M. Meier, D. Reinhardt, S. Wendzel (Hrsg.): Sicherheit 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 209

Order Preserving Encryption for Wide Column Stores

Tim Waage[1]

Abstract: Order-preserving encryption (OPE) allows encrypting without losing information about the order relation between the encrypted data items. Thus, the execution of compare, order and grouping operations can be done like on plaintext data. In particular it allows databases to do range queries over encrypted data, which is a useful feature especially for cloud databases that usually run in untrusted environments. Several OPE schemes have been proposed in the last years, but almost none of them are used in real world scenarios. While OPE was at least implemented for some SQL-based prototype systems before (e.g. [Po11, Tu13], our work identifies the practical requirements for utilizing OPE in existing NoSQL cloud database technologies. It also provides runtime analyses of two popular OPE schemes combined with two popular NoSQL wide column store databases.

Keywords: Order-preserving encryption, wide column stores, Apache Cassandra, Apache HBase

1 Introduction

Nowadays distributed data storage becomes more and more important due to the increased amount of data being produced every day, by private persons (e.g. in social media platforms) as well as by business or research. Especially modern web services have a high demand for availability, consistency, partition tolerance, performance, scalability and elasticity, that are at best difficult and expensive to achieve with traditional relational databases. NoSQL databases running in distributed cloud environments were made to meet those requirements. Unfortunately security was not a primary concern of their designers [Ok11]. Instead some sort of front end is assumed to take care of authentication, authorization, etc. Yet NoSQL databases, especially the sub-category of wide column stores (WCSs), are a key technology behind many popular platforms, e.g. HBase behind Facebook [Bo11], Cassandra behind eBay or BigTable behind almost every Google service [Ch08].

Encryption is always a handy tool when it comes to outsourcing data to such untrustworthy environments. Unfortunately it also limits the options for interacting with the data that was encrypted. Using only traditional encryption methods like AES or RSA is unfeasible, because such schemes do not preserve the plaintext properties, that database systems are relying on (see section 3). We focus on one of those properties: the order relation. Preserving it during encryption enables a database to perform tasks such as executing range queries like on plaintext data. Although order-preserving encryption (OPE) is an active field of research, the practical feasibility of most schemes is insufficient.

Our work makes the following contributions: (i) it evaluates the practical feasibility of two fundamentally different OPE schemes, namely [BCO11, KS14], in existing database sys-

[1] Georg-August-Universität Göttingen, Institut für Informatik, Goldschmidtstraße 7, 37077 Göttingen, tim.waage@informatik.uni-goettingen.de

tems, (ii) it identifies the special requirements of NoSQL WCSs regarding OPE encryption and (iii) it conducts a practical performance comparison of [BCO11, KS14] in combination with the currently most popular NoSQL WCS platforms [So] Apache Cassandra [LM10] and Apache HBase [Bo11] to assess their respective strength and weaknesses.

2 Feasibility and Security of OPE in Practice

OPE can be described briefly like follows. If D is the plaintext space, R the ciphertext space (both with an order defining relation $\leq$) and k is a secret key, then an OPE scheme is a function $f_k : D \rightarrow R$ for which: $x \leq y \Rightarrow f_k(x) \leq f_k(y)$ for all $x, y \in D$. [Ag04] were the first to define this problem of OPE and proposed a theoretical scheme to address it. However, achieving practical feasibility without sacrificing security is still a challenging task. We evaluate the feasibility of OPE schemes in database scenarios based on three criteria:

Ciphertext (im-)mutability. An OPE scheme is called *mutable*, if it requires its ciphertexts to be changed as more and more input gets encrypted. An example of this category is [KS14]. Its state is a set of ordered plaintext-ciphertext-pairs, initialized with $\{(-1,-1), (maxPlaintextValue, maxCiphertextValue)\}$. A new value is always inserted in the middle of the gap between the next lower and next greater already encrypted value. If these values are directly consecutive, there is no gap here and the state requires a re-balancing. In practice this results in the overhead of reading, re-encrypting and writing back the data to the database. However a practical advantage of mutable OPE schemes is their fast and simple decryption process, as can be observed in section 4. *Immutable* OPE schemes avoid the re-encryption overhead in the first place. Immutable means once a plaintext is encrypted, the corresponding ciphertext is final. An instance of this category is [BCO11]. It is based on the fact that any order-preserving function from $1...M$ to $1...N$ can be represented by a combination of M out of N ordered items. Thus, ciphertexts can be computed by sampling values according to the negative hypergeometric distribution. However, using immutable OPE schemes results in having less security than using a mutable schemes [PLZ13].

