Efficiency and Security Trade-Off in Supporting Range Queries on Encrypted Databases

Jun Li Edward R. Omiecinski
College of Computing
Georgia Institute of Technology, Atlanta, GA 30332
{junli, edwardo}@cc.gatech.edu

Abstract

The database-as-a-service (DAS) model is a newly emerging computing paradigm, where the DBMS functions are outsourced. It is desirable to store data on database servers in encrypted form to reduce security and privacy risks since the server may not be fully trusted. But this usually implies that one has to sacrifice functionality and efficiency for security. Several approaches have been proposed in recent literature for efficiently supporting queries on encrypted databases. These approaches differ from each other in how the index of attribute values is created. Random one-to-one mapping and order-preserving are two examples. In this paper we will adapt a prefix-preserving encryption scheme to create the index. Certainly, all these approaches look for a convenient trade-off between efficiency and security. In this paper we will discuss the security issues and efficiency of these approaches for supporting range queries on encrypted numeric data.

Keywords: Confidentiality, range query, interval-matching, prefix-matching, random one-to-one mapping, prefix-preserving, order-preserving.

1 Introduction

The database-as-a-service (DAS) model [14] is a new computing paradigm which has emerged recently. To save cost, data storage and management are outsourced to database service providers. In other words, highly sensitive data are now stored in locations that are not under the data owner's control, such as leased space and partners' sites. This can put data confidentiality at risk. Therefore, it is desirable to store data in encrypted form to protect sensitive information. Also queries may reveal private information about the user [6]. In this paper, we discuss how to efficiently support searching functionality, in particular, range queries, while preserving data confidentiality and user privacy. The motivation within this model of processing is to provide security and privacy but also have the database service provider do most of the processing. The strategy is to process as much of the query as possible at the service providers' site. Several approaches have been proposed to generate the index which enables queries to be processed against encrypted data with different levels of efficiency and security [13, 8, 3, 15]. In this paper, we will adapt a prefix-preserving encryption scheme to create the index. Certainly, all these approaches look for a convenient trade-off between efficiency and security. In this paper we will discuss the security issues and efficiency of these approaches for supporting range queries on encrypted data.

One simple way to preserve the confidentiality is to decrypt the data when performing search. There are several drawbacks with this approach. First, all the data stored in the database needs to be decrypted for every query. This is very inefficient in terms of computation. Second, this approach assumes the server is secure and fully trusted. This assumption is less justified in the DAS paradigm.

A major type of database queries is *range-based*, composed of *intervals* in the underlying domain of the attributes. Attributes such as name, not typically thought of as numerical, can be indexed and therefore linearized in some fashion. In this paper we will mainly be concerned with *interval-matching* or *exact-matching* as query conditions[1]. Interval-matching is defined as a boolean function $f_{[a,b]}(x)$, which returns true if and only if $x \in [a,b]$. Because computers can handle only inherently finite and discrete attribute values, one can assume without loss of generality x, a and b are all nonnegative integers. Exact-matching is a special case of interval-matching in which a is equal to b.

The paper is organized as the follows, in Section 2, we survey the related work and

[1]Those more complex query conditions usually can be converted into exact-matching, though it might not be efficient to do so.

discuss possible solutions based on well-known mechanisms. Section 3 shows how a relation is encrypted and stored on the server. In Section 4, we first show how an interval-matching problem can be transformed into a set of prefix-matching problems. Then the prefix-preserving encryption algorithm is presented. In Section 5, we describe that with prefix-preserving encryption how a condition in a range query is translated to a condition over server-side representation and how select operations are implemented. Section 6 analyzes attacks against the random one-to-one mapping scheme, the order-preserving scheme and the prefix-preserving scheme under a particular assumption, while in Section 7 we have some additional discussion on the security of the prefix-preserving scheme. Section 8 compares the prefix-preserving scheme with the random one-to-one mapping scheme in the aspects of client side cost, server side cost and communication cost for supporting range queries. In Section 9, we discuss about how to support query conditions other than exact-matching/interval-matching, and relational operations other than selection. We then conclude the paper in Section 10.

2 Related Work

Recently providing security and privacy in DAS has drawn considerable attention [13, 8, 3, 15]. The bucket index technique proposed in [13, 15] relies on partitioning attribute domains of a client's table into sets of buckets. The index value of each remote table attribute value is the bucket number to which the corresponding plain value belongs. This representation supports efficient evaluation on the database service provider of both exact-matching and interval-matching predicates; however, it makes it awkward to manage the correspondence between bucket numbers and the actual attribute values present in the database. For the convenience of comparison, in the rest of this paper, when we discuss about this approach, we will assume that the size of each bucket is 1 and the bucket number is generated by a random one-to-one mapping of the plaintext value. In this case, the server will not return any redundant data to the client. Therefore, the client does not need any database functionality to filter out unsolicited data. This fulfills the goal of the DAS model, i.e., outsourcing database management and having the database server do most of the work.

In [8], the authors quantitatively evaluate the level of inference exposure associated with the publication of attribute indexes generated by a random one-to-one mapping. To efficiently support range queries on encrypted data, they propose to build a B^+-tree over plaintext values, but then encrypt every tuple and the B^+-tree at the node level using conventional encryption. The advantage of this approach is that the content of B^+-tree is not visible to an untrusted database service provider. The disadvantage is that a lot of data processing has to occur on client machines. This mitigates the advantage of the DAS model.

In [17], a sequence of strictly increasing polynomial functions is used for encrypting integer values while preserving their order. In [3], another form of order-preserving encryption is provided for computing the index. It takes a user-provided target distribution as input and transform the plaintext values in such a way that the transformation preserves the order while the transformed values follow the target distribution. They assume an application environment where the goal is safety from an adversary who has access to all (but only) encrypted values (the so called ciphertext only attack [19]). In this paper, we will not only examine the prefix-preserving scheme under ciphertext only attack, but also examine it under known plaintext attack [19] (i.e., an adversary is assumed to gain full knowledge to certain number of ⟨plaintext, ciphertext⟩ pairs through means other than compromising the key).

A potential technique that can support searching on encrypted data is computing with encrypted data [10, 1]. However, an expensive protocol between clients and database service providers is needed. A closely related topic is Private Information Retrieval (PIR) [6, 7]. PIR mechanisms allow clients to query databases without revealing which entries are of interest. PIR schemes often require multiple non-colluding servers, consume large amounts of bandwidth, and do not guarantee the confidentiality of the data.

Many researchers have investigated the problem of zero-knowledge proof [11, 12]. In particular, several protocols have been proposed [16, 5, 4] to prove that a committed number lies in an interval without actually revealing the number. But the performance in terms of bandwidth and CPU power makes these protocols unattractive to the DAS model. Moreover, it can only be used to preserve data confidentiality, but not query privacy.

