Symmetric-key Searchable keyword Concealment (SSC)

10/26/2010

YACOV YACOBI[1]

ABSTRACT

We discuss what is commonly known as "searchable symmetric keywords encryption," although we prefer to replace "encryption" with "concealment," since many of these transformations are not (efficiently) reversible (they look more like one way hashing).

The thrust of this paper is practical approaches to cryptographic solutions for cloud databases security and privacy. We limit the scope of this paper to symmetric key keyword concealment systems, and show the following: (i) for the attacker (with the DB admin as the attacker), any Symmetric-key Searchable keyword Concealment (SSC) with high probability behaves as far as the attacker is concerned like a deterministic cipher (therefore, we might as well use deterministic systems and rip their efficiencies). (ii) Range queries can be handled more securely and more efficiently than current proposals, such as OPE, at the cost of some extra bandwidth, but good compromises are easy to find; (iii) Conjunctions of keywords (and indeed any Boolean predicate) are easy to handle at the cost of some additional information leakage. We do not know of any real example where this extra leak matters.

IACR Category: Cryptographic Protocols.

[1] This work was done when the author was with Microsoft Research.

Keywords:

Cloud cryptography, searchable encryption.

1. INTRODUCTION

Informal Glossary:

Global vs. local Indexes: A *local index* is similar to a keyword list attached to a paper (the *document*, or *record*). A *global index* is similar to the index at the end of a textbook (where pages play the roles of documents).

Search system based on local indexes is inherently inefficient, since it requires searching the local indexes of each of the records in the DB, and hence must run in time linear in the number of records. In contrast, we can search a global index in time logarithmic in the number of records, just as we do with the index at the end of a textbook. We therefore view only the latter as a candidate for practical systems. All the papers that we know of about keywords conjunctions use local indexes (we contribute a small idea in this direction in section 5 of this paper, but we do not view it as a candidate for a practical system).

Result vectors: For a given keyword, a *result vector* is a vector that specifies all the documents that match that keyword.

Internal vs. external conjunctions: Readers of a database (DB) usually specify searches as conjunctions of keywords (KW). If the DB can see each KW separately, find its result vector and then intersect the vectors to find a vector matching the conjunction, then we call it *external conjunction.* If the DB gets a representation of the conjunction and finds the result vector of the conjunction without seeing individual KW, nor their corresponding result vectors, then we call it *internal conjunction.* The latter is more secure than the former (which leak individual result vectors).

Range queries: A range query has the general form: *Give me all the (encrypted) variables to which I have read-access (within some context), whose values are in the range [u,v].*

A *Bloom Filter* (BF, [B]) has two inputs: Representations of a set, S, and of element u. Its output is the truth-value of the predicate $u \in S$. In its simplest form it works as follows: Represent keywords as binary L-tuples. Let F denote a family of random functions [GGM], where for $f \in F$, $f:\{0,1\}^L \rightarrow \{0,1\}^k$. The Bloom Filter is a binary array of length 2^k. It is initially set to the all zeros vector. Pick r random functions $f_i \in F,\ i=1,2,\ldots r$. Let $S=\{w_j\},\ j=1,2,..m$, $w_j \in \{0,1\}^k$, compute $f_i(w_j),\ i=1,\ldots r,\ \ j=1,2,..m$ and set bits $f_i(w_j)$ in the BF array to 1. To check if a new word u is included in S, compute $f_i(u),\ i=1,2,\ldots r,$ and check if bits $f_i(u),\ i=1,2,\ldots r$ are all set to 1 in the BF array of S. If the answer is positive, then with high probability (for a proper parameter choice) $u \in S$. If even in one of these locations the value of the BF entry is zero, then $u \notin S$. In practice, the theoretical family of random functions is replaced by standard keyed one-way hash functions.

Each user is associated with a vector of Boolean attributes. For example, an entry may be "older than 21," another entry may be "holds a valid driver's license," a third may be "resident of Washington State." The user may be associated with additional non binary fields, such as address, age, eye color, and height.

Definition: A *policy* is a Boolean predicate over the vectors of binary attributes.

