Secure Keyword Search over Data Archives in the Cloud
Performance and Security Aspects of Searchable Encryption

Christian Neuhaus[1], Frank Feinbube[1], Daniel Janusz[2] and Andreas Polze[1]
[1]*Operating Systems and Middleware Group, Hasso Plattner Institut, Potsdam, Germany*
[2]*DBIS Group, Humboldt-Universit¨at zu Berlin, Berlin, Germany*

Keywords: Keyword Search, Searchable Encryption, Cloud Computing, Performance, Security, Data Confidentiality.

Abstract: Encryption can protect the confidentiality of data stored in the cloud, but also prevents search. To solve this problem, searchable encryption schemes have been proposed that allow keyword search over document collections. To investigate the practical value of such schemes and the tradeoff between security, functionality and performance, we integrate a prototypical implementation of a searchable encryption scheme into a document-oriented database. We give an overview of the performance benchmarking results of the approach and analyze the threats to data confidentiality and corresponding countermeasures.

1 INTRODUCTION

Data sharing is essential to companies and government services alike. A striking example is healthcare, where doctor's offices, hospitals, and administrative institutions rely on exchange of information to offer the best level of care and optimizing cost efficiency at the same time. For scenarios like these, moving to the cloud solves many problems: The scalability of the cloud makes resources simple to provision and extend and centralization of data improves the availability and helps to avoid information silos. Most importantly, cloud computing helps to reduce IT expenses – an effect most welcome in healthcare. However, concerns about data confidentiality still prevent the use of cloud in many domains. Traditional encryption is of little help: It effectively protects the privacy of data but also prevents important operations such as search. While efficient encryption schemes that enable generic operations on encrypted data are still elusive, searching over encrypted data is possible: searchable encryption schemes enable keyword search without disclosing these keywords to the cloud operator. The query performance of such schemes cannot match unencrypted operation, but may well be suitable for areas of application such as electronic health records, where data has to be retrieved from a cloud-hosted archive.

In this paper, we investigate the trade-off between performance and security when using searchable encryption schemes for data archives in the cloud. We make the following contributions:
1) We report on an architecture for integrating Gohs Z-IDX searchable encryption scheme (Goh et al., 2003) into a database and present a practical implementation by the example of MongoDB.
2) We discuss the overhead introduced by encrypted search and provide benchmark results on the performance of using Gohs scheme for encrypted search with MongoDB. These benchmarks give a meaningful account of the practical performance and usability of searchable encryption in databases.
3) We give a qualitative assessment of the security implications of using searchable encryption schemes for cloud data archives using attack-defense-tree models. This assessment is generic to searchable encryption and not limited to Goh's scheme. We also discuss mitigation strategies to manage threats by statistical inference attacks.

2 RELATED WORK

In this section, we review related work in the field of private database outsourcing and searchable encryption.

Private Database Outsourcing. Outsourcing private data to a remote database inherently bears the risk of exposure of confidential information – through eavesdropping, data theft or malfunctions. The key

Neuhaus C., Feinbube F., Janusz D. and Polze A..
Secure Keyword Search over Data Archives in the Cloud - Performance and Security Aspects of Searchable Encryption.
DOI: 10.5220/0005428704270438
In *Proceedings of the 5th International Conference on Cloud Computing and Services Science* (CLOSER-2015), pages 427-438
ISBN: 978-989-758-104-5
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

challenge is to protect private data from being accessed by potentially untrusted cloud providers. In this paper, we focus on technologies that protect data within a database. While encryption is the basic mechanism to ensure data confidentiality, providing an efficient database-as-a-service that can run on encrypted data is a challenging task. Several recent approaches try to offer solutions for outsourcing private databases.

TrustedDB (Bajaj and Sion, 2011) and Cipherbase (Arasu et al., 2013) offer SQL database functionalities that support the full generality of a database system while providing high data confidentiality. Both systems use a secure co-processor for performing operations on the cloud server side. The drawbacks of such approaches are at least twofold: On one hand all clients have to trust the secure co-processor with their private data. On the other hand it is not clear how the co-processor scales up in the number of clients connected and the amount of data processed. In CryptDB (Popa et al., 2011), the authors apply an layered approach that makes use of several cryptographic schemes, where values are only decrypted to a level that is required to complete the query.

Another class of approaches aims at processing encrypted data directly without any decryption. To this day, there are no efficient encryption schemes that enable fully encrypted operation of a DBMS (database management system) without loss of functionality. An early approach for keyword search on encrypted data was published by (Song et al., 2000). An approach for securely processing exact match queries on database cells was proposed by (Yang et al., 2006). However, most DBMS rely on other common operations such as range and aggregation queries as well as updates, inserts and deletes. Existing approaches cannot efficiently process this type of queries on encrypted data. A common solution is to reduce data confidentiality to gain query efficiency, e.g., order preserving encryption (Agrawal et al., 2004) may reveal the underlying data order. Most methods can be attacked by statistical analysis of the encrypted data or the access patterns. Another solution is to lose some query efficiency in order to guarantee confidentiality. While *(fully) homomorphic encryption schemes* as proposed by Rivest et al. (Rivest et al., 1978) in fact allow the encrypted computation of any circuit (and therefore computer program), current constructions (see (Gentry, 2009; Van Dijk et al., 2010)) are yet too inefficient for practical application.

Traditional databases use indices for efficient record search. The existing methods have been adapted to work on encrypted data (Shmueli et al., 2005). Private indexing (Hore et al., 2004) enable an untrusted server to evaluate obfuscated range queries with limited information leakage. Wang et al (Wang et al., 2011) propose a secure B^+-Tree to efficiently process any type of database query. Encrypted index-based approaches do not rely on any trusted third parties or trusted hardware. This seems to be a practical and secure method to search in encrypted databases. The next section discusses searchable encryption.

