Searchable Encryption to Reduce Encryption Degradation in Adjustably Encrypted Databases

Florian Kerschbaum[1] and Martin Härterich[2]

[1] University of Waterloo, Canada
florian.kerschbaum@uwaterloo.ca
[2] SAP
Karlsruhe, Germany
martin.haerterich@sap.com

Abstract. Processing queries on encrypted data protects sensitive data stored in cloud databases. CryptDB has introduced the approach of adjustable encryption for such processing. A database column is adjusted to the necessary level of encryption, e.g. order-preserving, for the set of executed queries, but never reversed. This has the drawback that long running cloud databases will eventually transform into only order-preserving encrypted databases. In this paper we propose searchable encryption as an alternative in order to reduce this encryption degradation. It maintains security while only marginally impacting performance when applied only to infrequently used queries for searching. We present a budget-based encryption selection algorithm as part of query planning for making the appropriate choice between searchable and deterministic or order-preserving encryption. We evaluate our algorithm on a long-tail distributed TPC-C benchmark on an experimental implementation of encrypted queries in an in-memory database. In one choice of parameters our algorithm incurs only a 1.5% performance penalty, but one of 15 columns is not decrypted to order-preserving or deterministic encryption. Our selection algorithm is configurable, such that higher security gains are possible at the cost of performance.

1 Introduction

In order to protect cloud databases data can be processed in encrypted form [1, 9, 10, 29, 30]. A common way to enable processing of encrypted data is order-preserving encryption [1, 2, 19, 23, 28]. Order-preserving encryption allows processing many SQL queries without modification. However, order-preserving encryption is susceptible to simple attacks on the static data [27].

In order to increase security CryptDB has introduced adjustable encryption [29]. The idea is to layer encryption in onions. Queries are analyzed and the encryption layer is adjusted before their execution.

This has the positive effect that only the layers necessary for the query execution, e.g. deterministic encryption instead of order-preserving encryption, are revealed and thus security is increased. A database starts in a completely secure

(cold) mode and transforms into a (hot) mode which is very efficient, since no more decryption operations are necessary, but all queries can be processed on the data as is. This transformation is also never reversed. Since it is not possible to determine when a cloud database has been compromised, there is no reason to encrypt data which has once been revealed to the cloud service provider.

This lack of reversion has the negative consequence that many databases may ultimately reach a state that has only order-preserving encrypted columns. Hence, in a long running database system adjustable encryption may be no better than pure order-preserving encryption. The set of all queries determines the encryption level, even if those queries contribute little to the overall load of the database. Particularly, the long tail of the query distribution may have a severe negative security effect. These queries are infrequently executed, e.g. only once, presumably using columns that are infrequently used for searching, but have the same impact on the security as the most frequently reoccurring ones. This paper proposes dealing with these infrequent queries differently in order to confine their impact.

This raises two research questions: First, how to detect infrequently used columns and second, how to handle them. For the second problem we propose to use searchable encryption [3, 5, 11, 12]. Searchable encryption is a randomized, strongly secure encryption scheme where the key holder can issue tokens for equality or range searches. The search algorithm is different than in a regular table scan and also significantly slower. Yet, all data for which no token has been issued remains semantically secure. Hence, searchable encryption is particularly suited for infrequently searched columns, since the search pattern is sparse.

For the first problem we propose a more intelligent encryption selection algorithm. It now has two choices: searchable encryption and order-preserving or deterministic encryption. It will first try searchable encryption until a certain threshold has been reached and only then decrypt. This increases the time for transforming from a cold to a hot database, but handles infrequently used columns with searchable encryption.