Need for additional data structures. OPE schemes producing mutable ciphertexts require additional data structures for storing their state (plaintext-ciphertext-mappings). That can be done using indices, trees, dictionaries etc., either on clientside (or at least a trusted enviroment), e.g. [KS14], or on serverside, e.g. [PLZ13, Ro15]. Note that in particular the maintenance of tree sturctures is very expensive to achieve for most (unmodified) database systems. Hence additional software components on serverside are often proposed for performance reasons, which makes practical implementations rather complex.

Need for additional architectural components. Client applications and database platforms normally do not have built-in mechanisms for (order-preserving) encryption. Thus additional components are required for both rewriting queries to make them work with the serverside data structures (as they might have to be altered for functioning with the OPE schemes) as well as for performing decryption and encryption itself. Usually those components have to reside in the trusted (clientside) environment (e.g. [Po11, Tu13]). However some OPE schemes even require components running co-located to the database server (e.g. [PLZ13]), which can not be considered practical due to the architectural overhead.

Regardless of these practical concerns the first formal security analysis of OPE [Bo09] proved that ideal security[2] with immutable ciphertexts can only be accomplished, if the ciphertext space size $|R|$ is exponential in the plaintext space size $|D|$, which is hard to achieve in practice. Security can still be improved, e.g. by modular plaintext shifting [BCO11] (easy to implement, but only a small security enhancement) or using fake queries to hide the query distribution [Ma15] (causing communication and computation overhead).

In practice ideal security can be achieved easier by OPE schemes producing mutable ciphertexts, because they do not have the requirement of a ciphertext space size being exponential in the plaintext space size. They also hide the frequency distribution of plaintext-ciphertext assignments much better, being able to achieve an almost uniform distribution (as shown e.g. by [Wo13]). Still, that also means dealing with unavoidable re-encryptions of (parts of) the ciphertext, that is already stored in the database. Recent schemes try to keep the number of such updates to a minimum [KS14] or take the burden of reassigning ciphertexts to components on serverside [PLZ13] to reduce at least communication costs.

An alternative approach to avoid re-encryption in the first place is pre-encrypting the whole plaintext space D in advance [Wo13, Li14]. The unfeasibility of such an approach can be illustrated using the following example: let D be defined by a common Integer datatype. Having a typical length of 32 bit, $|D|$ would be of size 2^{32}, which means 4.3 billion items would have to be pre-computed and stored (even if the majority is never used).

3 Wide Column Stores and OPE

Due to general working principles all WCSs share, OPE schemes have to satisfy certain requirements, depending on what kind of data they are supposed to encrypt. These principles can be roughly described as follows. While WCSs use tables, rows and columns like traditional relational (SQL-based) databases, the fundamental difference is that columns are created for each row instead of being predefined by the table structure. Every row has an identifier that is unique for the table (commonly referred to as "row key"). Data is maintained in lexicographic order by that key. As WCSs are distributed systems, ranges of such row keys serve as units of distribution. Hence similar row keys (and thus data items that are likely to be semantically related to each other) are always kept physically close together, in best case on neighboring sectors on disk, but at least on the same node of a cluster, so that reads of ranges require communication to a minimum number of machines. The smallest units of information are key-value-pairs with the key itself having multiple components. Thus, more formally WCSs can be considered sparse, distributed, multidimensional maps of the form $(table, rowkey, column, timestamp) \rightarrow value$ (see [Ch08]). Using OPE is essential for encrypting row keys to preserve the order of the rows and thus the way of data distribution. Only immutable OPE schemes should be used for that task, since mutable OPE schemes would cause row keys to change over time. Because row keys are used for coordinating distribution, that would result in changing the data's physical position inside the database (cluster), which is prohibitively expensive. That is why most WCSs do not

[2] meaning "IND-OCPA": ciphertexts reveal nothing, but their order

even support changing row keys at all. However, mutable OPE schemes can be used for the column data itself to gain more security and performance. Since disk access and memory management in WCSs are performed at column family level, it is advisable to build the indices in the same way. In particular having one global index for all the system's order-preserving encrypted data items should be avoided, because it poses a performance bottleneck and a security threat (a compromised index would affect the entire database).