Searchable Encryption. Searchable encryption schemes provide one or many cryptographic data structures called *search indices* that allow encrypted keyword search for exact keyword matches. A good overview of searchable encryption schemes is given in (Kamara and Lauter, 2010). In general, searchable encryption schemes do not replace symmetric encryption schemes but provide the search capability through additional data structures – the *index* (see figure 1). To provide keyword search on data, a list

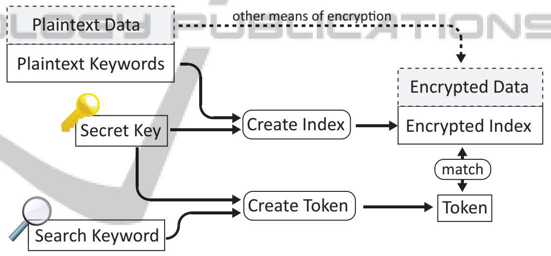

Figure 1: Searchable Encryption: Conceptual View.

of keywords is extracted from the plaintext. This keyword list is used to create a secure index using a dedicated secret key for the searchable encryption scheme. The data is encrypted separately (usually symmetric block ciphers such as *AES*) and uploaded stored alongside the encrypted index in a remote location. To search over the uploaded data in the remote location, a *search token* in generated for a search keyword using the secret key. This token is is sent to the remote server. The remote sever can now determine whether the token matches a search index without being able to learn the keyword.

Searchable encryption schemes can be distinguished between *Symmetric Searchable Encryption* (SSE) and *Asymmetric Searchable Encryption* (ASE) schemes. SSE schemes use the same secret key both for insertion and searching of data. In general, they are more efficient than ASE schemes and provide stronger security guarantees. They were first introduced by (Song et al., 2000), where the authors provide a linear search capability over ciphertext – one of the few schemes that does not make use of indices. To speed up search, the scheme of Goh (Goh et al.,

2003) uses indices that are created separately for every searchable data item, which enables efficient update. Improved search time can achieved by using an inverted index (see e.g. (Curtmola et al., 2006)). A scheme that enables both efficient updates and optimal search time (linear in the number of documents that contain the keyword) is offered in a recent construction by Kamara et al. (Kamara et al., 2012).

In contrast, ASE schemes use different keys for insertion and searching of data, which provides greater flexibility. However, the constructions of ASE schemes are generally less efficient than those of SSE schemes and provide weaker security guarantees. The first construction was given by Boneh et al. (Boneh et al., 2004) and is based on elliptic curve cryptography. Improved constructions were introduced in (Abdalla et al., 2005). Unfortunately, ASE are generally susceptible to dictionary attacks against search tokens (see (Byun et al., 2006)). This limits the application of ASE schemes to use case where keywords are either hard to guess or the keyword attack is tolerable.

3 THE Z-IDX SCHEME

For our implementation, we chose the Z-IDX searchable encryption scheme by (Goh et al., 2003). As a symmetric scheme, it is not susceptible to dictionary attacks on search tokens like ASE schemes (see section 2). This scheme offers several desirable properties:

- **Maturity.** While the field of research in searchable encryption schemes is rather young, Goh's scheme was one of the earliest proposed. In contrast to more recent constructions, the scheme passed several years without the discovery of security flaws.
- **Per-document Indexing.** The Z-IDX scheme creates per-document indices. This property facilitates integration into existing DBMS.
- **Standard Cryptographic Primitives.** The cryptographic mechanisms used by Z-IDX are widely available in software libraries for most platforms.

In this section, we give an overview of *Bloom Filters* and how they are used to construct Gohs Z-IDX scheme.

3.1 Bloom Filters

The encrypted indices in Z-IDX make use of space-efficient probabilistic data structures called *bloom filters* (Bloom, 1970). For a set of elements $E = \{e_1, ..., e_n\}$, the set membership information is encoded in a bit array of length l. A number of r hash functions $h_1, ..., h_r$ is selected that map every element of E to a number $\in [1; l]$. To store the set membership of an element e_x in the filter, its hash value from every hash function $h_1, ..h_r$ is calculated. These hash values $h_1(e_x), ..., h_r(e_x)$ are used as index positions in the filter bit array. At every referenced index position, the bit in the array is set to 1. To test the set membership for an element e_y, the procedure is similar: All hash values $h_1(e_y), ..., h_r(e_y)$ are calculated and used as index positions in the filter bit array. If all positions in the array pointed to by the hash function values are set to 1, the element is assumed to be in the set.

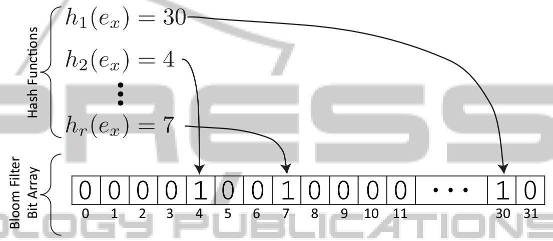

Figure 2: Example of a Bloom Filter with a 32-bit array.

This design of bloom filters can produce false positives: If all corresponding array positions of an element e_z were set to 1 by insertion of other elements, the bloom filter produces a false positive for e_z. On the other hand, false negatives do not occur. The false positive rate of a bloom filter can be influenced by adjusting the size of the bit array and the number of hash functions used.

3.2 Gohs Secure Indexes

Based on bloom filters, Goh constructs a secure index scheme called *Z-IDX* (Goh et al., 2003) that allows encrypted keyword search. Like similar schemes, it does not replace other means of encryption but provides additional data structures for its functionality (see figure 1). The scheme builds upon the abstraction of *documents*, which are the units of granularity for keyword search. Every document $d_i \in D$ can contain a number of keywords $w \in W$ and is identified by a unique ID $i \in I$. Authorized clients hold a secret key K_{priv}. The scheme is then defined by the following operations:

- `Keygen`(s) outputs a secret key K_{priv}, where s is a variable security parameter.
- `Trapdoor`(K_{priv}, w) outputs a trapdoor T_w for keyword w using the secret key K_{priv}.
- `BuildIndex`(d, K_{priv}) outputs an encrypted index for document d using the secret key K_{priv}.