We perform a set of experiments using the TPC-C benchmark on our algorithms in an implementation of encrypted queries in an in-memory, column-store database. Our algorithm is configurable in order to allow different trade-offs between security and performance according to the preference of the database administrator. However, we were particularly interested in a set of parameters where the gain of security is clearly higher than the cost of security. In one particular choice of parameters our algorithm may incur only a small 1.5% performance penalty, but the most infrequently used column is not decrypted. Note that the economic value of a single non-decrypted column can be very high for sensitive data, such as salaries, outstanding sales prices or health care data. Due to the difficulty of scientifically assessing sensitivity values, we only report the percentage of still randomly encrypted data; in our case a 6.7% security improvement which is still more than 4 times the performance penalty. We also show in detail the different trade-offs between security and performance in query planning for different parameters of the algorithm (Section 5).

2 Related Work

2.1 Queries on Encrypted Data

Hacigümüs et al. introduce the first database for processing SQL queries on encrypted data [10]. They use deterministic encryption and binning for range queries. Binning requires the client to post-process the result and filter non-matching entries. Agrawal et al. improve on this by order-preserving encryption [1]. In order-preserving encryption plaintexts are mapped to ciphertext in the same order. This removes the necessity for post-processing (and query rewriting) and range queries can be processed with the same relational operator as on plaintexts. Later, Popa et al. extend this concept using adjustable encryption which adjusts the ciphertext to the query [29]. Hang et al. also show how to implement this over multiple keys [13].

Boldyreva et al. formalize order-preserving encryption [2]. They provide a proof that their scheme is the best possible stateless encryption scheme [2]. Recently Naveed et al. have shown that this security definition is rather weak and simple attacks can exploit the static leakage of order-preserving encryption [27]. A stronger notions of security – indistinguishability under (frequency-analysing) order-preserving chosen plaintext attack – are achieved by the scheme by Popa et al. [28] and Kerschbaum [19], respectively. Yet, their schemes are not efficiently compatible with adjustable encryption. Hence, system builders have to make a choice between the two. Our experiments indicate that adjustable encryption has a high security improvement, e.g. in the TPC-C benchmark only 15 columns need to use deterministic encryption. We hence believe that adjustable encryption is preferable to a stronger order-preserving encryption.

We build on adjustable encryption providing a further enhancement of security. Particularly, we introduce a query planning algorithm for searchable encryption in order to handle infrequently used columns in adjustable encryption, such that even not all 15 columns need to be deterministic.

2.2 Searchable Encryption

Searchable encryption offers stronger security than order-preserving or deterministic encryption. Using a token generated by the secret key one can search for values or within ranges. Without the token the ciphertext is as secure as common standard encryption.

The first sub-linear search time, (inverted) index-based searchable encryption scheme was introduced by Curtmola et al. [5]. Its idea is to provide an index of deterministically encrypted keywords and an encrypted list of documents. Since each deterministic ciphertext is unique, these schemes are not as susceptible to frequency analysis, but still more efficient to search. Hahn and Kerschbaum showed how the index can be built from the access pattern [11].

Searchable encryption also supports complex queries. The fastest method has been proposed by Demertzis et al. in [6] where all subranges in the disjunctions are indexed (and leaked on a match).

There exist also a large number of applications which have been developed on top of encrypted, in-memory, column-store database with specific protocols, e.g., benchmarking [4, 15, 16, 20, 22, 26], RFID tracking [18, 21, 25], smart metering [14], supply chain planning [7, 24], web applications [8] or reputation systems [17].

3 Searchable Encryption

3.1 Definitions

We propose to use searchable encryption as an alternative to deterministic and order-preserving encryption in adjustable encryption. Searchable encryption allows the (private) key holder to issue a search token for a query string. Using this search token the ciphertext holder can compare a ciphertext to the query string. The result of this comparison (match / no match) is immediately revealed in plaintext.

We employ the symmetric key variant due to better performance and the lack of need for a public key in our scenario. A searchable encryption in our scenario consists of the following algorithms:

- $sk \leftarrow KeyGen(\lambda)$: Generates a secret key sk for a security parameter λ.
- $c \leftarrow Enc(sk, x)$: Encrypts a plaintext x into a ciphertext c using secret key sk.
- $t \leftarrow TrapDoor(sk, x)$: Generates a trapdoor search token t for plaintext x using secret key sk.
- $\top/\bot \leftarrow Test(t, c)$: Returns $\top$ if the search token matches and $\bot$ if not.

