A Secure Frequency Hiding Index for Encrypted Databases

Somayeh Sobati Moghadam

Université Lumière Lyon 2
Email: `Somayeh.Sobati-moghadam@univ-lyon2.fr`

***Abstract*—Cloud computing offers the opportunity of data outsourcing as well as the data management tasks. However, for the sake of various privacy issues, confidential data must be encrypted before outsourcing to the cloud. But query processing over encrypted data without decrypting the data is a very challenging task. Deterministic (DET) encryption scheme allows encrypting data while still enabling efficient querying over encrypted data. The inherit merits of DET make it very suitable and efficient scheme for cloud data outsourcing. But the security of DET scheme is still a challenge while DET is vulnerable to frequency attacks. We present a new scheme for indexing encrypted data, which hides data frequency to achieve a strictly stronger notion of security. The proposed indexing method is secure against frequency attacks, hence, data cannot be recovered from indexes. Moreover, our scheme is still efficient for query processing.**

Keywords–Data outsourcing; data privacy; querying encrypted data.

I. Introduction

A naive solution to preserve privacy is encrypting data before outsourcing to the cloud. In the context of relational databases, the state-of-the-art solutions use property preserving encryption schemes. Property preserving encryption schemes enable processing query over encrypted data without decryption. For instance, order preserving encryption scheme (OPE), preserves the order of ciphertexts as original plaintexts, means OPE preserves the order property. Deterministic (DET) scheme encrypts the same plaintext into the identical ciphertexts, thus, the equality property is preserved [1]. Property preserving encryption schemes are undoubtedly efficient schemes while enabling queries to be directly processed over encrypted data. But such schemes leak some information about the plaintext.

DET scheme allows the server to perform a large number of queries which means it can perform `SELECT` with equality predicates, equality `JOIN`, `GROUP BY`, `COUNT`, `DISTINCT`, etc. [2]. DET scheme is vulnerable to frequency attacks, while DET leaks the frequency distribution of underlying data. In frequency attacks, an adversary not only has access to the encrypted data but also has some prior knowledge about the plaintext domain and its frequency distribution. Frequency attack does not impose any threat when the underlying data has uniform frequency distribution [3]. If the frequency distribution of plaintext not uniform, DET scheme must be replaced with a probabilistic encryption scheme, but it makes query processing impossible over encrypted data.

CryptDB is a first practical system uses property preserving schemes to support a wide range of queries processing over encrypted data. As the best of our knowledge, other systems like BigQuery demo [4], Always Encrypted [5], Cipherbase [6] and Relational Cloud [7] use a DET scheme too. As a result, they are vulnerable to frequency attacks. Naveed et al. in [8] demonstrate that a large fraction of the records from DET encrypted columns, can be decrypted by frequency attacks.

We present a new scheme to improve the security of DET by hiding the frequency of plaintexts. The proposed scheme hides the frequency of plaintexts by means of a new indexing scheme. In our scheme, the indexes have a uniform distribution,and hence the proposed scheme is robust against frequency attacks. While increasing security, our scheme preserves the functionality of DET scheme. This scheme can be applied as an alternative to a DET scheme.

II. Frequency Hiding Scheme

The basic idea is to create an index such that no frequency information from repeated plaintexts leaks. Note that this index should still enable querying while hiding the frequency of data. Therefore, any plaintext value of frequency f is mapped into multiple index values. Thus, the target distribution remains close to flat, i.e., uniform Intuitively, if t distinct plaintext values are mapped into $m > t$ unique values that are of the same frequency, none of these target values can be explicitly mapped to the corresponding plaintext values by the frequency attacks.

To enable efficient query processing, we propose to add some auxiliary metadata for an attribute A at the server side. In our solution, we add an auxiliary column named $EqIdx$ (**Eq**$uality$ $checking$ **In**d**e**x) along an attribute `A`. This column allows equality checking over `A`. To create $EqIdx$ column, first the values of A are sorted in ascending order. Then, an incremental ID is assigned to the sorted values, which are stored in $EqIdx$ column. As a result, for each plaintext value v_i, we have f_i distinct values in the column $EqIdx$, from s_i to l_i. We call s_i and l_i as boundaries of v_i.

In order to query processing, the user should keep the boundaries of each plaintext value; otherwise, when the user has a query, he cannot know how to transform the values in a query. Storing the corresponding boundaries at the user side induces storage overhead which is in contrary with the benefits of data outsourcing. Thus, we store an auxiliary table that maintains the information about the boundaries for each plaintext value. We call this table as *frequency table*, fT. Frequency table's values are encrypted, thus, leak no information about the plaintext values. In order to query processing, the user first sends a query to retrieve the corresponding boundaries from frequency table. Then, using the result of the first query, the user sends another query to retrieve the desired values.