- `SearchIndex`(T_w, d) takes a trapdoor for keyword w and tests for a match in the index of document d. If d contains w it outputs 1 and 0 otherwise.

Additionally, a pseudorandom function $f : \{0,1\}^* \times \{0,1\}^s \to \{0,1\}^m$ is required. For a precise formal definition, e.g. with respect to bit string lengths, please see the original publication (Goh et al., 2003). We also omit the step of adding *blinding bits* to the filter. To set up the scheme, security parameter s, a number of hash functions r and a index size m are chosen (for choice of m and r, see section 5.1). Then, a secret key is generated by the `Keygen` operation, so that $K_{priv} = (k1, ..., k_r) \leftarrow \{0,1\}^{sr}$.

To create a search index for a document d with a set of keywords $W_d = \{w_1, ..., w_x\} \subset W$, `BuildIndex` operation first creates an empty bloom filter with a bit array of length m. First, a trapdoor T_w is calculated for every keyword w using the `Trapdoor` operation, so that $T_w = (t_{w_1}, ..., t_{w_r}) = (f(w, k_1), ..., f(w, k_r))$. This results in a set of trapdoors : Using the set of trapdoors $T_{w_1}, ..., T_{w_x}$ and the *id* of the document d, the set of codewords $C_{w_1}, ..., C_{w_x}$ is calculated. For every trapdoor T_w the codeword C_w is calculated so that $C_w = (c_{w_1}, ..., c_{w_r}) = (f(id, t_{w_1}), ..., f(id, t_{w_r}))$. Then, the filter of the document is populated by setting every bit position ti `1` that is referenced by the trapdoors: For every trapdoor C_w, the bits at positions $c_{w_1}, ..., c_{w_r}$ are set to `1` (see figure 2).

To query a collection of documents for a keyword w, the trapdoor T_w is calculated using the `Trapdoor` operation and sent to the server. To test whether a document contains the keyword, the server calculates the codeword C_w using the trapdoor T_w and the document *id*. Using the trapdoor C_w the server tests whether all bit at positions $c_{w_1}, ..., c_{w_r}$ are set to `1`. If so, the document is sent back to the client as a match. This process is applied to all documents in the collection. In the Z-IDX scheme, a separate index data structure is created per document. This accounts for a search time that is linear over the number of documents, but facilitates the administration of secure indices, as they can be created stored alongside the documents. This makes the addition or removal of documents a simple operation.

From a more technical perspective, the above steps can be described and implemented using a keyed hash function such as HMAC-SHA1 (Krawczyk et al., 1997), which is also used in our implementation (see section 4). In a first step, a keyword w is hashed with all elements of the secret key $k_1, ..., k_r$ to obtain the trapdoor vector. The elements of the trapdoor vector are each hashed again together with the document identifier *id* to obtain the codeword vector. Each of the codeword vector elements is used as an index position to set a bit in the bloom filter bit array to 1.

4 SEARCHABLE ENCRYPTION IN MongoDB

To evaluate the practical usability of searchable encryption, we integrated the Z-IDX scheme into the document oriented database *MongoDB*. In this section, we explain why we chose MongoDB, present the architecture of our prototype, introduce new commands for secure keyword search and present implementation details.

4.1 Selection of a Database System

While the searchable encryption scheme Z-IDX can be used standalone, its practical usability and performance under realistic workloads can only be evaluated if the scheme is used in conjunction with other means of encryption and data handling. To do this, we integrated Z-IDX into an existing DBMS. The choice of a DBMS has to correspond to the basic properties of the Z-IDX scheme – exact keyword matching as a search mechanism and the notion of *documents* as the basic units of granularity for searching.

To select a DBMS, we considered different database paradigms: The most widespread type of databases are **relational databases** – most of them supporting the *Structured Query Language* (SQL). This type of database has a long development history and offers features such as transactional security, clustering techniques and master-slave-configurations to ensure availability. The SQL language allows detailed queries, where specific data fields in the database can be selected and returned base on complex criteria based on structure or data field values and logical combinations thereof. The expressive power of the SQL language goes far beyond simple keyword search. It is therefore difficult to isolate queries that can make use of searchable encryption. Additionally, the fine-grained selection of data fields does not correspond well to the document-oriented approach of searchable encryption.

Besides relational databases, other database types have been developed under the umbrella term of NoSQL databases. A very minimalistic approach are *key-value stores* (e.g. *Redis, Dynamo*): They omit many of the features known from SQL databases in favor of simplicity and performance. However, the complexity of data structures is severely limited. This makes storing documents and associated indices difficult or impossible.

Document-oriented databases, however, are well-suited to implement searchable encryption. As the name suggests, data is organized in containers called *documents* as opposed to tables in relational databases. These documents are the units of granularity for search operations and can contain complex data structures without adhering to a schema definition. As this approach corresponds well to the properties of searchable encryption schemes, we chose to add searchable encryption features to the open-source document-oriented database *MongoDB*.

Floratou et al. (Floratou et al., 2012) compare MongoDB to Microsoft SQL Server. They show that relational databases may have better query performance. However, MongoDB is optimized for storing data records across multiple machines and offers efficient load balancing, which makes it more suitable for cloud-based applications. Furthermore, the increasing use of NoSQL databases in real world applications lead to an increasing demand for enhancing these databases with privacy technologies such as searchable encryption.

4.2 Extended MongoDB Commands

As MongoDB is a document-oriented database, a *document* is the primary unit of abstraction for organization of data. A document does not adhere to a fixed schema and can store data in a JSON-like fashion of field-value pairs. Like in JSON, documents support a number of primitive data types (e.g. `integer`, `String`) and a data structures like arrays. All of these data structures can be nested. In addition to standard JSON, MongoDB can also store binary data in fields.