Each record in a cloud DB is associated with **read** and **write** policy, and they may be distinct. A user is allowed to read or write if and only if her binary attributes fulfill the **read/write** policies, respectively.

Remark: In the literature (ABE [GPSW]) sometimes the starting point is access structures realizable by threshold secret sharing scheme, but they end up implementing Boolean predicates as above (an n out of n threshold implements logical AND, and a 1 out of n implements a logical OR)

In any system we can implement any logical NOT in a policy by propagating all the NOTs to basic attributes, using De Morgan's Law, and treating them as any other attribute.

2. LIMITATIONS ON SECURITY

We find the definitions in [CGKO] very illuminating. They led us to the observation that as far as the cryptanalyst is considered any keyword concealment system behaves with high probability like a deterministic cipher (see below).

Let K, M, C denote the key, message[2], and cryptogram spaces, resp. More specifically, M is the set of messages $\{m_1, m_2 \ldots m_L\}$, where $\Pr[m_i]>0$, $i=1,2,\ldots L$. For a given *deterministic* encryption algorithm, E, and a key $k \in K$, let $C_{E,k}=\{c \mid \exists m \in M, c=E_k(m)\}$. It follows that for any deterministic encryption, E and any key k, $C_{E,k}$ has the same distribution as M.

Let *n* denote the number of *records* in a DB. Suppose that keyword w_i has probability $0<q_i<1$ to be included in any given record (record=file; a keyword with q=1 is useless since it filters nothing). That means that the average Hamming Weight of its result vector is $\sim q_i n$. Two keywords w_i w_j are *independent* if their result vectors are uncorrelated.

The result vectors of keywords w_i and w_j are more likely to be identical when $q_i=q_j=q$ (than when $q_i \neq q_j$). But even then the probability is close to zero when a realistic size DB is considered. In that case, on average the probability that the two result vectors are identical is $\varepsilon = (1-q)^{n(1-q)} q^{nq}$. A practical DB can easily have $n=10^6$, in which case ε is practically zero regardless of the value of q, as long as $0<q<1$. We therefore proceed under the assumptions that each keyword has a unique result vector. Therefore the space of result vectors, S, has the same distribution as the message space M. Knowledge of the distribution of S (which is visible to the attacker; the DB admin) gives the attacker the same benefits as knowing the distribution of C in deterministic systems. I.e. these systems are

[2] Messages=keywords.

(with Pr ≅ 1- ε) as vulnerable as deterministic systems.

Therefore, regardless of how we encrypt the keywords, when viewed together with their result-vectors the encryption is *essentially deterministic*.

It is therefore a waste to encrypt the keywords using complex semantically secure techniques and pay the extra cost of (greatly) reduced efficiency.

3. RANGE QUERIES

[ABO] and [BCLO] propose solutions to range queries. However, [ABO] exposes prefixes, and [BCLO] exposes the relative order of values. We hope to achieve the goal while leaking less info, even if we have to raise the cost a bit. [SBCSP] uses a local index, hence runs in time linear in the DB size, and is not practical.

Trading bandwidth for security:
Range queries are of the form "Give me all the records, to which I have *read* access, in which the value of field *x* is $v(x) \in [v_1, v_2]$." The goal is to answer such queries when $v(x)$, v_1 and v_2 are encrypted in such a way that the DB cannot decrypt them, and without leaking more information than is necessary.

One approach is Order Preserving Encryption (OPE) [BCLO]. An encryption function is order preserving if the cryptograms (viewed as integers) maintain the same relative order as the underlying messages. A reader then issues a query by encrypting the range values v_1 and v_2 with the same function and the same key as the stored data, *x*, and the DB can respond the same way it would respond in the absence of encryption. Such systems leak the order by definition.

We propose a different approach. In our approach the DB may return more records than needed. The user decrypts all of them (she gets only records to which she has *read* access) and throws away the extra records. There is a tradeoff between bandwidth and computation.

The idea is to tag records with a range keyword. For example, if the field is body-temperature, we can have encrypted keywords A, B, C, D, corresponding to the ranges [90-95), [95,97), [97,99), [99,101). If the exact value of the field is 98.6, then the record is tagged with C. Keywords are concealed (encrypted or one-way hashed). If the reader wants to retrieve all the records with body temperature in the range (96,100), she specifies (encrypted) keywords C,D, and she may get records outside the range.