3 Data Organization

In a relational DBMS, data are organized in tables (e.g., employee data in Table 1, where the underlined attribute represents the key of the table). The database can be encrypted with regard to different units, which can be individual table, a column of a table, a row (tuple) of a table or a given column within each row (i.e., the data item value). Encrypting at a coarser level of granularity such as a table implies that entire table must be returned as the result of a query, although encryption/decryption will be more efficient. Encrypting at a finer level such as a data item allows for more efficient query processing but requires

FNAME	LNAME	SSN	ADDRESS	SALARY	DNO
John	Smith	123456789	731 Fondren, Storrs, CT	30000	5
Franklin	Wong	333445555	638 Voss, Storrs, CT	40000	5
Alicia	Zelaya	999887777	3321 Castle, Storrs, CT	25000	4
Ahmad	Jabbar	987987987	980 Dallas, Storrs, CT	25000	5
James	Borg	888665555	450 Stone, Storrs, CT	55000	1

Table 1: Employee

Enc_tuple	I_{SSN}	I_{SALARY}	I_{DNO}
vWHkaHTF7J1p1ZFX	068764019	6488	250
otvfOg5dFQbGNFKp	277737042	45639	250
uJKJkM3huAsbI5C3	080581877	53798	224
tp1eSkSEtiae54V3	203690710	53798	250
DjjMU8qMaIqkb0AU	929644962	20577	59

Table 2: Encrypted_Employee

increased overhead for encryption/decryption [14]. As in [13, 8, 3, 15], we assume encryption to be performed at the tuple level. To provide the server with the ability to select a set of tuples to be returned in response to a query, we associate each encrypted tuple with a number of indexing attributes. An index can be associated with each attribute on which conditions need to be evaluated for query processing.

Each plaintext relation will be stored as a relation with one attribute representing the encrypted tuple and additional attributes representing the indexes. Each plaintext tuple $t(A_1, ..., A_n)$ is mapped onto a tuple $t'(E(t), I_1, ..., I_m)$ where $m \leq n$. The attribute $E(t)$ stores an encrypted string that corresponds to the entire plaintext tuple, and each I_i corresponds to the index over some A_j. The encryption function E is treated as a black box in our discussion. Any block cipher such as AES [2], DES [9] etc., can be used to encrypt the tuples. Table 2 illustrates an example of the corresponding encrypted/indexed relation Encrypted_Employee where Enc_tuple contains the encrypted triples, while I_{SSN}, I_{SALARY}, and I_{DNO} are indexes over attributes SSN, SALARY, and DNO respectively.

4 A Prefix-Preserving Encryption based Scheme

4.1 Transforming Interval-Matching into Prefix-Matching

In this section, we will transform interval-matching into prefix-matching. Prefix-matching has been used widely in databases and networks. The transformation is based on the fact that an arbitrary interval can be converted into a union of *prefix ranges*, where a prefix range is one that can be expressed by a prefix [18]. For example, the interval [32, 111], the 8-bit binary representation of which is [00100000, 01101111], can be represented by a set of prefixes $\{001*, 010*, 0110*\}$. Throughout this paper, the notation $*$ is used to denote an arbitrary suffix. To verify that a number is in the interval is equivalent to check that the number matches any of those prefixes in the set. For example, 37 (00100101 in binary) is in the interval as it matches prefix $001*$, while 128 (10000000 in binary) is not in the interval since it matches none of those three prefixes.

Let n denote the length of the binary representation of the data, and let p_n denote the number of prefixes needed to represent an interval. We have the following theorem on the upper bound of p_n.

Theorem 1 *For any interval* $[a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n]$ $(n \geq 2)$, $p_n \leq 2(n-1)$.

The proof of this theorem is presented in Appendix A. Note that for interval $[1, 2^n - 2]$, it can be easily verified that p_n is equal to $2(n-1)$. Therefore, the upper bound is tight.

Theorem 2 *For a given n, considering all possible intervals $[a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n]$, if we assume all the intervals appear with the same probability, i.e., all queries are equi-probable, the average number of p_n is equal to $\frac{(n-2)2^{2n-1}+(n+1)2^n+1}{2^{2n-1}+2^{n-1}}$, which is approximately equal to $n-2$, when n is large.*

The proof of this theorem is presented in Appendix B. From these two theorems we see that the upper bound of p_n is a linear function of n and the average number of p_n is approximately a linear function of n. This is a very nice feature.

In Figure 1 we present a recursive algorithm to generate the set of prefixes for a given interval $[a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n]$.

We have seen that matching an interval based on a set of prefix-matchings is both simple and efficient. Therefore prefix-preserving encryption algorithm can be used to efficiently

```
1. Starting from $k = 1$, find the most significant bit, numbered $k$, for which $a_k < b_k$.
2. If $k$ is not found, i.e., for all $1 \leq i \leq n$, $a_i = b_i$, then the interval can be denoted by prefix
   $a_1 a_2 \cdots a_n$. Return $a_1 a_2 \cdots a_n$.
3. If for all $k \leq i \leq n$, $a_i = 0$ and $b_i = 1$, then return $a_1 a_2 a_{k-1}*$ (return $*$ if $k = 1$).
4. Transform interval $[a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n]$ into $[a_1 \cdots a_{k-1} 0 a_{k+1} \cdots a_n, a_1 \cdots a_{k-1} 011 \cdots 1] \cup$
   $[a_1 \cdots a_{k-1} 100 \cdots 0, a_1 \cdots a_{k-1} 1 b_{k+1} \cdots b_n]$.
5. Run this algorithm with interval $[a_{k+1} \cdots a_n, 11 \cdots 1]$ as input, concatenate $a_1 \cdots a_{k-1}0$ before
   all the returned prefixes. Then run this algorithm with interval $[00 \cdots 0, b_{k+1} \cdots b_n]$ as input,
   concatenate $a_1 \cdots a_{k-1}1$ before all the returned prefixes. Return all the prefixes.
```

Figure 1: The Algorithm for transforming interval $[a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n]$ into prefixes

support interval-matching as a query condition while preserving the confidentiality of data and queries.

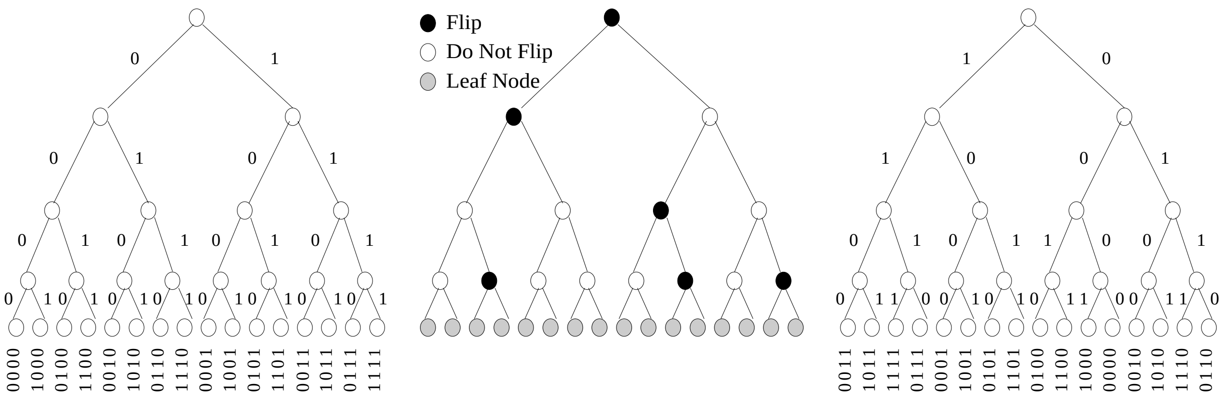
4.2 Prefix-Preserving Encryption

After transforming interval-matching into prefix-matching, we need a prefix-preserving encryption scheme to generate the index, so that the database system will be able to answer the queries based on encrypted data and queries. We apply an encryption scheme proposed by Xu *et al.* [20] for prefix-preserving IP address anonymization. First we introduce a formal definition of prefix-preserving encryption.

