Searchable Encryption in Apache Cassandra

Tim Waage, Ramaninder Singh Jhajj, Lena Wiese

Institute of Computer Science
University of Göttingen
Goldschmidtstrasse 7
37077 Göttingen
Germany
{tim.waage,lena.wiese}@uni-goettingen.de, r.jhajj@stud.uni-goettingen.de

Abstract. In today's cloud computing applications it is common practice for clients to outsource their data to cloud storage providers. That data may contain sensitive information, which the client wishes to protect against this untrustworthy environment. Confidentiality can be preserved by the use of encryption. Unfortunately that makes it difficult to perform efficient searches.
There are a couple of different schemes proposed in order to overcome this issue, but only very few of them have been implemented and tested with database servers yet. While traditional databases usually rely on the SQL model, a lot of alternative approaches, commonly referred to as NoSQL (short for "Not only SQL") databases, occurred in the last years to meet the new requirements of the so called "Web 2.0", especially in terms of availability and partition tolerance. In this paper we implement three different approaches for searching over encrypted data in the popular NoSQL database Apache Cassandra (offered by many cloud storage providers) and run tests in a distributed environment. Furthermore we quantify their performances and explore options for optimization.

Keywords: Searchable Encryption, Benchmarking, Apache Cassandra

1 Introduction

Industry is moving towards distributed data storage due to the increased amount of data being produced every day and the requirements of Web 2.0 services, needing high availability, consistency and partition tolerance [1] as well as good properties concerning scalability on commodity hardware. NoSQL databases running in distributed cloud environments were designed to meet those requirements. They provide ease of use and flexibility at low costs, without needing the customer to worry about the consumption of resources for storing and sharing data. Furthermore cloud service providers often provide such storage space which can be booked flexibly on demand.

However, outsourcing sensitive data to third party storage providers has always been a security risk, in the private sector (e.g. sharing of photos or health

information, messaging) as well as in the business sector (e.g. classified documents or confidential mailing). Not only adversaries with physical access to data servers are potentially dangerous, (honest but) curious or malicious database administrators of hosting providers also may snoop on sensitive data and thereby pose a thread. NoSQL databases usually do not provide any mechanisms to ensure confidentiality of the data items they are storing. Thus this lack of security features often impedes a wider use of cloud storage.

Encryption is always a handy countermeasure in such untrustworthy environments. It can ensure confidentiality of the externally stored data records against any illegitimate read accesses, but it is usually connected to some limitations concerning the interacting possibilities with the encrypted data, in particular when it comes to searching. Several symmetric *searchable encryption* schemes have been proposed to overcome this issue [2], but to our knowledge none of them have been tested with existing cloud database technologies. Thus in this paper we make the following contributions:

- We implement three schemes for searchable encryption [3–5].
- We quantify the performance of all schemes in a small distributed environment consisting of two nodes.
- Furthermore we evaluate their usability in practice and discuss performance optimizations.

2 Apache Cassandra

Apache Cassandra [6] can be considered as key-value store, as well as (wide) column family store. Its data model was designed to represent loosely structured data items like they are typical for the Web 2.0. It is currently the most popular database in its category[1]. Based on a strictly symmetric peer-to-peer concept Cassandra uses the Gossip protocol for coordination purposes in a distributed installation. It makes use of the local file system and runs in a single Java process per node. Concerning the CAP Theorem [7, 1] Cassandra offers availability and partition-tolerance.

In contrast to most traditional relational SQL-based systems Cassandra does not provide user, role or access rights management. Frontends have to offer this functionality, if desired (for example, the cloud storage interface). Storing the data in an encrypted form can provide a much higher level of security, but Cassandra does not provide any native mechanisms for doing so.

3 The searchable encryption schemes

This section gives a brief overview of the different ideas of the searchable encryption schemes that we used: the CGK scheme [4] proposed by Curtmola et.

[1] SolidIT: DB-Engines Ranking. `http://db-engines.com/en/ranking`, accessed 13/07/2015

al, the HK scheme [5] proposed by Hahn and Kerschbaum as well as the SWP Scheme [3] proposed by Song et. al. As the amount of space for this article is limited we refer the reader to the original papers for more detailed information on how the schemes work. For security definitions, see [2].