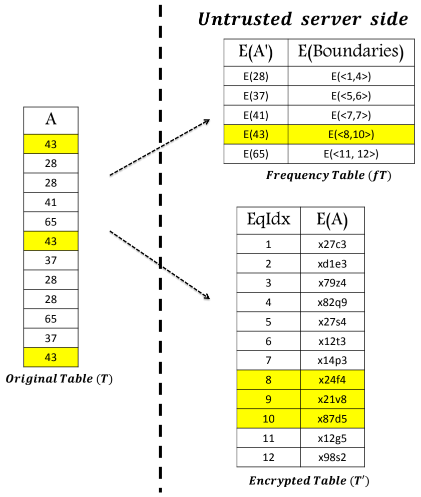
A. Building Index

To simplify our discussion, let us assume a relational table T consists of one column A (additional columns, if any, can be processed similarly) and we wish to encrypt and store it at a service provider. The encrypted version of T is T' at the server side. Considering A has t distinct values $\{v_1, v_2, ..., v_t\}$ with the frequencies $\{f_1, f_2, ..., f_t\}$. First, we sort the plaintext values in ascending order. Then, we assign incremental values in to the sorted data. We store the incremental values in an auxiliary column called $EqIdx$ along the encrypted values of A, $E(A)$. Note that the encryption scheme for encrypting A's values could be a scheme with high security guarantees (e.g., a randomize encryption scheme that encrypts the same plaintexts into the different ciphertexts). Each plaintext value v_i is mapped into f_i distinct values I_i in $EqIdx$ such that $I_i \in [s_i, l_i]$ $\forall i = 1, ..., f_i$. In other words, all values in an interval like $[s_i, l_i]$ are corresponded to the same plaintext v_i.

B. Building Frequency Table

Frequency table fT consists of two attributes, the first one maintains the encryption of distinct plaintext values and the second attribute stores the encrypted boundaries corresponding to the plaintext values.

As we explained before, for any distinct plaintext values v_i in attribute A, we have two corresponding boundaries s_i and l_i, $i = 1, ..., t$. s_i and l_i are concatenated $< s_i \parallel l_i >$, and encrypted with a key k, $E_k(< s_i \parallel l_i >)$. v_i is encrypted with the same key, $E_k(v_i)$. Then, $E_k(< s_i \parallel l_i >)$ and $E_k(v_i)$ $i = 1, ..., t$ are stored in the attributes $E(Boundaries)$ and $E(A')$ in fT, respectively. Figure 1 shows an example of frequency table and $EqIdx$ for attribute A.



A
43
28
28
41
65
43
37
28
28
65
37
43

E(A')	E(Boundaries)
E(28)	E(<1,4>)
E(37)	E(<5,6>)
E(41)	E(<7,7>)
E(43)	E(<8,10>)
E(65)	E(<11, 12>)

EqIdx	E(A)
1	x27c3
2	xd1e3
3	x79z4
4	x82q9
5	x27s4
6	x12t3
7	x14p3
8	x24f4
9	x21v8
10	x87d5
11	x12g5
12	x98s2

Figure 1. Metadata at the server side. Ciphertexts shown are not full-length.

C. Query Processing

Considering a simple query example like "`SELECT * FROM T WHERE A=`v_i". First, the user sends a query to retrieve the boundaries of v_i from fT. Therefore, the user encrypts v_i with k and sends the following query:
`SELECT E`$_k$`(Boundaries) FROM` fT `WHERE E`(A')`=E(`v_i`)`. When the user receives the result, he decrypts and recovers the corresponding boundaries s_i and l_i. Subsequently, the user sends a query using the extracted boundaries to retrieve all values that have $EqIdx$ between s_i and l_i:
`SELECT E(A) From T`$'$ `WHERE` $s_i \leq$ `EqIdx` $\leq l_i$.
Finally, the user decrypts the final results returned back by the server.

Confidentiality in our approach relies on the secure cryptographic scheme using for encryption of data and corresponding metadata. Uniform distribution in $EqIdx$ hides the frequency of data. Thus, our solution is robust against frequency attacks. Note that the drawback of this scheme is its inefficiently for data updating. Typically, our indexing method uses the distribution of plaintext, while update operations may change it, rendering re-calculation unavoidable.

III. Conclusion

In this paper, we propose a new scheme that can effectively resist against frequency attacks in cloud data outsourcing. The adversary in our scenario has knowledge of the frequency of original data in a database. In the proposed scheme, all sensitive data are encrypted at the server side and some metadata is used to query encrypted data. A new indexing scheme is introduced to hide the frequency of data. The proposed solution, not only provides robust security guarantees against frequency attacks, also allows efficient and correct query processing over encrypted data. We plan to introduce security proof for the proposed scheme and extend it in order to support any query over encrypted data.

References

[1] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in CRYPTO, 2007, pp. 535–552. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74143-5_30

[2] R. A. Popa, "Building practical systems that compute on encrypted data," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.

[3] T. Sanamrad, L. Braun, D. Kossmann, and R. Venkatesan, "Randomly partitioned encryption for cloud databases," in DBSec, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-43936-4_20

[4] Google Encrytped Big Query. [Online]. Available: https://github.com/google/encrypted-bigquery-client (retrieved: May, 2016)

[5] Always Encrypted. [Online]. Available: https://msdn.microsoft.com/enus/library/mt163865(v=sql.130).aspx (retrieved: May, 2016)

[6] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with cipherbase," in CIDR 2013, 2013. [Online]. Available: http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper33.pdf

[7] Relational cloud. [Online]. Available: http://relationalcloud.com/ (retrieved: June, 2016)

[8] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in SIGSAC, 2015, pp. 644–655. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813651