Definition 1 (Prefix-preserving encryption) ([20]) *We say that two n-bit numbers $a = a_1 a_2 \cdots a_n$ and $b = b_1 b_2 \cdots b_n$ share a k-bit prefix $(0 \leq k \leq n)$, if $a_1 a_2 \cdots a_k = b_1 b_2 \cdots b_k$, and $a_{k+1} \neq b_{k+1}$ when $k < n$. An encryption function E_p is defined as a one-to-one function from $\{0,1\}^n$ to $\{0,1\}^n$. An encryption function E_p is said to be prefix-preserving, if, given two numbers a and b that share a k-bit prefix, $E_p(a)$ and $E_p(b)$ also share a k-bit prefix.*

It is helpful to consider a geometric interpretation of prefix-preserving encryption [20]. If a plaintext can take any value of a n-bit number, the entire set of plaintexts can be represented by a complete binary tree of height n. This is called the *plaintext tree.* Each node in the plaintext tree (excluding the root node) corresponds to a bit position, indicated by the height of the node, and a bit value, indicated by the direction of the branch from its parent node. Figure 2(a) shows a plaintext tree (using 4-bit plaintexts for simplicity).

A prefix-preserving encryption function can be viewed as specifying a binary variable



(a) plaintext tree (b) encryption function (c) ciphertext tree

Figure 2: An example of prefix-preserving encryption

for each non-leaf node (including the root node) of the plaintext tree. This variable specifies whether the encryption function "flips" this bit or not. Applying the encryption function results in the rearrangement of the plaintext tree into a *ciphertext tree*. Figure 2(c) shows the ciphertext tree resulting from the encryption function shown in Figure 2(b). Note that an encryption function will, therefore, consist of $2^n - 1$ binary variables, where n is the length of a plaintext.

A general form of prefix-preserving encryption function is presented in [20]. Let f_i be a function $\{0,1\}^i$ to $\{0,1\}$, for $i = 1, 2, \cdots, n-1$ and f_0 is a constant function. Given a plaintext $a = a_1 a_2 \cdots a_n$, the ciphertext $a'_1 a'_2 \cdots a'_n$ will be computed by the algorithm given in Figure 3. According to Theorem 1 (canonical form theorem) in [20], the algorithm given in Figure 3 is a prefix-preserving encryption algorithm.

1. Compute a'_i as $a_i \oplus f_{i-1}(a_1 a_2 \cdots a_{i-1})$, where $\oplus$ stands for the exclusive-or operation, for $i = 1, 2, \cdots, n$.
2. Return $a'_1 a'_2 \cdots a'_n$.

Figure 3: Prefix-preserving encryption algorithm

In [20], the prefix-preserving encryption scheme is defined as instantiating functions f_i with cryptographically strong stream ciphers or block ciphers as follows:

$$f_i(a_1 a_2 \cdots a_i) := \mathcal{L}(\mathcal{R}(\mathcal{P}(a_1 a_2 \cdots a_i), \kappa)) \tag{1}$$

where $i = 0, 1, \cdots, n-1$ and $\mathcal{L}$ returns the "least significant bit". Here $\mathcal{R}$ is a pseudorandom function or a pseudorandom permutation (i.e., a block cipher). κ is the cryptographic key used in the pseudorandom function $\mathcal{R}$. Its length should follow the guideline specified for the pseudorandom function that is actually adopted.

The encryption function can be performed quickly as it only involves n symmetric key cryptographic operations, and these n operations can be done in parallel. A prefix expresses a prefix range, thus a prefix-matching query can be efficiently processed as a range query with a B^+-tree index structure. Therefore, high performance in the database system can be achieved. We will compare the performance of the prefix-preserving scheme with previously proposed schemes in Section 8.

5 Implementing Range Queries over Encrypted Relations

5.1 Mapping Range Query Conditions Map_{cond}

With the prefix-preserving encryption algorithm, denoted by E_p, we can translate specific query conditions in operations (such as selects and joins) to corresponding conditions over server-side representation. This translation function is called Map_{cond}. Since this paper is mainly focused on supporting range queries, we will only consider select operations in this section. We will discuss other relational operations in Section 9.

A query condition is a Boolean expression specified on relation attributes. It can be made up of a number of *clauses* of the form
<attribute> <comparison op> <value>,
or
<attribute> <comparison op> <attribute>
where <attribute> is the name of an attribute, <comparison op> is one of the operations $\{=, \leq, \geq\}$, and <value> is a constant value from the attribute domain. Clauses can be arbitrarily connected by the Boolean operators AND, OR, and NOT to form a general query condition. It has been discussed in [14] to translate a composite condition to the corresponding condition over server-side representation after each clause is translated. Hereafter we discuss how to translate a single clause.

attribute = value: Since the prefix-preserving encryption is a one-to-one mapping, the mapping is simply defined as follows:
$Map_{cond}(A_i = v) \Rightarrow E_p(A_i) = E_p(v)$

attribute $\leq$ value: A query condition $A_i \leq v$ is equivalent to an interval-matching of $f_{[v_{min},v]}(A_i)$, where v_{min} is the lower bound of the attribute domain. The interval $[v_{min}, v]$ can be converted into a union of prefix ranges, $\{P_1, P_2, \cdots, P_l\}$, with Algorithm 1. Therefore, interval-matching $f_{[v_{min},v]}(A_i)$ can be transformed to a set of prefix-matchings

$\{M_{P_1}(A_i), M_{P_2}(A_i), \cdots, M_{P_l}(A_i)\}$, where $M_{P_k}(A_i)$ denotes the boolean function, which returns true if and only if the value of A_i matches prefix P_k. Then the prefix-preserving encryption can be applied on the prefixes. Therefore, the mapping is defined as follows:
$Map_{cond}(A_i \leq v) \Rightarrow \{M_{E_p(P_1)}(E_p(A_i))$ OR $M_{E_p(P_2)}(E(A_i))$ OR $\cdots$ OR $M_{E_p(P_l)}(E_p(A_k))\}$.

attribute $\geq$ value: This condition is symmetric with the previous one. A query condition $A_i \geq v$ is equivalent to an interval-matching of $f_{[v,v_{max}]}(A_i)$, where v_{max} is the upper bound of the attribute domain. The interval-matching can be transformed to a set of prefix-matchings $\{M_{P_1}(A_i), M_{P_2}(A_i), \cdots, M_{P_l}(A_i)\}$ with Algorithm 1. Then the prefix-preserving encryption can be applied on the prefixes. Therefore, the mapping is defined as follows:
$Map_{cond}(A_i \geq v) \Rightarrow \{M_{E_p(P_1)}(E_p(A_i))$ OR $M_{E_p(P_2)}(E_p(A_i))$ OR $\cdots$ OR $M_{E_p(P_l)}(E_p(A_k))\}$.

5.2 Implementing Select Operation over Encrypted Relations

Select operation (σ): Consider a select operation $\sigma_C(R)$ on a relation R, where C is a condition specified on one or more of the attributes $A_1, A_2, \cdots, A_n$ of R. The operation can be rewritten as follows:
$\sigma_C(R) = D(\sigma_{Map_{cond}(C)}(E(R))$,
where $E(R)$ is the encrypted relational table (e.g., the Encrypted_Employee table presented in Table 2), and D is the corresponding decryption function of E. The operation $\sigma_{Map_{cond}(C)}(E(R))$ will be executed at the server. The results will be transmitted to the client. Then the client can get the query results by applying decryption function D.

6 Attack with a Set of Queries

In this section, we will discuss the security issues of the prefix-preserving scheme proposed in this paper and the schemes proposed in the literature, i.e., random one-to-one mapping and order-preserving. There are many possible attacks against these schemes [8, 3]. We are not going to be exhaustive on all the possible attacks, instead we will discuss a particular one.