3.1 CGK [4] - Index per Keyword

The approach of Curtmola et al. is very promising in terms of search time. It relies on an index consisting of an array A which stores lists of document identifiers from document set D containing unique words and a lookup table T to identify the first element in A for a particular word being searched. The index is created per unique word from the document set D instead of per document. The CGK scheme[2] is the first of its kind to achieve optimal search time and the index generated is linear in the number of distinct words per document. Due to the way data is indexed, updates are expensive which makes this scheme more suitable for 'write once' databases. The non-adaptive version provides IND-CKA1 (indistinguishable under chosen keyword attacks) security, the adaptive version IND-CKA2.

3.2 HK [5] - Index per Document

The also IND-CKA2 secure HK algorithm works with two indices: γ_f and γ_w. The main idea is to store all unique words per document (in contrast to CGK) in the forward index γ_f using a special encrypted form similar to SWP. In addition γ_w is an inverted index storing the outcome of previous searches for providing future results in constant time. The encryption process needs as much iterations as there are unique words per document. Thus it can be very fast depending on the given dataset. On the other hand that means it can by design only deliver information on whether a searchword occurs in a document or not.

3.3 SWP [3] - Sequential Scan

The sequential scan based SWP algorithm[3] is almost the only choice when it is desired to avoid having an index (e.g. for practical reasons) [2]. The basic idea is to encrypt words of a fixed size n and embed a hash value within the ciphertext using a specific form. During search the hash value gets extracted again. If the value is of that special form there is a match. SWP does not require any sort of state information, thus it is instantaneously ready to encrypt or search and easy to implement for many scenarios. In contrast to most index based schemes it also delivers information about the exact number (and positions) of matches in documents. On the downside as being typical for linear scan algorithms, encryptions and searches take linear time and thus potentially very long for large datasets. SWP is IND-CPA (indistinguishable under chosen plaintext attacks) secure.

[2] Whenever we refer to the CGK Algorithm in this paper, we mean its "non-adaptive" version.

[3] Whenever we refer to the SWP Algorithm in this paper, we mean its "final scheme".

4 Implementation

We implemented all three schemes in Java 8. Concerning the necessary cryptographic primitives we used the implementations of two different crypto providers: the Java Cryptography Extension (JCE) as well as the Legion of Bouncy Castle package (BC)[4]. In case both providers offered the desired functionality, we always chose the one that performed faster. In order to connect to Apache Cassandra we used the Java Driver 2.1 in combination with the current version 3 of the Cassandra Query Language (CQL).

5 Benchmarks

We employ the popular scenario of using searchable encryption for data in a mailbox. We use a subset of the TREC 2005 Spam Track Public Corpus[5]. We assume average mailbox sizes of 1,000 mails up to 10,000 mails. Hence, we start our measurements with the first 1,000 mails of the corpus and increase that number up to 10,000 mails to see how the schemes and database scale. Thereby the number of plaintext words increases from roughly 700,000 to over 7 million with around 40% of the words being unique in the sense of the HK scheme. Note that every word in a mail counts, even words like "a", "the", "in" and so on. That means a search is possible for every word, too.

Cassandra is a key-value store, which means the mail documents have to be mapped somehow to a key-value format. We do that as follows. All the mails are written into one table of one keyspace. Thereby the file path of a mail within the data set is used as unique key and the encrypted mail content as its value. Of course more sophisticated structures would be possible, e.g. splitting up the mails into sender, receiver, body and so on, then use appropriate extra keys. However for the sake of simplicity we use this basic format, since there is no reason to expect doing it otherwise would have a serious impact on the results.

In our experimental setup the client connects to a distributed Cassandra Cluster consisting of two nodes, each equipped with a Intel Core i7 3770 CPU (@ 3.4GHz) and 16 GB RAM, running Ubuntu 14.04 LTS and Apache Cassandra 2.1.4. All measurements include the time caused by the required network traffic.

5.1 Encrypting

In our first test we measure the time taken by the encryption process, which also includes the time needed for outputting the results (encrypted files itself as well as lookup tables, indices etc. where necessary) to the database.