Note that the ability to decrypt is optional in our scenario and hence not implemented, since we can use another encrypted column of the same data for decryption.

3.2 Performance Calibration

We compare the execution time of SQL queries on searchable encryption to that of on deterministic encryption or order preserving encryption. These results are used to calibrate our query planning algorithm. This algorithm compares the runtime of a query using searchable encryption with its equivalent using deterministic or order-preserving encryption, respectively. Both executions – on searchable and on deterministic or order-preserving encryption – share some common effort which includes query pre-processing and decryption of the result set on the client and data transfer between the client and the server. We omit this time in the following evaluation, because it does not contribute to the advantage of one execution strategy over the other.

For searchable encryption we must consider the generation of the trapdoor and the runtime of the UDF. Our tests show that the UDF scales very well, so that we have the following linear cost model. Let N denote the number of

entries in the searched database column, $t_{\text{UDF scan}}$ denote the scan time per database row and $t_{\text{trapdoor generation}}$ includes both the actual execution of the cryptographic algorithm and the row-independent execution time for query processing.

$$t_{\text{searchable}} = Nt_{\text{UDF scan}} + t_{\text{trapdoor generation}}$$

When executing equality or range searches on deterministic or order-preserving encryption, respectively, we use unmodified relational operators that compare the values stored in the database to the search values. In this case the main execution time stems from encrypting these literal values to deterministic or order-preserving ciphertexts. Hence in our linear model

$$t_{\text{deterministic}} = Nt_{\text{scan}} + t_{\text{encryption}}$$

the slope t_{scan} is very small. Note that in particular for in-memory databases we find that $t_{\text{scan}} \ll t_{\text{UDF scan}}$. In fact, for table sizes up to 100 million entries there is a total runtime of less than 50ms.

Our measurements lead to the following values for $t_{\text{UDF scan}}$, $t_{\text{trapdoor generation}}$, t_{scan} and $t_{\text{encryption}}$ which we use for the calibration of our query planning algorithm.

$t_{\text{UDF scan}}$	4.5 μs
$t_{\text{trapdoor generation}}$	65 ms
t_{scan}	$\sim$ 0 μs ($<$ 1 ns)
$t_{\text{encryption}}$	20 ms

Table 1. Constants used for calibration

4 Detecting and Handling Infrequently Used Columns

Searchable encryption in our UDF can handle selection similar to deterministic or order-preserving encryption but at higher security and lower performance. We now aim to identify infrequently used columns, such that we can decide to handle them by searchable encryption keeping the performance impact low, but maximizing the relative security gain.

4.1 Problem

Consider an adjustably encrypted database and the following two sequences A and B of queries:

Sequence A:

`SELECT` x `FROM` T `WHERE` $y > 10$
`SELECT` x `FROM` T `WHERE` $y > 10$

SELECT x FROM T WHERE $y > 10$
SELECT x FROM T WHERE $y > 10$
SELECT x FROM T WHERE $y > 10$

Sequence B:

SELECT x FROM T WHERE $y = 10$
SELECT x FROM T WHERE $y > 10$
SELECT x FROM T WHERE $y = 10$
SELECT x FROM T WHERE $y = 10$
SELECT x FROM T WHERE $y = 10$

Using the standard adjustment algorithm both sequences result in an order-preserving encryption of column y. Yet, if in sequence B the second query is handled using searchable encryption, then the security would remain at deterministic encryption and the performance impact would be small. Our problem is to identify and handle differently this specific (infrequent) query.

The problem is a typical scheduling problem where an optimizer has to make a decision based on future inputs (queries). The decision problem in case of the first query of sequence A and the second query of sequence B is almost identical: A query requiring a database adjustment appears for the first time. The optimizer has to decide whether to use searchable encryption or to decrypt.