An adversary may compromise the confidential information by gathering query predicate conditions. Sometimes it is reasonable for an adversary to assume that the index set against one attribute from each query may be derived from a single interval. In other words, each index set, though contains multiple indexes, represents only a single interval. Based on this assumption, the encryption mapping may be revealed partially, i.e., the adversary can figure out a coarse order of a set of encrypted indexes. This will be further explained in the rest of this section.

The feasibility of this attack is constrained by the ability of adversaries to collect enough queries. Furthermore, clients may not obey the assumption, i.e., they may not always submit a single interval-matching for each predicate against one index attribute in a query. This will complicate the attack as well.

To alleviate this problem, clients can specify different keys to generate indexes for different attributes, thus preventing an adversary from aggregating the information from different attributes. Also, clients can inject some noise into their queries to undermine the adversary's assumption. But the price paid is that the clients will receive some data that are not of interest. This compromises the purpose of the DAS model, since the clients still need certain database functionality to be able to filter out redundant results.

Since the order-preserving encryption preserves the order of the plaintext, it is trivial for the adversary to figure out the order of any set of encrypted indexes generated from the order-preserving scheme. In the remainder of this section we will analyze possible attacks against the random one-to-one mapping scheme and the prefix-preserving scheme.

6.1 Against the Random One-to-One Mapping Scheme

An adversary is assumed to be able to collect a set of queries. In each query there is a tuple of encrypted index sets. Based on our assumption, the encrypted indexes in a set should represent a single interval. Assume the size of the index domain is 2^n. If the adversary is able to collect all the $2^n - 1$ *two-index sets* which contain two consecutive indexes, then he/she will be able to figure out an order of all the indexes, but without knowing whether it is an ascending or descending order. If the adversary knows at least one plaintext/ciphertext pair, then he/she will be able to decrypt any encrypted index. For example, when $n = 2$, if the adversary is able to collect 3 index sets, $\{a, b\}, \{b, c\}, \{c, d\}$, the he/she will be able to figure out an order of the indexes, a, b, c, d, without knowing if it is an ascending or descending order.

An algorithm to collect two-index sets from a list of encrypted index sets is given in Figure 4.

6.2 Against the Prefix-Preserving Scheme

To better illustrate the attack against the prefix-preserving scheme, we introduce a definition as follows.

1. Discard one-index sets.
2. For any two sets A, B in the list of index sets, if none of $A \cap B$, $A \cap \overline{B}$, $\overline{A} \cap B$ is an empty set, then add these sets into the new list of index sets. (Note that any of these resulted sets still represents a single interval.)
3. If any new set is added, go to step 1. Otherwise, collect all two-index sets.

Figure 4: The Algorithm for attacking queries against the random one-to-one mapping

Definition 2 *Given two k-bit ($k \geq 2$) encrypted prefixes $a = a_1 a_2 \cdots a_k *$ and $b = b_1 b_2 \cdots b_k *$, if there exists an i, $1 \leq i \leq k-1$ such that $a_i \neq b_i$ and the range of a and b can be merged into a single interval, then we call the set of prefixes $\{a,\ b\}$ a* ***non-trivial two-prefix set with length*** *k. We call an encrypted two-prefix set $\{a_1 a_2 \cdots a_{k-1} a_k *, a_1 a_2 \cdots a_{k-1} \overline{a_k} *\}$ a* ***trivial two-prefix set****.*

It is easy to see that if an adversary has all $(2^{k-1} - 1)$ non-trivial two-prefix sets of length k, then he/she will be able to create an order for all k-bit encrypted prefixes without knowing if it is an ascending or descending order. If the adversary knows at least one plaintext/ciphertext pair, then he/she will be able to decrypt any k-bit prefix. For example, if an adversary has the following three non-trivial two-prefix sets of length 3, $\{a_1 a_2 a_3 *, a_1 \overline{a_2} b_3 *\}$, $\{a_1 \overline{a_2 \overline{b_3}} *, \overline{a_1} b_2 c_3 *\}$, $\{\overline{a_1} b_2 \overline{c_3} *, \overline{a_1} \overline{b_2} d_3 *\}$, then he/she will be able to figure out the following order for all 3-bit prefixes: $a_1 a_2 \overline{a_3}$, $a_1 a_2 a_3 *$, $a_1 \overline{a_2} b_3 *$, $a_1 \overline{a_2} \overline{b_3} *$, $\overline{a_1} b_2 c_3 *$, $\overline{a_1} b_2 \overline{c_3} *$, $\overline{a_1} \overline{b_2} d_3 *$, $\overline{a_1} \overline{b_2 d_3} *$ without knowing whether it is an ascending or descending order. If the plaintext of $a_1 a_2 a_3 *$ is known to be $001*$, the adversary will be able to decrypt any 3-bit prefix.

Before we further describe the attack, we give the following property about transforming an interval into prefixes.

Theorem 3 *Given the set of prefixes S transformed from the interval $[a, b]$, the longest prefixes in set S match either a or b.*

Proof: Let prefix $c = c_1 c_2 \cdots c_k *$ be one of the longest prefixes in S. Prefix $c_1 c_2 \cdots \overline{c_k} *$ is not in S, otherwise in S there should be prefix $c_1 c_2 \cdots c_{k-1} *$ instead of prefix c. For $\forall i < k$, prefix $c_1 c_2 \cdots c_i *$ is not in S, otherwise prefix c should not be in S. Since c is the longest prefix in S, any prefix which is longer than k-bit and matches $c_1 c_2 \cdots \overline{c_k} *$ is not in S. So we know $[c_1 c_2 \cdots \overline{c_k} 00 \cdots 0, c_1 c_2 \cdots \overline{c_k} 11 \cdots 1] \cap [a, b] = \emptyset$, while $[c_1 c_2 \cdots c_k 00 \cdots 0, c_1 c_2 \cdots c_k 11 \cdots 1] \subseteq [a, b]$.

If $c_k = 0$, then we have $c_1 c_2 \cdots c_{k-1} 100 \cdots 0 \notin [a, b]$, while $c_1 c_2 \cdots c_{k-1} 011 \cdots 1 \in [a, b]$.

1. Preprocess the encrypted prefix sets.

   - Mark the longest prefixes in each set as the edge prefix of that set.
   - For any prefix longer than k-bit, $a_1a_2\cdots a_ka_{k+1}\cdots a_l*$ ($l > k$), replace it by $a_1a_2\cdots a_k*$.
   - For any prefix shorter than k-bit, $a_1a_2\cdots a_l*$ ($l < k$), replace it by all the k-bit prefixes which share the l-bit prefix.
     (Note that after the preprocessing, the encrypted prefixes in a set still represent a single interval.)

2. Discard trivial two-prefix sets and one-prefix sets.
3. For any two sets A, B in the list of prefix sets, if none of $A \cap B$, $A \cap \overline{B}$, $\overline{A} \cap B$ is an empty set, then add these sets into the new list of prefix sets. Or if $A \subset B$, and A contains the edge prefix of B, then add $\overline{A} \cap B$ into the new list of prefix sets. (Note that any of these resulted sets still represents a single interval.)
4. If any new set is added, go to step 2. Otherwise, collect all non-trivial two-prefix sets.

Figure 5: The Algorithm for attacking queries against the prefix-preserving scheme

Since $c_1c_2\cdots c_{k-1}100\cdots0 = c_1c_2\cdots c_{k-1}011\cdots1+1$, we must have $b = c_1c_2\cdots c_{k-1}011\cdots1$, i.e., prefix c matches number b.