Documents in MongoDB are stored in *collections*, these, in turn, are stored in a *database*. The prime commands for data handling in collections are `insert()` and `find()`. They accept a document as a parameter. To make searchable encryption explicitly available, we introduced two additional commands:

- The `insertSecure()` can be used to insert documents into a collection using searchable encryption. Using this command, every array of strings in the document is removed and its content used as keywords. The contained strings are inserted into a Z-IDX filter or encrypted search. Every other datatype remains untouched.
- The `findSecure()` command triggers encrypted search over all documents of a collection. As a parameter, it takes a keyword embedded in a document, e.g.: `findSecure({keyword: 'foo'})`

4.3 Architecture and Implementation

To integrate searchable encryption into MongoDB, we chose to add the extended functionality to the server and the command line client. An overview of the architecture of MongoDB server and client is given in figure 3. In theory, it is possible to add searchable encryption to MongoDB modifying only the client but not the server. However, this leads to a disproportionately high increase in communication overhead as per-document operations would have to be carried out on the client, each requiring the transmission of the documents Z-IDX data structures.

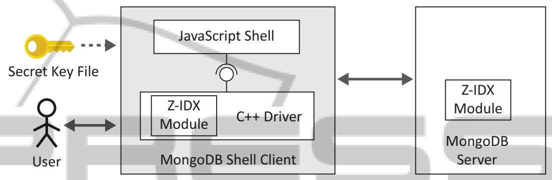

Figure 3: Architecture of MongoDB Server and Client.

The MongoDB command line client is comprised of a JavaScript shell that uses a core driver written in C++. The client connects to the server, which is also written in C++. To provide searchable encryption functionality, we implemented the Z-IDX scheme (see section 3) and additional helper functions in a separate module that is compiled both into the server and the C++ driver of the client (*Z-IDX Module*, see figure 3). As suggested by Goh, we apply data compression (zlib) to the index data structures before transmission over the network. As these data structures are very sparse, the compression works very effectively and the additional compute overhead is easily outweighed by reduced transmission times in most settings.

To integrate the functionality, we made the following modifications: The **JavaScript shell** is modified to read the secret key information from a file, which has to be passed as a parameter at startup. If a secure search or insert request is identified, the request is modified to include the secret key information. This information is stored in a dedicated `_zidx` field in the query. After this, the request is passed to the clients' C++ driver. The **C++ driver** is modified to recognize queries that contain Z-IDX key information injected by the JavaScript shell. For inserts, a Z-IDX filter is built and populated with the contained strings of every string array in the document. Subsequently, the string arrays and the key information are removed and the command is passed on to the server. For a search query, the C++ driver uses the key to compute trapdoors for every search keyword. The trapdoors are inserted, the key is removed and the query is passed on to the server. The **MongoDB server** is modified to process the search queries. For the trapdoors of a search query, the server generates the corresponding codewords using the *document id*. These codewords are then checked against the bloom filters of a document to test for a match. This architecture and implementation makes searchable encryption available without affecting non-encrypted use of the database, as regular MongoDB commands are processed as expected.

5 PERFORMANCE EVALUATION

The use of encrypted search functionality introduces an overhead in computation, storage and data transmission. Since speed and throughout are critical factors for databases, we present performance measurements of our approach in this section. The figures allow to evaluate the practicability of searchable encryption in databases for real-life scenarios.

To assess the performance impact of our approach, we ran insert and search queries in encrypted and unencrypted settings under various parameters settings (dictionary size, false positive rate) and analyzed the memory footprint of the additional data structures of Z-IDX. To avoid synthetic test data, we chose the publicly available *Enron corpus* – a collection of emails which we use as documents. All benchmarks were run on a Intel Core i5-3470 machine with 8GB main memory, running Ubuntu 12.04 LTS.

5.1 Memory Footprint of Z-IDX Filters

As the encrypted filters are added to every document, they add overhead to communication and storage footprint. They are therefore a crucial factor that influences the performance of a database using this scheme.

The size of these data structures is determined by the desired false positive rate f_p and the number of unique keywords to be represented by the filters n. From the false positive rate f_p, the number of hash functions r is determined by calculating $r = -log_2(f_p)$. From r, the number of bits m in the filter can be determined by calculating $m = nr/ln2$. In practice, these data structures can become quite large. This is especially unfavourable in settings with large numbers n of distinct keywords and small document sizes, as the filter sizes can easily exceed the size of the original documents.

To improve the efficiency of the scheme, data compression can be used on the filters (as suggested by Goh). While filter compression decreases storage and communication overhead, it also introduces additional steps of computation on the client and server side: Upon document insertion, filters have to be compressed and decompressed for every search operation. This represents a tradeoff between data size and computational overhead.

To investigate this issue, we first tested the effectiveness of compression on indexes. In practice, these filters are bit array that contain mostly `0`'s and sparsely distributed `1`'s (depending on the number of contained keywords). To determine the achievable compression ratio, we used a set of 1000 documents from the Enron corpus containing 127.5 keywords on average. Assuming a set of 100000 distinct keywords and a false positive rate of 0.0001% leads to an uncompressed filter size of 252472 bytes. We implemented the compression of filters using the free *zlib*[1] compression library. Using the *zlib* standard compression strategy, the average compression ratio achieved is 0.02 with the given parameters. Using a run-length encoding strategy that exploits the sparse property of the filters, compression becomes even more effective with an average compression ratio of 0.0154. This means that using compression, filter sizes can be considerably reduced in size (here: to 1.54% of their original size, average size of compressed filters 3889 bytes).

Our benchmarking results show that using filter compression dramatically speeds up database operations even over fast network connections (100 Mbit/s speed). This means that the overhead for data compression is by far outweighed by the advantage in network transmission speed due to smaller filters. Therefore, we use RLE-based filter compression as a default in all subsequent measurements.

