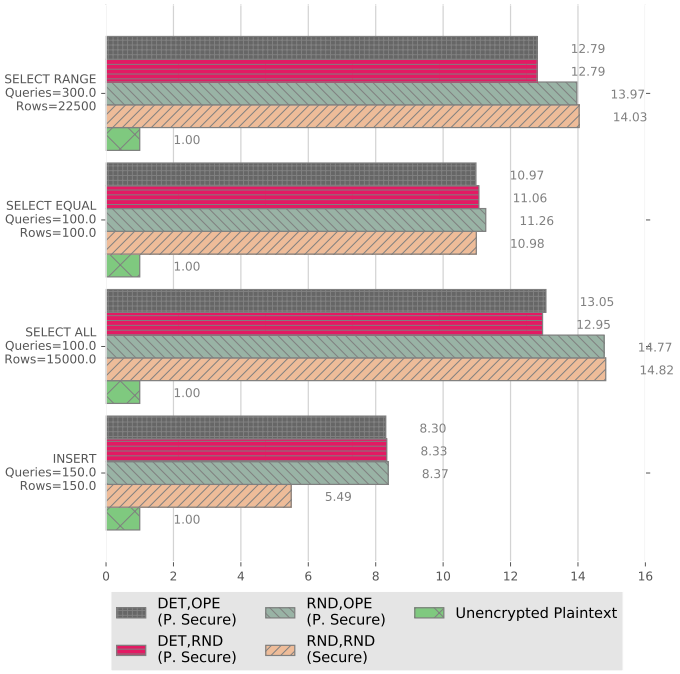

Fig. 2. Relative execution time of Benchmark Suite 1, Plaintext Primary Key with a Single Char Payload, indexed to Plaintext MySQL- Control 1(=1) .

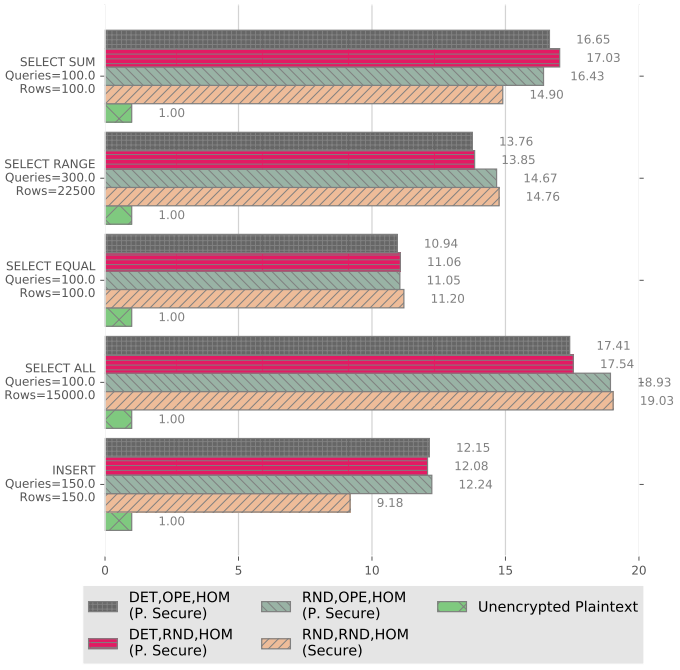

Fig. 3. Relative execution time of Benchmark Suite 2, Plaintext Primary Key with a Single Int Payload