If $c_k = 1$, then we have $c_1c_2\cdots c_{k-1}011\cdots1 \notin [a,b]$, while $c_1c_2\cdots c_{k-1}100\cdots0 \in [a,b]$. Since $c_1c_2\cdots c_{k-1}100\cdots0 = c_1c_2\cdots c_{k-1}011\cdots1+1$, we must have $a = c_1c_2\cdots c_{k-1}100\cdots0$, i.e., prefix c matches number a. □

Note that after prefix-preserving encryption, the ciphertext of the longest prefix will still be the longest prefix in the encrypted set S.

An adversary is assumed to be able to collect a set of queries. In each query there is a tuple of encrypted prefix sets. Based on our assumption, the encrypted prefixes in a set should represent a single interval. An algorithm to collect non-trivial two-prefix sets of length k from a list of encrypted prefix sets is given in Figure 5.

Hereafter we give a simple example of this attack. Suppose an adversary wants to attack the 3-bit prefixes, and he/she has the following encrypted prefix sets from the queries. $A = \{a_1a_2a_3*$ (edge)$, a_1\overline{a_2}*, \overline{a_1}b_2*, \overline{a_1}\overline{b_2}b_3*$ (edge)$\}$, $B = \{a_1\overline{a_2}c_3*$ (edge)$, \overline{a_1}b_2*, \overline{a_1}\overline{b_2}b_3*$ (edge)$\}$, $C = \{\overline{a_1}b_2d_3e_4*$ (edge), $\overline{a_1}\overline{b_2}*\}$, where edge means that prefix contains the boundary of the interval represented by that prefix set. The adversary can get the following 3-bit prefix sets. $A' = \{a_1a_2a_3*\text{ (edge)}, a_1\overline{a_2}c_3*, a_1\overline{a_2c_3}*, \overline{a_1}b_2d_3*, \overline{a_1}b_2\overline{d_3}*, \overline{a_1}\overline{b_2}b_3*\text{ (edge)}\}$, $B' = \{a_1\overline{a_2}c_3*\text{ (edge)}, \overline{a_1}b_2d_3*, \overline{a_1}b_2\overline{d_3}*, \overline{a_1}\overline{b_2}b_3*\text{ (edge)}\}$, $C' = \{\overline{a_1}b_2d_3*\text{ (edge)}, \overline{a_1}\overline{b_2}b_3*, \overline{a_1}\overline{b_2b_3}*\}$. Then he/she will get the following non-trivial two-prefix sets: $A' \cap \overline{B'} = \{a_1a_2a_3*, a_1\overline{a_2c_3}*\}$, $A' \cap C' = \{\overline{a_1}b_2d_3*, \overline{a_1}\overline{b_2}b_3*\}$, $B' \cap \overline{C'} = \{a_1\overline{a_2}c_3*, \overline{a_1}b_2\overline{d_3}*\}$. Finally the adversary will be able to figure out the following order of the 3-bit prefixes: $a_1a_2\overline{a_3}*$, $a_1a_2a_3*$, $a_1\overline{a_2c_3}*$, $a_1\overline{a_2}c_3*$, $\overline{a_1}b_2\overline{d_3}$, $\overline{a_1}b_2d_3*$, $\overline{a_1}\overline{b_2}b_3*$, $\overline{a_1}\overline{b_2b_3}*$, without knowing whether it is an ascending or descending order.

7 Additional Security Analysis for Prefix-Preserving Encryption

In this section we will have more discussion about the security of the prefix-preserving encryption scheme. It has been proved that with the instantiating functions as (1) the prefix-preserving encryption scheme is indistinguishable from a *random prefix-preserving function*, a function uniformly chosen from the set of all prefix-preserving functions when the adversaries are assumed to be computationally bounded. This is elaborated in [20]. Moreover, as mentioned in Section 4.2, when plaintexts can take any value of a n-bit number, the prefix-preserving encryption function consists of $2^n - 1$ binary variables. Therefore, we have a key of 2^{2^n-1} possibilities. For example, when n is only 16, the number of possible keys is 2^{65535}. Therefore, the key κ in (1) can be sufficiently long such that it is impractical for adversaries to try each possible key to compromise the prefix-preserving scheme.

In the remainder of this section, we discuss another possible way in which the prefix-preserving scheme may be attacked. An adversary is assumed to have compromised (gain full knowledge to) certain number of $\langle$plaintext, ciphertext$\rangle$ pairs through means other than compromising the key, i.e., the known plaintext attack model [19]. Then he/she will be able to infer information from other ciphertexts by prefix-matching, because the encryption is prefix-preserving. For example, if an adversary knows $\langle$plaintext, ciphertext$\rangle$ pair $\langle a_1a_2\cdots a_n, a'_1a'_2\cdots a'_n\rangle$, then given another ciphertext $a'_1a'_2\cdots a'_{k-1}\overline{a'_k}b_{k+1}\cdots b'_n$, he/she knows the k-bit prefix of the plaintext should be $a_1a_2\cdots a_{k-1}\overline{a_k}$. Note that if an adversary knows one $\langle$plaintext, ciphertext$\rangle$ pair $\langle a_1a_2\cdots a_n, a'_1a'_2\cdots a'_n\rangle$, then he/she should also know the $\langle$plaintext, ciphtertext$\rangle$ pair $\langle a_1a_2\cdots\overline{a_n}, a'_1a'_2\cdots\overline{a'_n}\rangle$. Therefore, an adversary always knows an even number of $\langle$plaintext, ciphertext$\rangle$ pairs.

Suppose an adversary knows 2 pairs of $\langle$plaintext, ciphertext$\rangle$. Given a random ciphertext, let $A(n)$ denote the average length of the prefix that can be inferred by prefix-matching, where n is the length of the binary representation of the data. The probability that the k-bit prefix of the plaintext can be inferred is $\frac{1}{2^k}$, for $1 \le k \le n-1$, while for $k = n$, the probability is $\frac{2}{2^n}$. Therefore, $A(n) = \sum_{i=1}^{n-1} \frac{i}{2^i} + \frac{2n}{2^n} = \sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}} < 2$. In other words, on the average an adversary can infer no more than 2 bits from a random ciphertext, if he/she knowns 2 pairs of $\langle$plaintext, ciphertext$\rangle$.

We also analyze the situation that an adversary knows $2k$ ($k > 1$) pairs of $\langle$plaintext, ciphertext$\rangle$. This is presented in Appendix C. In summary, when $n \to \infty$, given a ciphertext, the average length of the prefix that can be inferred is bounded by $\log_2 k + 2$ based on numerical results. So the prefix information an adversary can obtain by comparing a ciphtertext against a few pairs of $\langle$plaintext, ciphertext$\rangle$ is limited. Therefore, we claim that the prefix-preserving scheme is secure even if a few pairs of $\langle$plaintext, ciphertext$\rangle$ are known by an adversary. To make the system even more secure, the data owner can specify different keys to generate indexes for different attributes, thus preventing an adversary from aggregating the information from different attributes.

8 Performance Comparison

8.1 Communication Cost

With the prefix-preserving scheme, the total length of the indexes for an interval-matching query condition is less than $2n(n-1)$ bit. With random one-to-one mapping, the total length is $l \cdot n$, where l is the length of the interval. So when l is larger than $2(n-1)$, the prefix-preserving scheme is more efficient. If we assume all the intervals appear with the same probability, the average length of the interval is $\frac{\sum_{i=1}^{2^n} \frac{i(i+1)}{2}}{\sum_{i=1}^{2^n} i} = \frac{2^{3n-1}+3\cdot 2^{2n-1}+2^n}{3(2^{2n-1}+2n-1)}$, which is approximately equal to $2^n/3$, when n is large. Therefore, the average number of bits of the indexes is about $n \cdot 2^n/3$, which is much greater than $2n(n-1)$, when n is large.