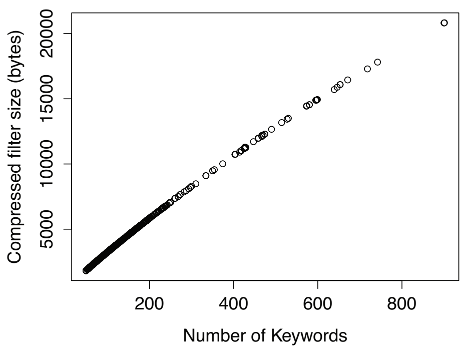

Figure 4: Relationship between number of document keywords and compressed filter size.

It can be observed that the size of compressed filters is closely correlated with the number of represented keywords (see figure 4): Documents with few

[1]http://zlib.net/

keywords have small compressed filters while more keywords produce larger sizes. This means that a trade-off of the Z-IDX scheme is mitigated: To accommodate large sets of distinct keywords without false-positives, large filter sizes are required. These large filters take up of large amounts of memory – even for small documents with few or no keywords at all. However, using compression, filter sizes can be generously chosen as compressed filters remain compact, depending on the number of keywords in the document. In fact, using the settings above, compressed filter sizes are 3389 bytes on average. When increasing the number of unique keywords from 100000 to a million (tenfold) , the average size is only 6648 bytes on average (only a twofold increase).

5.2 Query Performance

To assess the performance of the scheme, we evaluated insert and search performance of our Z-IDX implementation embedded in MongoDB. To obtain realistic results, we tested our setup under two different network profiles: The LAN profile corresponds to the typical properties of a wired local network (2ms ping, 100 Mbit/s), the WAN profile corresponds to the properties of a domestic internet connection in Germany (20ms ping, 10 Mbit/s). For reference, the same benchmarks were also conducted with a Localhost profile, where the network delays are essentially nonexistent. The LAN and WAN profiles were generated by using network link conditioning on the machines' loopback network device, using Linux' `tc` command. All benchmarks were conducted using a false positive rate of 0.001 and a maximum dictionary size of 10000.

Insert Query Performance. To assess the performance of insert queries, we inserted a collection of 10000 documents from the Enron corpus in batches of 100. We ran every insert query 100 times and took the mean as our measurement value. The results for these queries in the Localhost, LAN and WAN profiles for encrypted and unencrypted operation are shown in figure 5. The longer duration of encrypted operation is explained by the additional steps required on the client: Before submission of a document, a Z-IDX filter has to be created using the document's keywords and the document content has to be encrypted. The Z-IDX filter introduces data which slightly increases the time of data transmission. On the server, no additional steps have to be executed on insert. Our experiments show that the performance penalty for encryption in insert queries is indeed moderate: In the Localhost- and LAN-settings, the insert time is about doubled

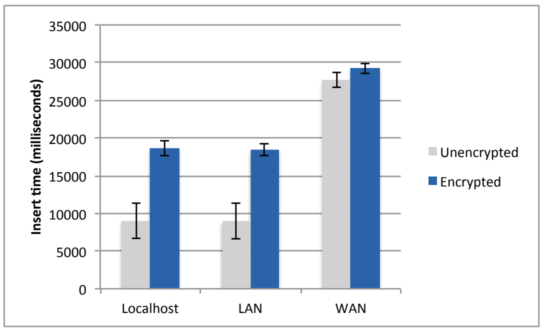

Figure 5: Benchmark: Insert of 10000 documents.

compared to the unencrypted setting. In the WAN setting, where network performance has a larger effect, the duration of encrypted and unencrypted insert queries are nearly the same.

Search Query Performance To determine the performance of search queries, we issued a search query with a randomly chosen keyword on the same document collection as used in the insert queries. We ran every search query 100 times and took the mean as our measurement value. The results for these queries in the Localhost, LAN and WAN profiles for encrypted and unencrypted operation are shown in figure 6. The duration of encrypted search queries

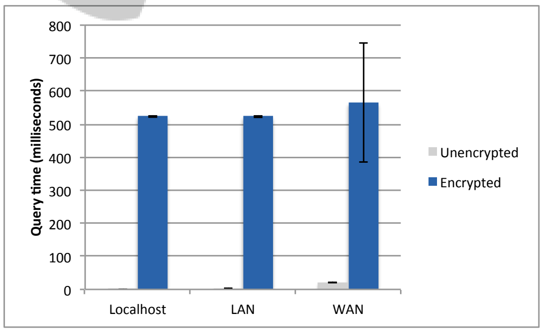

Figure 6: Benchmark: Query over 10000 documents.

is increased significantly compared to unencrypted operation, due to a fundamental difference between search implementation: Searching in an unencrypted database is usually carried out using an inverted index, where the matching documents for a given keyword can be looked up with linear complexity ($O(1)$). In encrypted operation using the Z-IDX-scheme, search complexity is linear in the number of documents in the collection ($O(n)$, n=number of documents). As a result, the unencrypted search time is very small (0,13 ms in the Localhost setting, 2,32 ms LAN, 20,37 ms WAN) when compared to encrypted operation and mainly determined by the network latency. In contrast, encrypted searches took around half a second ($\approx$ 530 ms), with little variation depending on network performance, as only little data had to be transmitted.

5.3 Implications for Practical Use

Our measurements have shown that the performance penalty for using the Z-IDX searchable encryption scheme in a database is very unevenly distributed: While the performance penalty for insert queries is almost negligible in under realistic conditions (WAN profile), the penalty for search queries is tremendous by comparison. At the same time, the query performance varies greatly depending on collection size (linear effort) and filter parameters: A search query on a 10000-documents-collection in our experiments took between 219 ms ($f_p = 0.01$, $n = 1000$) and 4612 ms ($f_p = 0.0001$, $n = 100000$).