As can be seen in figure 1 the time needed for encryption grows linearly in all schemes. The HK scheme is the fastest with the SWP scheme being not significantly slower. Both schemes beat the CGK scheme roughly by a factor of

[4] The Legion of the Bouncy Castle. `http://bouncycastle.org`, accessed 13/07/2015

[5] Available at `http://plg.uwaterloo.ca/~gvcormac/trecspamtrack05`, accessed 13/07/2015

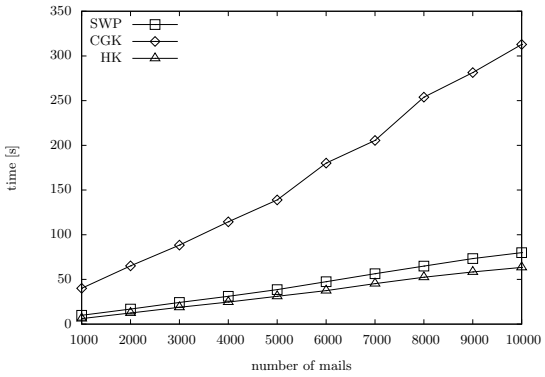

Fig. 1. Time needed for encryption with increasing data set size

4.5. Its creation of the array A of linked lists is much more complex than the encryption steps of the other schemes. Thus the SWP and HK schemes manage to encrypt between 95.000 and 130.000 words per second, the CGK algorithm reaches only circa 23.000 words per second, which can still be considered feasible in practice.

5.2 Searching

In our second test we measure the time taken by the search process for one single word, since all three schemes do not provide a better way than using a trivial "AND"-combination for multiple words. In order to allow a fairer comparison we slightly modified the SWP scheme by allowing to abort the search within a document as soon as the first match occurs and continue with the next document. Thus it delivers the same information as the other schemes, namely whether a document contains the search word or not.

Figure 2 presents the results. The high encryption effort of the CGK scheme pays off in sublinear search time (0.13 seconds when searching 10.000 mails). Due to its index γ_w only the HK scheme can be faster (constant search time), but only if searching for the same word again (HK2). It performs orders of magnitude worse when searching a word for the first time (HK1). Then it is almost as slow as the SWP scheme. Note that the SWP scheme as slowest one in our test still manages to search over half a million words per second.

6 Options for Optimization

During our tests we noticed that all schemes leave room for optimizations in practice, which we describe briefly.

The CGK scheme creates an array A in which it stores a list of document identifiers for each distinct word. While creating the index, if the insert command

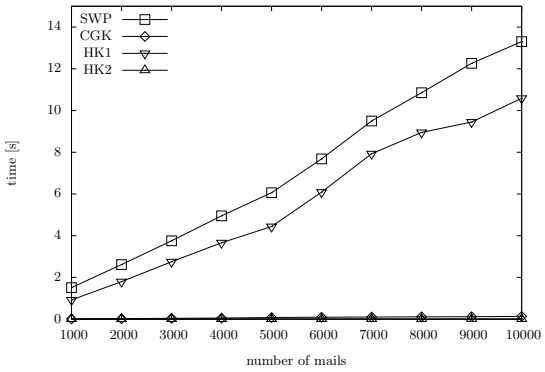

Fig. 2. Time needed for searching with increasing data set size

is executed separately for each node from the number of lists, this results in a significant performance hit in terms of time taken to build the index. In order to optimize this, we used bulk insertions of the nodes into the database to reduce the number of interactions with the servers. For our data set we found the optimal value to be 500 nodes to be inserted at a time, which results in almost 65% improvement compared to single insertions.

The HK scheme barely allows performance optimizations in terms of speed, like the previous schemes do. A potential problem in practice rather is the index size of γ_f. As it can become quite large, one solution to prevent it from growing too fast is reducing the output length of the used pseudo random number generator (in the original work referred to as G). That causes the encrypted representations stored in γ_f to be smaller without being a security issue. In doing so we achieved up to 20% less disk space consumption for γ_f.

As mentioned earlier the SWP scheme uses words of a fixed length n, achieved by splitting and/or padding the original plaintext words. As the algorithm needs as much iterations as there are words to encrypt, a large n improves the overall performance (less iterations needed), while a small n can save disk space (less padding needed). For our individual data set we found the optimal value to be $n \geq 8$, which was 35% faster compared to $n = 4$.

7 Related Work

Since the presentation of the SWP [3] scheme for sequential scan on encrypted data, numerous variations have been proposed. A recent survey [2] provides an excellent source for comparing the different schemes. The main differentiation is between symmetric and asymmetric primitives used. Asymmetric searchable encryption (public key encryption with keyword search or short "PEKS") is commonly used in a setting where multiple users can write the encrypted data

by using the public key while search can be done by a single reader having the private key. Yet, that is more inefficient than the symmetric variants.