8.2 Client Side Cost

During the encryption of the database, it costs more to use the prefix-preserving scheme to compute the index for the records. Since the length of the index attribute n is most likely smaller than the block size of a typical block cipher, to compute one index, the prefix-preserving encryption will require $(n-1)$ block cipher encryptions. In contrast, the random one-to-one mapping will require only 1 encryption. Similarly, to encrypt an exact matching query condition, it costs more with prefix-preserving encryption. But to encrypt an interval-matching query condition, with prefix-preserving, at most $2(n-1)^2$ encryptions are needed. With random one-to-one mapping, the number of encryptions needed is equal to the length of the interval l. So when l is larger than $2(n-1)^2$, the prefix-preserving scheme is more efficient. If we assume all intervals appear with same probability, then the average length of the interval is about $2^n/3$, which is much larger than $2(n-1)^2$, when n is large.

8.3 Server Side Cost

As for the server side cost, we will be mainly concerned about the cost of disk accesses for executing an interval-matching query, since in most cases it is the bottleneck. To estimate the cost of disk accesses, we must know the number of the records (r), and the number of blocks (b) (or close estimates of them). Also, we need to know the number of levels (h) of B$^+$-tree, which is the typical database storage structure for indexes. Another important parameter is the number of distinct values (d) of an attribute and its selectivity (sl), which is the fraction of records satisfying an exact-matching condition on the attribute. The estimation of the selection cardinality ($s = sl \times r$) of an attribute is the average number of records that will satisfy an exact-matching selection condition on that attribute. By making an assumption that the d distinct values are uniformly distributed among the records, we estimate $sl = 1/d$ and so $s = r/d$.

Without any index structure, to do a sequential scan of the whole database table, the cost of disk accesses is $b \cdot t_s$, where t_s is the time needed for a sequential disk block access. In the random one-to-one mapping scheme, with a B$^+$-tree index structure the number of disk accesses needed for retrieving the indexes is $l \cdot h$, where l is the length of the interval, and the number of disk accesses needed for retrieving the actual records is $l \cdot f \cdot sl \cdot b$, where f is the percentage of the values in the interval that actually exist in the table. Therefore, the total timing cost of disk accesses is $l(h + f \cdot sl \cdot b) \cdot t_r$, where t_r is the time needed for a random disk block access. As mentioned in 4.2, a prefix-matching query can be processed as a range query. In the prefix-preserving scheme, with a B$^+$-tree index structure the number of disk accesses needed for retrieving the indexes is less than $2(n-1)(h-1) + l$, and the number of disk accesses for accessing the actual records is $l \cdot f \cdot sl \cdot b$. So the timing cost of disk accesses is less than $(2(n-1)(h-1) + l + l \cdot f \cdot sl \cdot b)t_r$. Therefore, when $l > 2(n-1)$, the prefix-preserving scheme is more efficient then the random one-to-one mapping scheme, if sequential scan is not needed. A typical value of n could be 32. Then when $l > 62$, the prefix-preserving scheme is more efficient. A typical disk block size is 32K bits. A typical

	Average Communication Cost (length of indexes)	Average Client Side Cost (number of encryptions)	Server Side Cost (timing of disk accesses)
prefix-preserving	$\leq 2n(n-1)$	$\leq 2(n-1)^2$	$\leq (2(n-1)(h-1)+l+l \cdot f \cdot sl \cdot b)t_r$
random mapping	$n \cdot 2^n/3$	$2^n/3$	$l(h+f \cdot sl \cdot b)t_r$

Table 3: Performance comparison of prefix-preserving and random one-to-one mapping for supporting range queries

size of a pointer to a disk block is 32 bits. Assume the number of records r in the database is 1M, i.e., 2^{20}. Then the order of the B$^+$-tree should be $32\text{K}/(32+32) = 2^9$, and the height h should be $\log_{2^9} 2^{20} = 3$. Figure 6 shows the number of disk accesses with random one-to-one mapping/prefix-preserving, assuming f to be 100%, $sl = 1/r = 2^{-20}$, and the total number of disk blocks in the database b is 512K. Note that, assuming $t_r = 64 \cdot t_s$, for the random one-to-one mapping scheme, it will be more efficient to do a sequential scan of the whole database table when $l > 3177$. With the prefix preserving scheme, it might be more efficient to do a sequential scan when $l > 1260$.

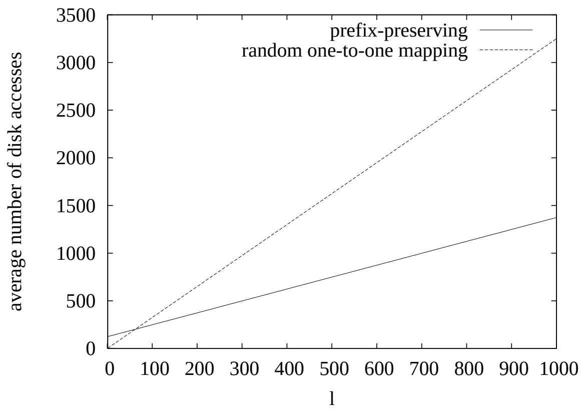


Figure 6: average number of disk accesses by varying l

In summary, we present the communication cost, client side and server side computation cost for supporting an interval-matching query with random one-to-one mapping/prefix-preserving in Table 3.

9 Discussion

9.1 Mapping Conditions other than Exact-Matching/Interval-Matching

attribute1 = attribute2: Such a condition might arise in a join. The two attributes can be from two different tables or from two instances of the same table. The condition can also arise in a select, and the two attributes can be from the same table. Since the prefix-preserving encryption is a one-to-one mapping, the following is the translation:
$Map_{cond}(A_i = A_j) \Rightarrow E_p(A_i) = E_p(A_j)$

attribute1 $\leq$ attribute2, attribute1 $\geq$ attribute2: Such conditions might arise in a join. These conditions can not be supported by our scheme conveniently. To support these conditions, either efficiency or security has to be sacrificed. If the available bandwidth between the client and the server is sufficient, we can transfer all the involved tables from server to client. Then client can decrypt the data and apply the query operation. If the server is trusted, the server can decrypt involved tables and run the query on the decrypted data. Since the most common use of join operation involves conditions with equality comparisons only, we argue that this disadvantage is outweighed by the advantages of our scheme in efficiently and securely supporting other more commonly used query conditions.

9.2 Implementing Relational Operations other than Selections

In this section we describe how relational operators other than selection, such as join, set difference and project operations, can be implemented in the proposed prefix-preserving scheme.

Since the prefix-preserving encryption is a one-to-one mapping, it is straightforward to implement project, union, intersection, minus, duplicate-elimination, and division operations.

Join operation ($\bowtie$): Consider a join operation $R \bowtie_C S$. Our scheme can efficiently support equijoin, in which the join condition C is an equality comparison, since the prefix-preserving is a one-to-one mapping. For more general conditions, as mentioned in previous section, either efficiency or security has to be sacrificed.

Aggregate functions and grouping operation (γ): We can define an AGGREGATE FUNCTION operation, using symbol γ, to specify these types of requests as follows:
$_{\langle\text{grouping attributes}\rangle}\gamma_{\langle\text{function list}\rangle}(R)$
where <grouping attributes> is a list of attributes of the relation specified in R, and

<function list> is a list of (<function> <attribute>) pairs. In each such pair, <function> is one of the allowed functions, such as SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT, and <attribute> is an attribute of the relation specified by R. Our scheme can efficiently support function COUNT, since the prefix-preserving encryption is a one-to-one mapping. For MAXIMUM/MINIMUM functions, in most cases, user should have some idea about the range of the result. So the user can send an interval-based selection query to the server, and decrypt all the results returned by the server, then apply the MAXIMUM/MINIMUM functions. For SUM/AVERAGE functions, an efficient solution can be to let the server keep encrypted statistics information, i.e., sum and average. Every time the client updates the database, the client will get the statistics information from the server, recompute the values, and update the server.