6 SECURITY

The motivation for using searchable encryption schemes such as Z-IDX is to protect the confidentiality of information that is stored on untrusted infrastructures (e.g. cloud providers). In this section, we give a qualitative evaluation of the security implications when searchable encryption schemes are used to search over encrypted data stored on a remote server. This security evaluation is generally applicable to searchable encryption schemes that correspond to the abstract model given in section 6 and therefore not specific to Goh's Z-IDX scheme (Goh et al., 2003), unless explicitly noted otherwise.

The *security* of computer systems constituted by the attributes of confidentiality, integrity and availability (as defined in the ITSEC criteria (ITSEC, 1991), see also (Avizienis et al., 2004)). As the purpose of searchable encryption is to protect the searched keywords from being disclosed to unauthorized parties, we focus our evaluation on the property of data confidentiality of search keywords.

Abstract System Model. For the security evaluation, we assume a setup as shown in figure 7 (see also (Islam et al., 2012)). A server holds a set of n documents $Doc_1, \dots, Doc_n$. It also holds an encrypted data structure which contains a mapping for every keyword $w \in W$ to all documents containing w. To query the encrypted index, the client generates a trapdoor T_w and sends it to the server over the network. Using this trapdoor, the server can determine all documents that contain keyword w and sends them back to the client over the network. The mapping between keywords and trapdoors $w \mapsto T_w$ is deterministic, i.e. under the same encryption key there exists exactly one trapdoor T_w for every keyword w. These properties apply to most symmetric searchable encryption schemes.

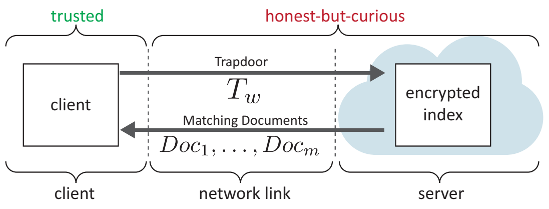

Figure 7: Encrypted Search on a remote system: Abstract Model.

Attacker Model. Attacks to learn the plaintext of keywords and their association with encrypted documents can generally be undertaken in any part of the architecture. Attacks on the client are the the most dangerous, as clients hold the cryptographic key and handle unencrypted information. We assume authorized users on these clients to be trustworthy. For the operator of the network link and the server we assume a honest-but-curious attacker model (see e.g. (Lindell et al., 2008)) : These operators will generally execute programs and transmit information correctly and faithfully, but can record arbitrary information and perform additional calculations on it. Under this adversarial model, data confidentiality is challenged while integrity and availability are not affected.

6.1 Threats to Keyword Confidentiality

To illustrate the threats to the confidentiality of keywords in the system we use the ADTree model (Attack-Defense-Trees, see (Kordy et al., 2012; Bagnato et al., 2012)), which build upon the concept of attack trees (Schneier, 1999). Attack trees are used to model the threats to a specific security property of a system and their logical interdependencies. Individual threats are represented as leaves of the tree and are connected by AND and OR operators to the root of the tree, which represents a specific security property. The attack of the system that corresponds to a specific threat is indicated in the model by assigning a boolean `TRUE` value of the node in the tree. If a combination of attacks results in a propagation of a `TRUE` value to the root node the security property is considered to be breached. By evaluating the attack tree, sets of possible attacks can be derived. The ADTree model extends attack trees by introducing and explicitly modeling countermeasures, which can be employed to mitigate or prevent attacks. In figure 8, an ADTree shows threats for keywords confidentiality in searchable encryption schemes and ac-

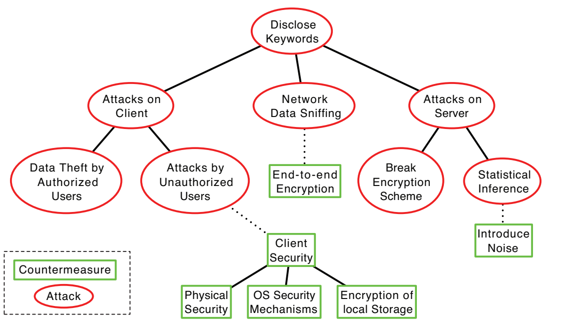

Figure 8: Attack-Defense-Tree: Threats for Confidentiality of Keywords.

cording countermeasures. Attacks to learn keywords can be undertaken on the client, on the network and the server which holds the encrypted index. In the following sections, we discuss the relevance and implications of the shown threats and their countermeasures.

6.2 Attacks on the Client

Attacks on the client are potentially severe as the client handles plaintext data and holds the cryptographic key for the searchable encryption scheme. By obtaining the key, an attacker can uncover document-keyword associations by generating valid queries and launching a dictionary attack against either the server or against intercepted trapdoors. Theft of data or keys cannot by authorized users cannot be prevented. However, in our attacker model, we assume the authorized users to be trustworthy. To protect the assets of the client systems against unauthorized users, different methods can be employed: Physical security measures can prevent unauthorized users from getting physical access to client machines. The security mechanisms of the clients operating system can ensure that only authorized users can log onto the machines directly or via network. Finally, data on the clients mass storage can be protected by hard disk encryption.

6.3 Network Data Sniffing

Interception of data exchanged by searchable encryption protocols could threaten the confidentiality of keywords as statistical properties of the trapdoor-keywords-associations can be exploited (for more detail, see section 6.4). If general security flaws of the underlying scheme become known, these could also be exploited. Data sniffing on the network can however easily be prevented by encryption of network traffic between client and server (e.g. by using *Transport Layer Security*).

6.4 Attacks on the Server

In general, threats that originate from network data sniffing also exist on the server, as the entire communication of the scheme is observable. However, as the searchable encryption scheme has to be processed on the server (i.e. matching of trapdoors to documents), an additional layer of encryption is not an option. In addition, the server also has direct access to the encrypted index, which could make attacks targeting this data structure very efficient. As the server can also monitor the program execution, side-channel attacks are theoretically possible (e.g. timing attacks). In the following, we discuss the implications of these threats.