The only sensible choice is to treat first-time appearing queries as infrequent until they reach a certain threshold and then decrypt. We present our algorithm in the next section.

4.2 Algorithm

We use a budget mechanism in order to determine infrequently used columns. For each column *col* we maintain a budget $budget[col]$. This budget describes the extra amount of time allowed for searchable encryption compared to deterministic or order-preserving encryption. It can be maintained in an arbitrary but fixed unit of time (say milliseconds). For each column we use searchable encryption until the budget is used up (i.e. reaches 0) and subsequently switch to the other encryption schemes.

We fix two parameters α and β for our algorithm. Whenever a query is executed the parameter α is added to the budget. The parameter β defines the upper bound of the budget, i.e. the budget is never increased beyond β. We call this process budget refilling.

When we choose searchable encryption we deduct the additional cost of the query from the budget. Let σ be the cost of searchable encryption for equality and τ be the cost of searchable encryption for ranges as determined in Section 3.2.

We can run our algorithm – in particular budget refilling – for several different sets of columns. A SQL query uses a number of columns, not only the ones in selection. We consider and later evaluate the following options for the budget update strategy, i.e. choosing the column *col*.

- S_1: Increase the budget for all columns used as selection parameters.

- S_2: Increase the budget for all columns occurring in the query in any role (e.g. also in the result list)
- S_3: Increase the budget for all columns of all tables occurring in the query.
- S_4: Increase the budget for all columns of the database scheme used.

4.3 Cost Estimation

Note that the costs σ and τ used in our algorithm depend on the number of rows to which the test function needs to be applied. For simple scans on complete database tables this number is readily available. However, as soon as there are other selection conditions which narrow the result set it is important that they are applied first. Hence the actual number of rows the function acts on has to be estimated. This is a difficult problem and lies at the heart of many query optimizations already for non-encrypted data. In our implementation we use a straight-forward approach and assume that the selection conditions occurring in our queries are independent and reduce the result set by a fixed factor.

5 Experimental Results

5.1 Security Measure

We define security as the encryption state of the database after one test run, i.e. a series of queries chosen according to the distribution described before. Each column that is decrypted to deterministic encryption lowers the security compared to columns encrypted using randomized and searchable encryption. Our security measure is hence simply the number of columns *not* decrypted to order-preserving or deterministic encryption.

This security measure is independent of the number of queries we have executed. We simply measure the state of the database. Note that without our encryption selection algorithm all 15 columns would be decrypted to deterministic encryption after the first query accessing them, i.e. after each test run. We hypothesize that using our algorithm the database will remain in a more secure state.

We report the percentage of columns not decrypted. The baseline in our set of queries from the TPC-C benchmark is 15 columns that may be decrypted without our encryption scheme selection algorithm. Hence, each not decrypted column is a 6.7% security improvement. We ignore any different sensitivity level of columns, since they are difficult to assess scientifically, but note that any non-decrypted column may already have high economic value.

We devise a simple theoretical test whether a column is likely to be decrypted or not. Given the distribution of the test queries and the strategy used for budget refilling one can calculate the expected value of the change per query execution of the budget for a given data base column *col*. Let Q_i be the query type of query i, p_j be the probability of picking a query type j ($1 \leq j \leq 11$), and $\chi(Q_i, col) = \chi^{\text{refill strategy}}(Q_i, col)$ be 1 if query type Q_i leads to a refill for