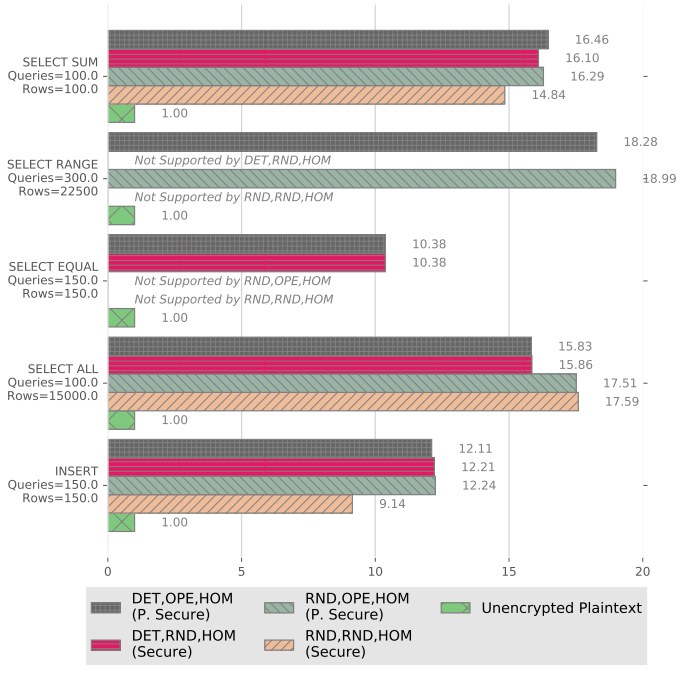

Fig. 4. Relative execution time of Benchmark Suite 3: Encrypted Integer Primary Key with no Payload. N.B Missing bars for an operation indicate that an operation is not supported by the encryption mode.

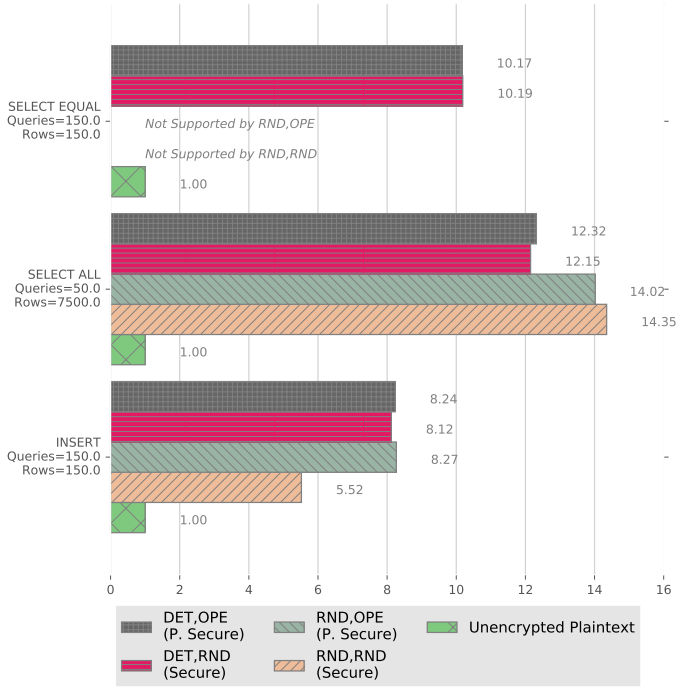

Fig. 5. Relative execution time of Benchmark Suite 4: Encrypted Text Primary Key with no Payload

previous assessments. We supplement prior studies by providing additional detail, clarity and guidance.

This paper reveals nuances relating to the performance, security and functionality of the PPE schemes used in *CryptDB* and validates the utility of micro-benchmark based assessment approach. The results have illustrated that the partial-secure PPE schemes which facilitate operations (`DET`, `OPE`, `HOM`) have a non-trivial storage and performance penalty compared to the secure `RND` scheme. All encryption tested here showed a significant overhead due to the system architecture. The results in section IV-C showed that combining multiple PPE schemes

facilitated the broadest range of operations though at a cost for some of those operations. As multiple encryption schemes are present and the transaction is atomic, the execution time for `INSERT` operations is bounded by the slowest encryption scheme. This suggests a potential optimisation strategy could be applied in certain applications by weakening the transactions atomic guarantees and adopting an *eventually-consistent* model where multiple encrypted columns are used. In all the `SELECT` cases the optimal execution strategy would be to use the column that can be decrypted fastest. From inspection of the suites `DET` and `RND` generally provides the fastest encryption and decryption.

A. Considerations for Application Designers

The data presented here supports the previously published experimental results [6] that illustrated that it is possible to design a secure database system with acceptable performance if close attention is paid to the operations required (and hence appropriate encryption schemes). These experiments have also highlighted that a *CryptDB* style solution may not suit certain workloads, e.g. an OLAP type applications with significant portion of computation performed in the database will incur a greater performance penalty than an OLTP application (which is not typically dominated by database performance) will suffer less degradation.