Attacks to the Encryption Scheme. The confidentiality of the keywords depends on the trust in the chosen underlying searchable encryption scheme. In the first place, it is desirable to use algorithms that are openly published and examined by cryptographic experts. In general, searchable encryption schemes are an active field of research, with many constructions from the recent past (see section 2) that need more evaluation before they can be considered mature.

The Z-IDX scheme by Goh is among the oldest searchable schemes with no general attacks to the scheme published. The construction of the scheme is based on keyed hash functions, which are well examined and proved cryptographic tools (HMAC SHA-1 (Krawczyk et al., 1997)). The scheme fulfills three security properties suggested by (Song et al., 2000): It supports *hidden queries* as the generated trapdoors do not reveal the keyword. Valid trapdoors cannot be generated without possession of the secret key (*controlled searching*). Both properties are ensured by using a keyed hash function. Finally, the scheme fulfills the property of *query isolation* which means that the server learns nothing more than the set of matching documents about a query. This security property is formalized as the *IND-CKA* (Semantic Security Against Adaptive Chosen Keyword Attack) property: An adversary is given two documents D_0 and D_1 and an index which encodes the keywords of one of these documents. If the adversary cannot determine which documents keywords are encoded in the index with a probability significantly better than $\frac{1}{2}$ the index is considered *IND-CKA-secure*. To the best of our knowledge, no attacks that break IND-CKA-security of the Z-IDX scheme have been published to date.

Statistical Inference. Attacks using statistical inference are a possible against all searchable encryption schemes that follow the basic model outlined in section 6. The threat of these attack is not based on weaknesses in the cryptographic constructions of searchable encryption schemes but is a direct consequence of the basic characteristics of such schemes. Under the same secret key K_{priv}, a keyword w is always mapped to the same trapdoor T_w. This allows the server to observe tuples $(w, \{D_1^w, ..., D_m^w\})$, i.e. combinations of encrypted queries and the set of matching documents, which leak statistical information: The sever can learn the frequency of certain queries as they occur over time and learn about the occurrence and frequency of distinct keywords in the document collection. While statistical information does not directly reveal keywords, it can be exploited to infer the semantics or plaintext of keyword using background knowledge about the data exchanged in the system. When handling medical data for example, very accurate assumptions about the prevalence of a specific medical condition among a population can be made using public sources of information. If this prevalence is expressed using a keyword and no other keyword in the document set possesses the same frequency, it is easy to infer the meaning of this keyword. While the given example might be trivial, statistical attack can pose a serious threat to the confidentiality of keywords. We review two practical attacks that have been published:

Search Pattern Leakage in Searchable Encryption: Attacks and New Constructions. (Liu et al., 2013) propose an attack based on the frequency of search patterns. The salient feature of the approach is that the frequency f_q at which a keyword q occurs is sampled over time, resulting in a frequency vector $V_q = \{V_q^1, ..., V_q^p\}$ for a specific keyword. Background knowledge for a dictionary of keywords $D = \{w_1, ..., w_m\}$ is drawn from external sources (the authors propose *Google Trends*) and represented as frequency vectors $V = \{V_{w_1}, ..., V_{w_m}\}$. To infer the plaintext of a keyword, a distance measuring function $Dist(V, V_{w_i})$ is used to determine the vector $\in V$ with the smallest distance to V_q – the corresponding keyword is then assumed to be q. The attack is amended by an active approach, where the background knowledge is adapted to a specific scenario (e.g. healthcare) to improve accuracy. To test the accuracy of their attack, they use frequency vectors obtained from Google Trends for the 52 weeks of the year 2011 and add varying levels of gaussian noise to simulate user queries. They show that under certain circumstances (e.g. keyword dictionary size of 1000, limited level of noise) it is easy to guess the keyword with a very high accuracy. They also present mitigation strategies, which are based on inserting random keywords along with every query, but do not consider the actual document matching on the server.

Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation. (Islam et al., 2012) propose a statistical attack which is based on the frequency at which keywords appear in the document set. As background knowledge, information about the probability of two keywords occurring in the same document is assumed. This information can be obtained by scanning public document sources for a dictionary keywords $k_1, ..., k_m$. It is represented by a $m \times m$ matrix M, where $M_{i,j}$ contains the probability of keywords k_i and k_j occurring in the same document. The attacker then tries to find an order of encrypted queries $q_1, ..., q_m$ whose results set produce another matrix which is similar to M. This sequence that produces the matrix most similar to M is considered the result of the attack and reveals keywords by aligning the vectors of queries and keywords so that q_x corresponds to m_x. The problem can be formalized by expressing the closeness between matrices as an arithmetic distance. The authors use simulated annealing to determine a keyword sequence that minimizes this distance. The quality of the attack is the percentage of keywords that are guessed correctly. This percentage is improved if the background knowledge also includes a set of known query-trapdoor associations – this is however not required. With 15% known queries of 150 observed queries, their attack was able to infer close to 100% of a set of 500 keywords correctly. To counteract the presented attack, they also suggest the insertion of noise to hide statistical properties of the query-document associations. Encrypted index structures are considered $(\alpha, 0)$-secure if for every keyword there are $\alpha - 1$ keywords that appear in the same set of documents - limiting an attackers probability of correctly inferring a keyword to $\frac{1}{\alpha}$ at best.

6.5 Implications for Practical Use

The threat model in section 6.1 shows that attacks on the confidentiality are possible in every part of the system. However, as shown in the previous sections, attacks by unauthorized users on the client and the network can effectively mitigated by access control and encryption. The most relevant threat is the possibility of inferring keywords by exploiting statistical properties that can be observed by monitoring queries. The threat posed by statistical inference attacks depends strongly on the set of keywords and their distribution in the document set. Statistical inference attacks are only a minor concern if the individual keywords exhibit very similar statistical properties, e.g. serial numbers that are evenly distributed across documents. However, attributes with statistical properties that could be available as background knowledge (e.g. medical diagnoses) to an attacker need to be treated with great caution and might require noise insertion.