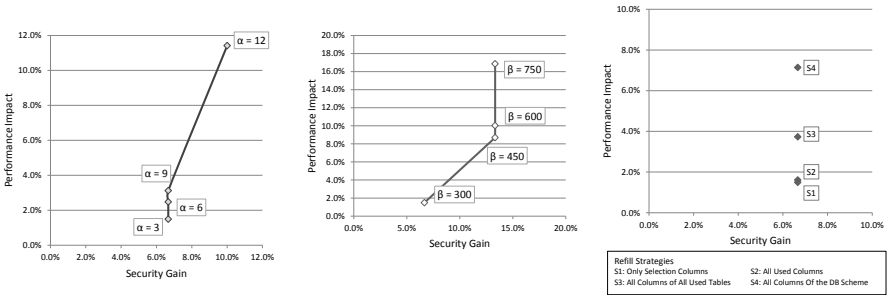

Fig. 1. (left) Security vs. Performance for $\alpha = 3, 6, 9, 12$, $\beta = 100\alpha$ and strategy S_1. (middle) Security vs. Performance for $\alpha = 3$, $\beta = 300, 450, 600, 750$ and strategy S_1. (right) Security vs. Performance for $\alpha = 3$, $\beta = 300$ and all strategies.

column *col* and 0 else. The cost $\sigma(Q_i, col)$ (cf. 4.3) of query i depends on the query type and the column. Then, we have

$$E(\Delta budget[col]) = \sum_i p_{Q_i} \left(\chi(Q_i, col)\alpha - \sigma(Q_i, col)\right) \tag{1}$$

If the expected value $E(\Delta budget[col])$ is negative, then the budget will eventually reach 0. If, however, the expected value is positive then there is a good chance that no decryption of the column *col* to a weaker security level is ever necessary.

5.2 Performance Measure

We measure the wall clock time the database requires for performing the queries on encrypted data. Let t_i be time for the i-th query. We use the sum $s = \sum_i t_i$ of all queries as the measured performance.

In an encrypted "hot" database without our encryption selection algorithm the query would always be performed using deterministic or order-preserving encryption. Our encryption scheme selection algorithm improves security, but infrequent queries may be slower. In order to measure this performance penalty we first measured a baseline. We executed each query type of our set from the TPC-C benchmark 500 times on deterministic encryption. We use the median b_j as the baseline performance for this query type ($1 \leq j \leq 11$).

Using the randomly chosen queries in a test run we compute a baseline for the entire test run. Let Q_i be the query type of query i ($1 \leq Q_i \leq 11$). Then, our baseline B is $B = \sum_i b_{Q_i}$. We report the performance penalty of our encryption scheme selection algorithm as $\frac{s}{B} - 1$. A performance penalty of 5% means that an encrypted database using our algorithm (for the chosen set of parameters) executes 5% slower than encrypted database without our algorithm. Recall that our algorithm improves security, i.e. we trade performance for security.

Each test run consists of 500 queries ($0 < i \leq 500$) chosen according to the geometric distribution described before. In order to have the database reach a "hot"

state in our experiments, we disregard the first 100 queries in our performance measurement. Even when using our encryption scheme selection algorithm, several columns need to be decrypted to deterministic encryption. Including this time will skew our measurements, since it is not included in the baseline. We argue that the performance of a "hot" database is critical for practical use rather than the cost of reaching this state, since this can be included in an installation or setup phase. Hence, we focus our measurements on the "hot" phase.

5.3 Experimental Setup

We execute all experiments on an SAP HANA database (SP05 release) running on an HP Z820 workstation with 128GB RAM und 16 dual cores (Intel Xeon CPU running at 2.60GHz). There was no network access, connections were performed via the loopback interface. Our performance measurement is solely based on the database execution time and hence independent of network performance. Our security measurement is obviously independent of network performance.

Our client is implemented in Java 1.7 as a JDBC driver and running on the 64-bit JVM. The crypto routines are implemented in C++, compiled with GCC 4.3 and accessed via JNI. The UDFs use the same crypto routines accessible as linked libraries.

We execute series of test runs. Each test run consists of 500 queries chosen according to the geometric distribution described before. A series consists of 20 test runs and we report the median performance penalty and median security improvement as described before of those 20 test runs. Note that the median of an even number of values is the mean of the middle two values. Hence, we sometimes have "half" decrypted columns.