The distribution of documents in intervals (which is different from the distribution of queries discussed in section 3.2) is visible to the attacker. We can flatten it to reduce information leak. For example, if some variable has a Gaussian distribution, we can choose smaller intervals at the center of the bell-curve. This method is more secure than OPE, since (after smoothing the distributions) it does not leak the order. For optimization analysis, see Appendix A.

The above method applies both to symmetric and to asymmetric keyword encryption.

This method leaks less than OPE, but it leaks. For example, two encrypted ranges that are queried together usually mean that the two ranges are adjacent. With this method the attacker does not know the order of the ranges, while with OPE she does[3]. If we have k ranges, then this method has $\log_2(k!)$ extra security bits compared with OPE, which leaks the order (since we have k! permutations to choose from).

4. CONJUNCTIONS

External conjunctions (see glossary): Local indexes require search time linear in the number of documents. This is not practical. All the algorithms for internal conjunctions known to us use local indexes, and inherit their impracticality. We do not know of any real example where the inferior security of external conjunction compared to internal conjunction matters. External conjunctions leak all the individual result vectors, while internal conjunctions leak only their intersection. Note that if a user builds the query gradually, adding keywords until she gets good filtering, then the DB learns a lot about individual result vectors (although not everything) even when using internal conjunctions.

Note also that externally we can handle any binary predicate, not just conjunctions.

Internal conjunctions: We discuss a potential new internal conjunction system, which is simpler than current proposals. Like its predecessors, it uses local indexes, and hence its search time is linear in the number of docs.

We can extend [G] to do simple internal conjunctions as follows: use an encrypted Bloom Filter ([B] see glossary in section 2), BF_1, for each doc as is done now, and create another encrypted Bloom Filter, BF_2, for a conjunctive query. Then just as we check for inclusion of a single word in BF_1 we can check if $BF_2 \subset BF_1$ (do all the 1's of BF_2 fall on 1's of BF_1?). This construction may leak some information about the number of conjuncts. For example, when we have two conjuncts, in ordinary BF the Hamming weight of BF_2 is between *k* and *2k*, where *k* is the number of hash functions used with each individual conjunct. We can mitigate this leak using blinding of BF_2 similar to step 3 in *BuildIndex* of [G]. This increases the probability of false positives, and we have to increase filter sizes to compensate. Maybe this construction leaks nothing more, but we still do not have a proof. Note that if we do not care about the above leak, we do not have to increase the size of the BF's over the case of a single query, since this internal conjunction gives precisely the same results as the corresponding external conjunction. It is also true that the security of individual keywords is not degraded by this construction (compared with [G]).

The new system inherits its impractical search complexity (linear in the number of docs) from [G], and is sub-linear in the number of conjuncts (external conjunction is linear in the number of conjuncts).

[ABO] with external conjunctions is overall more efficient (logarithmic in the global

[3] Randomizing the queries would not buy extra security (over deterministic) even here, since the DB admin can glean the same info from the result vectors.

index size, and linear in the number of conjuncts).

REFERENCES

[ABO] Georgios Amanatidis Alexandra Boldyreva Adam O'Neill, New Security Models and Provably-Secure Schemes for Basic Query Support in Outsourced Databases.

[BBO] M. Bellare, A. Boldyreva, A. O'Neill, Deterministic and Efficiently Searchable Encryption, Crypto'07 Proc. LNCS, Vol. 4622, pp. 535-5 52, A. Menezes ed. Springer, 2007.

[B] BURTON H. BLOOM, Space/Time Trade-offs in Hash Coding with Allowable Errors, Communications of the ACM Volume 13 / Number 7 / July, 1970, pp. 422-426.

[BCLO] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill, Order-Preserving Symmetric Encryption, A. Joux (Ed.): EUROCRYPT 2009, LNCS 5479, pp. 224–241, 2009.

[BKM] L. Ballard, S. Kamara, and F. Monrose, Achieving Efficient Conjunctive Keyword Searches over Encrypted Data, ICICS 2005, LNCS 3783, pp. 414-426, 2005.