10 Conclusions

This paper discusses concerns about protecting sensitive information of data and queries from adversaries in the DAS model. Data and queries need to be encrypted, while the database service provider should be able to efficiently answer queries based on encrypted data and queries. Several approaches are studied in this paper, random one-to-one mapping, prefix-preserving and order-preserving. Possible attacks against these approaches and the performance of these approaches are investigated. The prefix-preserving scheme is more efficient than random one-to-one mapping for supporting range queries. In terms of communication cost, the length of indexes for an interval with prefix-preserving is less than $2n(n-1)$ bits, while with random one-to-one mapping the average is $n \cdot 2^n/3$. In terms of client side cost, the number of encryptions needed with prefix-preserving is $2(n-1)^2$, while with random one-to-one mapping the average is $2^n/3$. In terms of server side cost, the average number of disk accesses with prefix-preserving is smaller than the random one-to-one mapping scheme, when the length of the interval is larger than 2(n-1). However, the prefix-preserving scheme is less secure than the random one-to-one mapping scheme, because of the constraint of prefix-preserving. For example, with the prefix-preserving encryption a coarse ordering of the encrypted data can be determined by a grouping based on a k-bit prefix, but not with a random one-to-one mapping.

References

[1] Martin Abadi, Joan Feigenbaum, and Joe Kilian. On hiding information from an oracle. In *Proceedings of the 19th ACM Annual Symposium on Theory of Computing*, pages 195–203, May 1987.

[2] AES. Advanced encryption standard. FIPS 197, Computer Security Resource Center, National Institute of Standards and Technology, 2001.

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

[4] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Proceedings of Advances in Cryptology – Proceedings of EUROCRYPT'00*, pages 431–444, May 2000.

[5] Ernest F. Brickell, David Chaum, Ivan Damgard, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In *Proceedings of CRYPTO'87*, pages 156–166, 1987.

[6] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 41–50, October 1995.

[7] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *Proceedings of Advances in Cryptology – EUROCRYPT'00*, pages 122–138, May 2000.

[8] Ernesto Damiani, S. De Capitani Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proceedings of CCS'03*, pages 93–102, 2003.

[9] DES. Data encryption standard. FIPS PUB 46, Federal Information Processing Standards Publication, 1977.

[10] Joan Feigenbaum. Encrypting problem instances, or ..., can you take advantage of someone without having to trust him? In *Proceedings of CRYPTO'85*, pages 477–488, June 1986.

[11] Zvi Galil, Stuart Haber, and Mordechai Yung. A private interactive test of a boolean predicate and minimum-knowledge of public-key cryptosystems. In *Proceedings of the 26th IEEE Annual Symposium on Foundations of Computer Science*, pages 360–371, 1985.

[12] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th ACM Annual Symposium on Theory of Computing*, pages 291–304, 1985.

[13] Hakan Hacigumus, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceesings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 216–227, 2002.

[14] Hakan Hacigumus, Bala Iyer, and Sharad Mehrotra. Providing database as a service. In *Proceesings of ICDE'02*, pages 29–38, 2002.

[15] Bijit Hore, Shared Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *Proceedings of VLDB'04*, pages 720–731, 2004.

[16] Wenbo Mao. Guaranteed correct sharing of integer factorization with off-line share-holders. In *Proceedings of Public Key Cryptography'98*, pages 27–42, 1998.

[17] S. C. Gultekin Ozsoyoglu, David Singer, and Sun S. Chung. Anti-tamper databases: Querying encrypted databases. In *Proceedings of the 17th Annunal IFIP WG11.3 Working Conference on Databse and Application Security*, August 2003.

[18] V. Srinivasan, George Varghese, Subash Suri, and Marcel Waldvogel. Fast and scalable layer four switching. In *Proceedings of ACM SIGCOMM'98*, pages 191–202, September 1998.

[19] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, 2002.

[20] Jun Xu, Jinliang Fan, Mostafa H. Ammar, and Sue B. Moon. Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme. In *Proceedings of the 10th IEEE International Conference on Network Protocols*, pages 280–289, November 2002.

Appendix

A Proof of Theorem 1

In this appendix, we offer a proof of Theorem 1 (introduced in Section 4.1), which gives the upper bound of the number of prefixes needed to represent an interval. We will use the same notations as in Section 4.1.

Lemma 1 *For any interval* $[0, a_1a_2\cdots a_n]$, $p_n \le n$.

Proof: We prove it by induction on n. The conclusion trivially holds for $n = 1$. Suppose the conclusion also holds for $n = k$.

We now prove the lemma for $n = k+1$. If a_1 is equal to 0, then according to induction hypothesis, we have $p_n \le k$. If for all $1 \le i \le k+1$, $a_i = 1$, then the interval can be represented by prefix $*$, i.e., $p_n = 1$. Otherwise, the interval $[0, a_1a_2\cdots a_{k+1}]$ can be represented by $[0, 011\cdots1] \cup [100\cdots0, 1a_2\cdots a_{k+1}]$. $[0, 011\cdots1]$ can be represented by prefix $0*$. According to induction hypothesis, we need at most k prefixes to represent interval $[100\cdots0, 1a_2\cdots a_{k+1}]$. Hence for $n = k+1$, we have $p_n \le k+1$. □

Lemma 2 *For any interval* $[a_1a_2\cdots a_n, 11\cdots1]$, $p_n \le n$.

The proof is omitted here, because it is similar to the proof of Lemma 1.

Theorem 1 *For any interval* $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$ $(n \ge 2)$, $p_n \le 2(n-1)$.

Proof: We prove it by induction on n.

For $n = 2$, if $a_1a_2 = 00$ or $b_1b_2 = 11$, then according to Lemma 1 and Lemma 2, we have $p_n \le 2$. If $a_1a_2 = b_1b_2 = 01$ or 10, then $p_n = 1$. If $a_1a_2 = 01$ and $b_1b_2 = 10$, then $p_n = 2$. So the conclusion holds for $n = 2$.

Suppose the conclusion also holds for $n = k$.

We now prove the theorem for $n = k+1$. If $a_1 = b_1$, then according to induction hypothesis, we have

$p_n \leq 2(k-1)$. Otherwise, we have $a_1 = 0$ and $b_1 = 1$. If for all $1 \leq i \leq k+1$, $a_i = 0$ and $b_i = 1$, then the interval can be represented by prefix $*$, i.e., $p_n = 1$. The interval $[a_1a_2\cdots a_{k+1}, b_1b_2\cdots b_{k+1}]$ can be represented by $[0a_2\cdots a_{k+1}, 011\cdots 1] \cup [100\cdots 0, 1b_2\cdots b_{k+1}]$. According to Lemma 2, we need at most k prefixes to represent interval $[0a_2\cdots a_{k+1}, 011\cdots 1]$, and according to Lemma 1, we need at most k prefixes to represent interval $[100\cdots 0, 1b_2\cdots b_{k+1}]$. Hence for $n = k+1$, we have $p_n \leq 2k$. □

B Proof of Theorem 2

In order to compute the average number of p_n presented in Section 4.1, we present Lemma 3 and Lemma 4 first.