4 Experiments

For our experiments we inserted up to 10.000 uniformly distributed and randomly created numeric values into Cassandra and HBase using two OPE schemes that are practical, based on the criteria we introduced in section 2. Firstly, we use [BCO11] for which there are no alternatives, when immutable ciphertexts and having no state is desired. Secondly, we use [KS14], because its need for additional data structures is minimal (only a clientside index). Both schemes do not require further architectural components. We use a plaintext space size $|D|$ of 25 bit and a ciphertext space size $|R|$ of 32 bit, which satisfies the recommendations of the authors of both schemes. While for [BCO11] the order of insertion does not matter, there are three cases to consider for [KS14]. The best case is when all elements of a perfectly balanced binary search tree are inserted in pre-order traversal order. The average case is a uniform input distribution. The worst case is inserting pre-sorted values. We use local installations to avoid network effects, as we want to measure the computation time of the schemes in combination with the insertion speed of the databases. All implementations were done in Java 8. We ran our experiments on an Intel Core i7-4600U CPU @ 2.10GHz, 8GB RAM, a Samsung PM851 256GB SSD using Ubuntu 15.04.

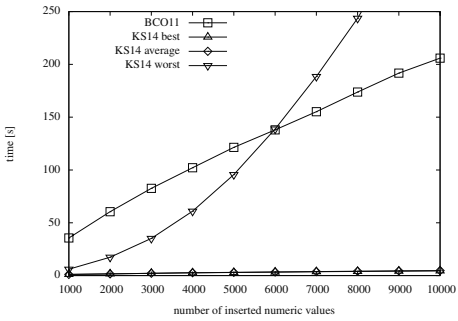

Fig. 1: Time needed for encryption with increasing data set size in Apache Cassandra

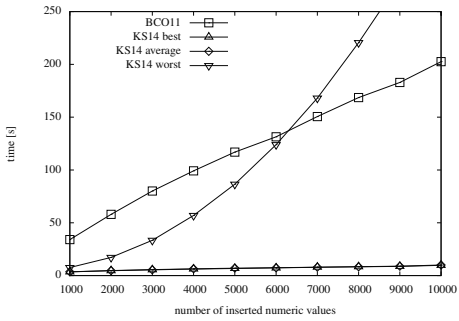

Fig. 2: Time needed for encryption with increasing data set size in Apache HBase

Figure 1 and 2 present the results, showing the average of five measurements. Both databases perform equally well for [BCO11], but [KS14] in its best and average case is much less computationally expensive and thus faster by a factor of roughly 45 using Cassandra and still 20 using HBase, which means Cassandra is twice as fast in those scenarios. This can be explained by two reasons. Firstly, [KS14] is so fast that the database systems pure insertion time requires a significant share in the the overall process of encrypting and inserting (which is not the case for the expensive encryption process of [BCO11]). Secondly,

[KS14] needs no re-encrypting for 10.000 values with the given sizes for plaintext and ciphertext space (in fact the first re-encryption occurs after inserting ca. 12.000 values on average). With Cassandra being optimized for writes it takes advantage of both. It can be further observed that [KS14] gets prohibitively slow when encrypting ordered values, starting to perform even worse than [BCO11] after 6000 insertions (which already require over 300 re-encryptions). Hence not only writing but also reading performance matters. Interestingly in this case HBase is always 12-15% faster than Cassandra, which seems to reflect the fact, that while Cassandra is optimized for writes, HBase is optimized for reads. However Cassandra performs equally or better in all other disciplines.

Because decrypting is very simple, we do not elaborate on it in the same level of detail as we did for encrypting. It does not even involve the database platforms. In [KS14] it is just a lookup in the clientside index which takes less than 1 ms. In [BCO11] it requires rather expensive computations, taking 68 ms on average due to Javas BigInteger class. This performance can be improved by using standard Integer types or caching already decrypted values, which results in the same speed ($<$ 1 ms) as for using [KS14].

5 Related Work

So far there is not much work using OPE with real world technologies. The most popular example surely is “CryptDB” [Po11] utilizing the immutable scheme of [Bo09], tweaked by operating with a binary search tree and caching in the background. Another system for executing queries over encrypted data is “Monomi”, also using [Bo09] for OPE. Both approaches are designed for working with SQL-based systems.