[BW] D. Boneh, and B. Waters, Conjunctive, Subset, and Range Queries on Encrypted Data, S.P. Vadhan (Ed.): TCC 2007, LNCS 4392, pp. 535–554, 2007

[CGKO] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions, *CCS'06,* October 30–November 3, 2006 ACM 1-59593-518-5/06/0010

[CKGS] B. Chor, E. Kushilevitz, O. Goldreich, M. Sudan: Private Information Retrieval. J. ACM 45(6): 965-981 (1998)

[CM] Yan-Cheng Chang and Michael Mitzenmacher, Privacy Preserving Keyword Searches on Remote Encrypted Data, J. Ioannidis, A. Keromytis, and M.Yung (Eds.): ACNS 2005, LNCS 3531, pp. 442–455, 2005.

[G] E.J. Goh, Secure Indexes.

[GSW] Philippe Golle, J. Staddon, and B. Waters, Secure Conjunctive Keyword Search over Encrypted Data, M. Jakobsson, M. Yung, J. Zhou (Eds.): ACNS 2004, LNCS 3089, pp. 31–45, 2004.

[KL] S. Kamara, and K. Lauter, Cryptographic Cloud Storage.

[OS] R. Ostrovsky, W.E. Skeith III, A Survey of Single-Database PIR: Techniques and Applications.

[S] C. E. SHANNON, Communication Theory of Secrecy Systems, Bell Systems J. 1949.

[SBCSP] E. Shi, J. Bethencourt,T-H. Hubert, C. D. Song and A. Perrig, Multi-Dimensional Range Query over Encrypted Data.

APPENDIX A: Optimization of range queries

The function *f(k)* that we want to minimize is the number of Concealed Keywords (CK) (times some constant c_1) plus the number of items that the DB sends back as a spill (times some constant c_2). These constants for example include the complexity of cryptographic operations. Let $c_1=ac_2$.

Assume *n* items evenly distributed in *k* compartments. The user specifies *m* compartments (out of the total *k*) and gets *nm/k* items back. We want to find the optimal *k*. The overall range of Real numbers is of size *U*, and the range query is a subset Real interval of size $R\leq U$. The following is an optimization for uniform distribution. For other distributions it makes sense to use non uniform interval sizes, with smaller intervals where density is higher. The user wants to get back a fraction *nR/U* items. She specifies *kR/U* concealed keywords. Due to the "quantization" effect of the intervals, there is a "spill" and she gets some items which are not in the subset of size R. This is a waste that we want to minimize

A typical spill is roughly the compartment size, i.e. *Un/k* (in the worst case it is 2 compartments, i.e. *2Un/k*). Wlg assume $c_2=1$. So,

$$f(k)\approx akR/U+Un/k$$

The value of *k* that minimizes *f(k)* is $k_0=U(n/(aR))^{1/2}$.

EXAMPLE-1: Suppose the variable is human body temperature, with overall range [80,110], and a physician is looking for all records with body temperature in the range [97,103]. So, $U=110-80=30$, and $R=103-97=6$. Suppose $a=1$, and there are $n=10^6$ record to which that physician has read access then the spill to payload ratio is $30/((10^6/6)^{1/2})=7.3485\times10^{-2}$. Namely, there is ~7% waste compared to precise DB (that has no spill). For this query the physician will use $k=U(n/R)^{1/2}=30(10^6/6)^{1/2}= 12247$ concealed keywords. She will get back a payload of $nU/R=10^6*30/6= 5.0\times10^6$ of useful data (and throw away the ~7% spill).

EXAMPLE-2: Suppose that $c_1=ac_2$. Then, assuming $c_2=1$, $f(k)\approx akR/U+Un/k$, and the optimal *k* is $k_0=U(n/(aR))^{1/2}$. This is a factor $a^{1/2}$ better than in the previous example. It could be even that $a=10^4$ (eg, if delivery of a record takes $1\mu s$ and a pairing op takes *10ms*). In that case, $a^{1/2}=100$. For the previous parameters, namely $U=30$, $R=6$, $n=10^6$, we get $k_0=122$.