Lemma 3 *For a given n, considering all possible intervals $[0, a_1a_2\cdots a_n]$, the average number of p_n, denoted by $\overline{p_n}$, is equal to $\frac{n2^{n-1}+1}{2^n}$, if we assume all the intervals appear with the same probability.*

Proof: Let B_n denote the sum of the number of prefixes needed for all intervals. Then we have $\overline{p_n} = \frac{B_n}{2^n}$. We compute B_n by classifying all the intervals into 2 different cases:

Case 1: $a_1 = 0$, obviously the sum of the number of prefixes needed for all the intervals in this case is B_{n-1}.

Case 2: $a_1 = 1$, the interval $[0, a_1a_2\cdots a_n]$ can be represented by $[0, 011\cdots 1] \cup [100\cdots 0, 1a_2\cdots a_n]$. So the sum of prefixes needed for all the intervals in this case is $B_{n-1} + 2^{n-1} - 1$. The present of minus 1 is due to the fact that interval $[0, 11\cdots 1]$ can be denoted by one prefix $(*)$ instead of two prefixes.

So we have $B_n = 2B_{n-1} + 2^{n-1} - 1$. And initially we have $B_1 = 2$. Thus

$$\begin{aligned} B_n &= 2^{n-1}B_1 + \sum_{k=1}^{n-1} 2^{k-1}(2^{n-k} - 1) \\ &= 2^n + (n-1)2^{n-1} - \sum_{k=0}^{n-2} 2^k \\ &= (n+1)2^{n-1} - (2^{n-1} - 1) \\ &= n \cdot 2^{n-1} + 1 \end{aligned}$$

Lemma 4 *For a given n, considering all possible intervals $[a_1a_2\cdots a_n, 11\cdots 1]$, the average number of p_n, denoted by $\overline{p_n}$, is equal to $\frac{n2^{n-1}+1}{2^n}$, if we assume all the intervals appear with the same probability.*

Proof: Let C_n denote the sum of the number of prefixes needed for all intervals. Then we have $\overline{p_n} = \frac{C_n}{2^n}$. Same as in Lemma 3, we have $C_n = n2^{n-1} + 1$. The computation of C_n is omitted here since it is very similar to the computation of B_n.

Theorem 2 *For a given n, considering all possible intervals $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$, the average number of p_n, denoted by $\overline{p_n}$, is equal to $\frac{(n-2)2^{2n-1}+(n+1)2^n+1}{2^{2n-1}+2^{n-1}}$, if we assume all the intervals appear with the same probability.*

Proof: Let A_n denote the sum of the number of prefixes needed for all intervals. Then we have $\overline{p_n} = \frac{A_n}{\sum_{i=1}^{2^n} i} = \frac{A_n}{2^{2n-1}+2^{n-1}}$.

We compute A_n by classifying all the intervals into 2 different cases:

Case 1: $a_1 = b_1$, obviously the sum of the number of prefixes needed for all the intervals in this case is $2A_{n-1}$.

Case 2: $a_1 < b_1$, the interval $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$ can be represented by $[0a_2\cdots a_n, 011\cdots 1] \cup [100\cdots 0, 1b_2\cdots b_n]$. So the sum of the number of prefixes needed for all the intervals in this case is $2^{n-1}(B_{n-1} + C_{n-1}) - 1$.

The present of minus 1 here is due to the fact that interval $[0, 11\cdots 1]$ can be denoted by one prefix $(*)$ instead of two prefixes.
So we have $A_n = 2A_{n-1} + (n-1)2^{2n-2} + 2^n - 1$. And initially we have $A_1 = 3$. Thus

$$\begin{aligned}
A_n &= 2^{n-1}A_1 + \sum_{k=1}^{n-1} 2^{k-1}((n-k)2^{2n-2k} + 2^{n-k+1} - 1) \\
&= 3 \cdot 2^{n-1} + n2^{2n}\sum_{k=1}^{n-1} 2^{-(k+1)} - 2^{2n}\sum_{k=1}^{n-1} k2^{-(k+1)} \\
&\quad +(n-1)2^n - \sum_{k=1}^{n-1} 2^{k-1} \\
&= 3 \cdot 2^{n-1} + n2^{2n-1}(1 - 2^{-(n-1)}) - 2^{2n}\sum_{k=1}^{n-1} k2^{-(k+1)} \\
&\quad +(n-1)2^n - (2^{n-1} - 1) \\
&= n2^{2n-1} + 1 - 2^{2n}\sum_{k=1}^{n-1} k2^{-(k+1)} \\
&= n2^{2n-1} + 1 - (2^{2n} - n2^{n+1} + (n-1)2^n) \\
&= (n-2)2^{2n-1} + (n+1)2^n + 1
\end{aligned}$$

C Known plaintext attack against the prefix-preserving scheme

In this appendix, we analyze the information an adversary can obtain by comparing known ⟨plaintext, ciphertext⟩ pairs to a ciphertext. Hereafter we will assume the length of the binary representation of the data, denoted by n, is very long, i.e., $n \to \infty$. Suppose that an adversary obtains $2k$ pairs of ⟨plaintext, ciphertext⟩ randomly, and the average length of the prefix he/she can obtain from a random ciphertext is A_k. Then A_k can be computed as follows.

In the first step, the adversary will compare the first bit of the ciphertext with all the $2k$ pairs of ⟨plaintext, ciphertext⟩. If, among them, $2l$ pairs of ⟨plaintext, ciphertext⟩ match the first bit of the ciphertext, then the other $2(k-l)$ pairs are not useful for further deriving prefix information. Therefore, the total prefix information the adversary can obtain is the first bit plus the prefix information he/she may obtain with $2l$ pairs of ⟨plaintext, ciphertext⟩, which is equal to $1 + A_l$. Since the possibility for $2l$ pairs out of $2k$ pairs of ⟨plaintext, ciphertext⟩ to match the first bit of a ciphertext is $(1/2)^k C_k^l$, the average length of the prefix an adversary can obtain is $A_k = \sum_{l=0}^{k}(1/2)^k C_k^l(1 + A_l)$.

It is obvious that A_0 is equal to 0. The remaining values of A_k can be computed inductively. For example, $A_1 = 1 + \frac{1}{2}A_1$, thus $A_1 = 2$, which is consistent with the result presented in Section 7. Table 4 shows A_k for several different values of k. Figure 7 presents the curve of A_k by varying k, which is bounded by $\log_2 k + 2$. Therefore, the prefix information that can be revealed by a few pairs of ⟨plaintext, ciphertext⟩ is limited.

k	1	2	4	8	16
A_k	2	2.666667	3.504762	4.421077	5.377378
k	32	64	128	256	512
A_k	6.355176	7.343990	8.338377	9.335558	10.334156

Table 4: A_k by varying k

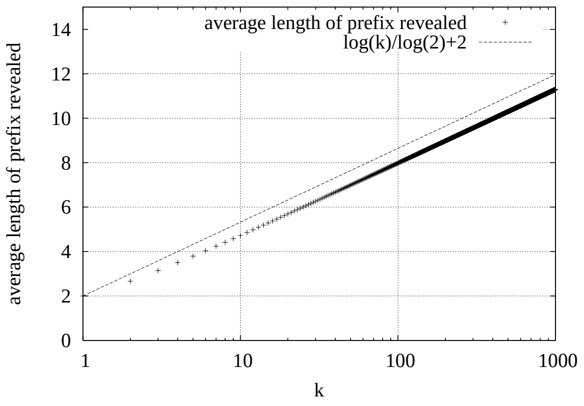


Figure 7: A_k by varying k