Each series of test runs has fixed parameters for the budget increment α, the budget upper limit β and the budget refilling strategy. The initial value for the budget is chosen to be $\frac{1}{2}\beta$. We conduct experiments varying each parameter individually. In the first experiment we vary α, in the second β and in the third the strategy. For each experiment we report the security improvement vs. performance penalty trade-off for the different parameter choices. We intend to give the database administrator guidance on configuring this trade-off in our algorithm.

5.4 Budget Increment α

We measure the impact of the budget increment α in our algorithm. As mentioned before α is linear in the expected value of the budget of a column and, hence, its probability of decryption. We choose four values for $\alpha = 3, 6, 9, 12$. We choose the budget upper limit $\beta = 100\alpha$. We choose the budget update strategy S_1, i.e. increase the budget for all columns used as selection parameters. We explain our choice of strategy using the results from the appropriate experiments in Section 5.6. Our random choices and experimental setup are as described before. Our results are depicted in Figure 1.

Discussion We can see in Figure 1 that the performance impact of searchable encryption is high, since the slope is steep. This can be expected from our calibration in Section 3.2. Nevertheless, we also see that for the values of $\alpha = 3, 6, 9$ the security gain is higher than the performance penalty whereas for $\alpha = 12$ the performance penalty is higher than the security gain. Particularly, for $\alpha = 3$ and $\beta = 300$ we have performance penalty of 1.5% and one non-decrypted column, i.e. a security gain of 6.7%. Hence, we conclude that for truly infrequent queries searchable encryption is indeed a viable alternative. In our subsequent experiments we use $\alpha = 3$ in order to address this optimal set of infrequent queries. Note that in our test data, smaller values for $\alpha < 3$ make little sense, since there is only one non-decrypted column left.

5.5 Budget Upper Limit β

We measure the impact of the budget upper limit β in our algorithm. It is linear in the expected time to reach decryption (hot state), but it also counters the probabilistic nature of the query distribution. It has to be high enough in order to allow bursts of infrequent queries in an otherwise "stable" query set. We choose the four values for $\beta = 300, 450, 600, 750$. We chose $\alpha = 3$ from the results in the first experiment and the same budget update strategy S_1 as before. Our random choices and experimental setup are as described before. Our results are depicted in Figure 1.

Discussion We can see in Figure 1 again a high performance impact of searchable encryption. Nevertheless, our choice of budget increment $\alpha = 3$ leads to higher security gains than performance penalties for $\beta = 300, 450, 600$. We conclude that $\alpha = 3$ is indeed addressing a promising set of infrequent queries. As expected the higher the budget upper limit β, the less columns are decrypted and the better the security, but the worse performance. Note that due to our budget update strategy S_1 of refilling only selection columns very few column budgets get increased and the budget limit (and initial budget) is more critical. For $\beta = 300$, we have the best security gain to performance penalty ratio. Hence, we use $\alpha = 3$ and $\beta = 300$ in our subsequent experiment for the budget update strategy.

5.6 Budget Update Strategy

We measure the impact of the budget update strategy S_1 to S_4. We expect less columns to be decrypted from S_1 to S_4, since budget increments occur more frequently. This implies more security gain and less performance penalty. We chose $\alpha = 3$ and $\beta = 300$ from the results in the first two experiments. Our random choices and experimental setup are as described before. Our results are depicted in Figure 1.

Discussion We can see in Figure 1 somewhat surprising results. For our choices of $\alpha = 3$ and $\beta = 300$ the security is not impacted by the budget update strategy. All four strategies on average do not decrypt one column. Still, we can see the expected decreased performance penalty.

Different strategies may impact different columns and hence we believe that there may be a qualitative difference between the strategies. Still, even if the security differences are just too small to be measured, the strategy S_1 is clearly superior. Hence, we recommend and used in all our other experiments strategy S_1 of refilling only selection columns.