Two general principles can be applied here; don't encrypt fields unless needed (based on the relevant threat model for the application) and carefully evaluate what operations *must* be performed by the database. For example `HOM` addition has a large storage overhead; it may be more appropriate (when using queries that return small subsets) to use `RND` and do the addition in the application. These principles have an implication for applications with evolving requirements; encryption decision designs may have to be reassessed as the needs of the application change over time.

A significant optimisation for application designers would be to bypass the *CryptDB* Proxy for queries/tables that do not contain encrypted fields. This would involve some application changes and require the application to be aware which fields are encrypted.

VI. Future Work

This work has presented a micro-benchmark approach that could be be applied to other systems such as the work of SAP [17]; Google's Encrypted BigQuery [18] or ARX [19] (a new encrypted database designed by the CryptDB authors).

While using multiple property preserving encryption schemes simultaneously allows additional operations to be executed on encrypted data, further research is needed to establish whether using multiple PPE schemes on the same data reduces the overall security of the system.

References

[1] O. Pandey and Y. Rouselakis, "Property Preserving Symmetric Encryption," in *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*. Springer-Verlag, 2012, pp. 375–391. [Online]. Available: https://www.iacr.org/archive/eurocrypt2012/72370369/72370369.pdf

[2] M. Poess and C. Floyd, "New TPC benchmarks for decision support and web commerce," *ACM Sigmod Record*, vol. 29, no. 4, pp. 64–71, 2000.

[3] R. H. Saavedra-Barrera, "CPU performance evaluation and execution time prediction using narrow spectrum benchmarking," Ph.D. dissertation, University of California, Berkeley, 1992.

[4] I. Manolescu and P. Michiels, "Towards micro-benchmarking XQuery," *ExpDB*, 2006. [Online]. Available: http://win.ua.ac.be/ adrem/bibrem/pubs/manolescu-expdb.pdf

[5] C. P. Kruger and G. P. Hancke, "Benchmarking Internet of things devices," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 7 2014, pp. 611–616. [Online]. Available: http://ieeexplore.ieee.org/document/6945583/

[6] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 85–100. [Online]. Available: https://people.csail.mit.edu/nickolai/papers/popa-cryptdb.pdf

[7] O. Goldreich, *Foundations of cryptography, Vol 2*. Cambridge University Press, 2003.

[8] S. Halevi and P. Rogaway, "A tweakable enciphering mode," in *Annual International Cryptology Conference*, 2003, pp. 482–499.

[9] A. Boldyreva, N. Chenette, Y. Lee, and A. Oneill, "Order-preserving symmetric encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2009, pp. 224–241.

[10] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 1999, pp. 223–238.

[11] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, 2000, pp. 44–55.

[12] M. Skiba, M. S. C. Mainka, D.-I. V. Mladenov, and J. Schwenk, "Bachelor Thesis Analysis of Encrypted Databases with CryptDB," 2015.

[13] Oracle, "mysqlslap - Load Emulation Client," 2016. [Online]. Available: http://dev.mysql.com/doc/refman/5.7/en/mysqlslap.html

[14] R. A. Popa, N. Zeldovich, and H. Balakrishnan, "Guidelines for Using the CryptDB System Securely," *IACR Cryptology ePrint Archive*, vol. 2015, p. 979, 2015.

[15] V. Kolesnikov and A. Shikfa, "On The Limits of Privacy Provided by Order-Preserving Encryption," *Bell Labs Technical Journal*, vol. 17, no. 3, pp. 135–146, 2012.

[16] R. A. Popa, "CryptDB - GitHub Repository," 2013. [Online]. Available: https://github.com/CryptDB/cryptdb

[17] P. Grofig, M. Haerterich, I. Hang, F. Kerschbaum, M. Kohler, A. Schaad, A. Schroepfer, and W. Tighzert, "Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data," in *Sicherheit*, 2014, pp. 115–125.

[18] T. Schindler and C. Skornia, "Secure Parallel Processing of Big Data Using Order-Preserving Encryption on Google BigQuery," *arXiv preprint arXiv:1608.07981*, 2016.

[19] R. Poddar, T. B. Raluca, and A. Popa, "Arx: A Strongly Encrypted Database System," *IACR Cryptology ePrint Archive 2016*, 2016. [Online]. Available: https://eprint.iacr.org/2016/591.pdf