The sequential scan introducing SWP scheme has also been applied as a search function over relational data in CryptDB [8]. Most of the subsequent schemes follow an index-based approach (besides the tested CGK and HK scheme e.g. [9–11]), because it proved to be efficient in particular for large data sets although the index size might become too large to be stored in memory [12] ([12] also presents practical experiments and benchmarks on searchable encryption on relational databases). However, relying on an index is not always possible. Building and maintaining indices is costly, especially if the dataset is very dynamic. Indices also require some sort of appropriate keyword assignment.

8 Conclusion and Future Work

We put three algorithms for searchable encryption into practice, namely the index-per-keyword based CGK scheme, the index-per-document based HK scheme the and the sequential scan based SWP scheme. We implemented them in Java and used Apache Cassandra as underlying database. We pointed out strengths and weaknesses in practical environments and quantified their performance in a distributed environment. Furthermore we discussed optimization strategies.

The CGK scheme is not as fast as the others when encrypting with roughly 23,000 words per second. With HK encrypting up to 95,000 and SWP encrypting even up to 130,000 words per second, it is 4–5 times slower. Still the results indicate that a practical usage of all schemes in real world applications seems feasible. The same applies for searching, where the SWP scheme processes up to 530,000 and the HK scheme up to 660,000. The expensive encryption procedure of the CGK Scheme pays off in the search process, in which it is 8–10 times faster than the others.

Future work can extend these results in various ways. On the one hand there is the need to support other functionality required by database queries. There are approaches for the search of multiple keywords at the same time, but with the effort of additional data structures [13, 14]. Schemes for order preserving encryption like [15, 16] can be used for range scans as well as for database internals like managing timestamps and sorting row keys. Thereby as much cryptographic functionality as possible should be done using Cassandras user defined functions to make sure encryption can be used in environments with no other components (e.g. like front ends), too. However for some tasks (e.g. query rewriting) a proxy client between the application and the database is inevitable. On the other hand tests with larger datasets in much larger clusters are required. Therefore we intend to run tests on popular cloud computing platforms like Google Cloud Platform or Amazon EC2, which provide the functionality for deploying Apache Cassandra and other NoSQL databases.

9 Acknowledgement

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

References

1. Brewer, E.: A certain freedom: thoughts on the CAP theorem. In: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing, ACM (2010) 335–335
2. Bösch, C., Hartel, P., Jonker, W., Peter, A.: A survey of provably secure searchable encryption. ACM Computing Surveys (CSUR) **47**(2) (2014) 18
3. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on, IEEE (2000) 44–55
4. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM conference on Computer and communications security, ACM (2006) 79–88
5. Hahn, F., Kerschbaum, F.: Searchable encryption with secure and efficient updates. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM (2014) 310–320
6. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review **44**(2) (2010) 35–40
7. Brewer, E.A.: Towards robust distributed systems. In: PODC. (2000) 7
8. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: Processing queries on an encrypted database. Communications of the ACM **55**(9) (2012) 103–111
9. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Advances in Cryptology–CRYPTO 2013. Springer (2013) 353–373
10. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Advances in Cryptology-ASIACRYPT 2010. Springer (2010) 577–594
11. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of the 2012 ACM conference on Computer and communications security, ACM (2012) 965–976
12. Cash, D., Jaeger, J., Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M.C., Steiner, M.: Dynamic searchable encryption in very large databases: Data structures and implementation. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). Volume 14. (2014)
13. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. Parallel and Distributed Systems, IEEE Transactions on **25**(1) (2014) 222–233
14. Wang, B., Yu, S., Lou, W., Hou, Y.T.: Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: INFOCOM, 2014 Proceedings IEEE, IEEE (2014) 2112–2120
15. Boldyreva, A., Chenette, N., ONeill, A.: Order-preserving encryption revisited: Improved security analysis and alternative solutions. In: Advances in Cryptology–CRYPTO 2011. Springer (2011) 578–595
16. Kerschbaum, F., Schröpfer, A.: Optimal average-complexity ideal-security order-preserving encryption. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM (2014) 275–286