7 CONCLUSION

In this paper, we evaluated the practical usability of searchable encryption for data archives in the cloud, illustrated by embedding an implementation of Goh's searchable encryption scheme into MongoDB. We found that the use of compression on the additional data structures keeps the data size at tolerable levels and relative to the number of embedded search keywords. Performance benchmarks revealed that for insert operations under typical network parameters, the additional overhead for insert operations is negligible compared to unencrypted operation. Search queries however exhibit a considerable impact for encrypted operation, as search operations are linear to the number of documents in Goh's scheme. However, the measured durations of encrypted queries could be acceptable for interactive use where the added security is required. To evaluate the security properties of searchable encryption, we presented threats to keyword confidentiality as an attack-defense-tree model, which applies to most searchable encryption schemes. The most relevant threat comes from inference attacks, which are possible if the keywords exhibit strong statistical properties which can be extracted using background knowledge. In such cases, noise insertion techniques can be used to mitigate such attacks.
Further research could investigate the performance more recent constructions of searchable encryption schemes with constant search complexity (e.g. (Kamara et al., 2012)) and schemes that provide extended search capabilities, such as range queries (see e.g. (Boneh and Waters, 2007; Wang et al., 2011)).

ACKNOWLEDGEMENTS

The authors would like to thank Martin Kreichgauer for providing the prototypical implementation of the Z-IDX scheme and the MongoDB integration.

REFERENCES

Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., and Shi, H. (2005). Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *Advances in Cryptology–CRYPTO 2005*, pages 205–222. Springer.

Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. (2004). Order preserving encryption for numeric data. In *Proceedings of SIGMOD '04 International Conference on Management of Data*, pages 563–574. ACM.

Arasu, A., Blanas, S., Eguro, K., Joglekar, M., Kaushik, R., Kossmann, D., Ramamurthy, R., Upadhyaya, P., and Venkatesan, R. (2013). Secure database-as-a-service with cipherbase. In *Proceedings of SIGMOD '13 International Conference on Management of Data*, pages 1033–1036. ACM.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33.

Bagnato, A., Kordy, B., Meland, P. H., and Schweitzer, P. (2012). Attribute decoration of attack–defense trees. *International Journal of Secure Software Engineering (IJSSE)*, 3(2):1–35.

Bajaj, S. and Sion, R. (2011). Trusteddb: A trusted hardware based database with privacy and data confidentiality. In *Proceedings of SIGMOD '11 International Conference on Management of Data*, pages 205–216. ACM.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Boneh, D., Di Crescenzo, G., Ostrovsky, R., and Persiano, G. (2004). Public key encryption with keyword search. In *Advances in Cryptology-Eurocrypt 2004*, pages 506–522. Springer.

Boneh, D. and Waters, B. (2007). Conjunctive, subset, and range queries on encrypted data. In *Theory of cryptography*, pages 535–554. Springer.

Byun, J. W., Rhee, H. S., Park, H.-A., and Lee, D. H. (2006). Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In *Secure Data Management*, pages 75–83. Springer.

Curtmola, R., Garay, J., Kamara, S., and Ostrovsky, R. (2006). Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88. ACM.

Floratou, A., Teletia, N., DeWitt, D. J., Patel, J. M., and Zhang, D. (2012). Can the elephants handle the nosql onslaught? *Proc. VLDB Endow.*, pages 1712–1723.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of Computing*, pages 169–178. ACM.

Goh, E.-J. et al. (2003). Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216.

Hore, B., Mehrotra, S., and Tsudik, G. (2004). A privacy-preserving index for range queries. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '04, pages 720–731.

Islam, M., Kuzu, M., and Kantarcioglu, M. (2012). Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Network and Distributed System Security Symposium (NDSS'12)*.

ITSEC (1991). Information technology security evaluation criteria (itsec): Preliminary harmonised criteria. Technical report, Commission of the European Communities.

Kamara, S. and Lauter, K. (2010). Cryptographic cloud storage. *Financial Cryptography and Data Security*, pages 136–149.

Kamara, S., Papamanthou, C., and Roeder, T. (2012). Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM.

Kordy, B., Mauw, S., Radomirović, S., and Schweitzer, P. (2012). Attack-defense trees. *Journal of Logic and Computation*.

Krawczyk, H., Bellare, M., and Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational). Updated by RFC 6151.

Lindell, Y., Pinkas, B., and Smart, N. P. (2008). Implementing two-party computation efficiently with security against malicious adversaries. In *Security and Cryptography for Networks*, pages 2–20. Springer.

Liu, C., Zhu, L., Wang, M., and an Tan, Y. (2013). Search pattern leakage in searchable encryption: Attacks and new constructions. Cryptology ePrint Archive, Report 2013/163.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011). Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100. ACM.

Rivest, R. L., Adleman, L., and Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. *Foundations of secure computation*, 32(4):169–178.

Schneier, B. (1999). Attack trees. *Dr. Dobb's journal*, 24(12):21–29.

Shmueli, E., Waisenberg, R., Elovici, Y., and Gudes, E. (2005). Designing secure indexes for encrypted databases. In *Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, DBSec'05, pages 54–68.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE.

Van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers. *Advances in Cryptology–EUROCRYPT 2010*, pages 24–43.

Wang, S., Agrawal, D., and El Abbadi, A. (2011). A comprehensive framework for secure query processing on relational data in the cloud. In *Proceedings of the 8th VLDB Workshop on Secure Data Management*, SDM'11, pages 52–69, Berlin, Heidelberg. Springer-Verlag.

Yang, Z., Zhong, S., and Wright, R. N. (2006). Privacy-preserving queries on encrypted data. In *Proceedings of the 11th European Conference on Research in Computer Security*, ESORICS'06, pages 479–495.