References

1. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM International Conference on Management of Data*, SIGMOD, 2004.
2. A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *Proceedings of the 28th International Conference on Advances in Cryptology*, EUROCRYPT, 2009.
3. D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Proceedings of the 33rd International Conference on Advances in Cryptology*, CRYPTO, 2013.
4. O. Catrina and F. Kerschbaum. Fostering the uptake of secure multiparty computation in e-commerce. In *Proceedings of the 3rd International Conference on Availability, Reliability and Security*, ARES, 2008.
5. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5), 2011.
6. I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis. Practical private range search revisited. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD, 2016.
7. J. Dreier and F. Kerschbaum. Practical privacy-preserving multiparty linear programming based on problem transformation. In *Proceedings of the 3rd IEEE International Conference on Privacy, Security, Risk and Trust*, PASSAT, 2011.
8. B. Fuhry, W. Tighzert, and F. Kerschbaum. Encrypting analytical web applications. In *Proceedings of the 8th ACM Cloud Computing Security Workshop*, CCSW, 2016.
9. H. Hacigümüs, B. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *Proceedings of the 9th International Conference on Database Systems for Advances Applications*, DASFAA, 2004.
10. H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM International Conference on Management of Data*, SIGMOD, 2002.
11. F. Hahn and F. Kerschbaum. Searchable encryption with secure and efficient updates. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, CCS, 2014.
12. F. Hahn and F. Kerschbaum. Poly-logarithmic range queries on encrypted data with small leakage. In *Proceedings of the 8th ACM Cloud Computing Security Workshop*. CCSW, 2016.

13. I. Hang, F. Kerschbaum, and E. Damiani. Enki: Access control for encrypted query processing. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD, 2015.
14. M. Jawurek, F. Kerschbaum, and G. Danezis. Sok: Privacy technologies for smart grids – a survey of options. Technical Report MSR-TR-2012-119, Microsoft, 2012.
15. F. Kerschbaum. Building a privacy-preserving benchmarking enterprise system. *Enterprise Information Systems*, 2(4), 2008.
16. F. Kerschbaum. Practical privacy-preserving benchmarking. In *Proceedings of the IFIP International Information Security Conference*, SEC, 2008.
17. F. Kerschbaum. A verifiable, centralized, coercion-free reputation system. In *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, WPES, 2009.
18. F. Kerschbaum. An access control model for mobile physical objects. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, SACMAT, 2010.
19. F. Kerschbaum. Frequency-hiding order-preserving encryption. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, CCS, 2015.
20. F. Kerschbaum, D. Dahlmeier, A. Schröpfer, and D. Biswas. On the practical importance of communication complexity for secure multi-party computation protocols. In *Proceedings of the ACM Symposium on Applied Computing*, SAC, 2009.
21. F. Kerschbaum and N. Oertel. Privacy-preserving pattern matching for anomaly detection in rfid anti-counterfeiting. In *Proceedings of the International Workshop on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec, 2010.
22. F. Kerschbaum, T. Schneider, and A. Schröpfer. Automatic protocol selection in secure two-party computations. In *Proceedings of the 12th International Conference on Applied Cryptography and Network Security*, ACNS, 2014.
23. F. Kerschbaum and A. Schröpfer. Optimal average-complexity ideal-security order-preserving encryption. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, CCS, 2014.
24. F. Kerschbaum, A. Schröpfer, A. Zilli, R. Pibernik, O. Catrina, S. de Hoogh, B. Schoenmakers, S. Cimato, and E. Damiani. Secure collaboratiue supply-chain management. *IEEE Computer*, 44(9), 2011.
25. F. Kerschbaum and A. Sorniotti. Rfid-based supply chain partner authentication and key agreement. In *Proceedings of the 2nd ACM Conference on Wireless Network Security*, WISEC, 2009.
26. F. Kerschbaum and O. Terzidis. Filtering for private collaborative benchmarking. In *Proceedings of the International Conference on Emerging Trends in Information and Communication Security*, ETRICS, 2006.
27. M. Naveed, S. Kamara, and C. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, CCS, 2014.
28. R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, S&P, 2013.
29. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, SOSP, 2011.
30. S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. In *Proceedings of the 39th International Conference on Very Large Data Bases*, PVLDB, 2013.