6 Conclusion and Future Work

We discussed how OPE can be used in NoSQL WCSs and quantified the performance of two OPE schemes on the two currently most popular platforms. As we already did the same for a couple of schemes for searchable encryption [WJW15], our next goal is to build a seamless integrating proxy client similar to “CryptDB” for executing more sophisticated queries on encrypted WCS databases.

Acknowledgement: This work was funded by the DFG under grant number WI 4086/2-1.

References

[Ag04] Agrawal, Rakesh; Kiernan, Jerry; Srikant, Ramakrishnan; Xu, Yirong: Order preserving encryption for numeric data. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. ACM, pp. 563–574, 2004.

[BCO11] Boldyreva, Alexandra; Chenette, Nathan; O’Neill, Adam: Order-preserving encryption revisited: Improved security analysis and alternative solutions. In: Advances in Cryptology–CRYPTO 2011, pp. 578–595. Springer, 2011.

[Bo09] Boldyreva, Alexandra; Chenette, Nathan; Lee, Younho; O'Neill, Adam: Order-preserving symmetric encryption. In: Advances in Cryptology-EUROCRYPT 2009, pp. 224–241. Springer, 2009.

[Bo11] Borthakur, Dhruba; Gray, Jonathan; Sarma, Joydeep Sen; Muthukkaruppan, Kannan; Spiegelberg, Nicolas; Kuang, Hairong; Ranganathan, Karthik; Molkov, Dmytro; Menon, Aravind: Apache Hadoop goes realtime at Facebook. In: Proceedings of the SIGMOD International Conference on Management of Data. ACM, pp. 1071–1080, 2011.

[Ch08] Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C; Wallach, Deborah A; Burrows, Mike; Chandra, Tushar; Fikes, Andrew: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2):4, 2008.

[KS14] Kerschbaum, Florian; Schröpfer, Axel: Optimal average-complexity ideal-security order-preserving encryption. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 275–286, 2014.

[Li14] Liu, Zheli; Chen, Xiaofeng; Yang, Jun; Jia, Chunfu; You, Ilsun: New order preserving encryption model for outsourced databases in cloud environments. Journal of Network and Computer Applications, 2014.

[LM10] Lakshman, Avinash; Malik, Prashant: Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review, 44(2):35–40, 2010.

[Ma15] Mavroforakis, Charalampos; Chenette, Nathan; O'Neill, Adam; Kollios, George; Canetti, Ran: Modular Order-Preserving Encryption, Revisited. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, pp. 763–777, 2015.

[Ok11] Okman, Lior; Gal-Oz, Nurit; Gonen, Yaron; Gudes, Ehud; Abramov, Jenny: Security issues in nosql databases. In: Trust, Security and Privacy in Computing and Communications, 2011 IEEE 10th International Conference on. IEEE, pp. 541–547, 2011.

[PLZ13] Popa, Raluca A; Li, Frank H; Zeldovich, Nickolai: An ideal-security protocol for order-preserving encoding. In: IEEE Symposium on Security and Privacy. pp. 463–477, 2013.

[Po11] Popa, Raluca Ada; Redfield, Catherine; Zeldovich, Nickolai; Balakrishnan, Hari: CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles. ACM, pp. 85–100, 2011.

[Ro15] Roche, Daniel; Apon, Daniel; Choi, Seung Geol; Yerukhimov, Arkady: POPE: Partial order-preserving encoding. Technical report, Cryptology ePrint Arch. 2015/1106, 2015.

[So] SolidIT: DB-Engines Ranking. `http://db-engines.com/en/ranking`, 19.01.2016.

[Tu13] Tu, Stephen; Kaashoek, M Frans; Madden, Samuel; Zeldovich, Nickolai: Processing analytical queries over encrypted data. In: Proceedings of the VLDB Endowment. volume 6. VLDB Endowment, pp. 289–300, 2013.

[WJW15] Waage, Tim; Jhajj, Ramaninder Singh; Wiese, Lena: Searchable Encryption in Apache Cassandra. In: Proceedings of the 8th Symposium on Foundations and Practice of Security (FPS). Springer, 2015.

[Wo13] Wozniak, Sander; Rossberg, Michael; Grau, Sascha; Alshawish, Ali; Schaefer, Guenter: Beyond the ideal object: towards disclosure-resilient order-preserving encryption schemes. In: Proceedings of the 2013 ACM workshop on Cloud computing security. ACM, pp. 89–100, 2013.