

Potential Security Holes in Hacıgümüş’ Scheme of Executing SQL over Encrypted Data

Kenny C.K. Fong

Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
kcfong@fas.harvard.edu

Abstract

Hacıgümüş, Iyer, Li and Mehrotra proposed the first scheme of executing SQL over encrypted data in the database-service-provider model. However, they did not address the security strength of the scheme over different attack models. While Hacıgümüş’ scheme looks secure in a general setting, this paper attempts to explore any plausible attacks against this scheme in specific environments. We present five potential security holes in the scheme, and propose solutions to several of them.

1 Introduction

Rapid advances in computer networking and Internet technologies have fueled the emergence of the “database as a service” model [12] for enterprise computing. The database-service-provider model provides users power to create, store, modify, and retrieve data from anywhere in the world, as long as they have access to the Internet. From the business perspective, this model allows organizations to leverage hardware and software solutions provided by the service providers, without having to develop them on their own. Perhaps more importantly, it provides a way for organizations to share the expertise of database professionals, thereby cutting the human cost of managing a complex information infrastructure, which is important both for industrial and academic organizations.

From the technical perspective, the model poses many significant challenges, the foremost of which being the issue of *data privacy* and *security*. In the database-service-provider model, user data reside on the premises of the database-service provider. Most corporations view their data as a very valuable asset. The service provider needs to provide sufficient security measures to guard data privacy. At least two data-privacy challenges arise. The first challenge is: how do service providers protect cus-

tomers data against eavesdropping and modification from outsiders? Encryption of stored data is the straightforward solution here. From both the cryptographic [13] and performance [12] points of view, encrypting by row (i.e., by tuple) is found preferable to encrypting by field (i.e., by attribute).

The second challenge is that of “total” data privacy, which is more complex since it includes protection from the database provider. One of the requirements is that encrypted data may not be decrypted at the provider site. A straightforward approach is to transmit the entire encrypted relation from the server (at the provider site) to the client, decrypt all the tuples in the relation, and execute the query at the client. But this approach mitigates almost every advantage of the service-provider model, since now primary data processing has to occur on client machines. Moreover, this approach is too costly in terms of bandwidth. It now becomes clear that how to perform database operations such as selections and joins over encrypted data stored at the provider site is an important and interesting research problem.

Hacıgümüş, Iyer, Li and Mehrotra [11] proposed the first scheme of executing SQL queries over encrypted data in the database-service-provider model. In this scheme, the encrypted database is augmented with additional information allowing a certain amount of query processing to occur at the server without jeopardizing data privacy. Based on the additional information stored, one can develop techniques to split an original query over unencrypted tuples into (1) a corresponding query over encrypted tuples to run on the server, and (2) a client query for post-processing results of the server query.

This scheme seems to do what it is supposed to do at first glance, however the additional information stored at the server may be a potential source of information leakage. Hacıgümüş et al. claimed in their paper that this scheme provides “total” data privacy (i.e. secu-

urity against untrusted servers) in the database-service-provider model, but they neither addressed this issue formally nor quantified the security strength of this scheme. In particular, they failed to notice that not decrypting the encrypted data at the database-service-provider site is a necessary but insufficient security measure.

While we believe that Hacigümüş’ scheme is secure in a general setting, this paper attempts to explore any plausible attacks against this scheme in specific environments. Our contribution is to present five potential security holes in Hacigümüş’ scheme, and to propose solutions to several of them.

The rest of this paper is organized as follows. We first discuss related work in Section 2 and give a brief overview of Hacigümüş’ scheme in Section 3. The five potential security holes in Hacigümüş’ scheme are presented in Sections 4 to 8. In Section 4, we show that Hacigümüş’ scheme is not semantically secure. In Section 5, we discuss any negative effects of the hash functions used in Hacigümüş’ scheme when their domains are small. In Section 6, we show that Hacigümüş’ scheme does not perfectly hide the query type. In Section 7, we point out that Hacigümüş’ scheme does not perform decryptions by field. In Section 8, we show that Hacigümüş’ scheme is incapable of detecting dishonest servers. Finally, we conclude the paper and suggest future work in Section 9.

2 Related Work

Various theoretical problems concerning computation with encrypted data and searching on encrypted data have appeared in the literature. Yao [19] posed the problem of which distributed computation problems have *secure multiparty protocols*. That is, when is it possible for equally-powerful parties P_i , $1 \leq i \leq n$, each holding a private input x_i , to cooperate in the computation of $y = f(x_1, \dots, x_n)$ in such a way that each P_i learns y and no P_i learns anything about x_j , $j \neq i$, except what is implied by y and x_i ? Several researchers have studied secure multiparty protocols in both the computationally-bounded and information-theoretical settings [8, 3].

Abadi et al. [1] posed the problem of which functions have *instance-hiding schemes*. That is, one party, say P_0 , has a private input x , but lacks the computational resources to compute f . Party P_1 has the power to compute f but cannot guarantee the privacy of inputs it receives. For which functions f can P_0 cooperate with P_1 to obtain $f(x)$ without revealing x ?

Chor et al. [4] posed the problem of *private information retrieval* (PIR). PIR schemes allow a user to retrieve information from a database while maintaining the privacy of the queries from the database. More formally, we view the data as an n -bit string x from which the

user wishes to obtain the bit x_i while keeping the index i private from the database. Chor et al. also proposed such a scheme under the assumption that the data are replicated in several sites which are assumed not to communicate with one another. Kushilevitz and Ostrovsky [14] proposed a PIR scheme that does not require replication of the data.

Rivest et al. [15] proposed the idea of *privacy homomorphisms* (PH), which are encryption functions that allow direct computation (i.e., performing arithmetic) on encrypted data. Song, Wagner and Perrig [18] provided the first practical solution for searching on encrypted data. They defined the problem of searching on encrypted data as follows. Suppose Alice has a set of documents D_1, \dots, D_n and stores them on an untrusted server Bob. Every document D_i is divided up into “words”. For any word W , Alice wants Bob to find all the documents D_i that contain W , without revealing W to Bob. The scheme proposed by Song et al. that solves this problem is provably-secure and instance-hiding (i.e., the word W is never revealed to Bob).

Although there seems to be a lot of schemes for computation with encrypted data or searching on encrypted data in previous work, the functionalities provided by them are very limited and insufficient in executing complex SQL queries over encrypted data. Hacigümüş’ scheme is the first scheme in the literature that supports execution of SQL queries over encrypted data in the database-service-provider model.

3 Hacigümüş’ Scheme

In this section, we give a brief description of Hacigümüş’ scheme. For details of this scheme, consult the paper by Hacigümüş et al. [11]. First, let us discuss how the encrypted data are stored at the server.

3.1 Storage Model and Encryption

For each relation $R(A_1, A_2, \dots, A_n)$, we store on the server an encrypted relation:

$$R^S(etuple, A_1^S, A_2^S, \dots, A_n^S)$$

where the attribute *etuple* stores an encrypted string that corresponds to a tuple in relation R (we explain how *etuple* is defined in Section 3.1.4). Each attribute A_i^S , called the *index* of the attribute A_i in R , stores the image of a function at the value stored at A_i , which will be used for query processing at the server (such a function, called a *mapping function*, is described in Section 3.1.3). For example, consider a relation *emp* below that stores information about employees.

eid	ename	salary	addr	did
23	Tom	70K	Maple	40
860	Mary	60K	Main	80
320	John	50K	River	50
875	Jerry	55K	Hopewell	110

The *emp* relation is mapped to a corresponding encrypted relation at the server:

$$emp^S(etuple, eid^S, ename^S, salary^S, addr^S, did^S)$$

It is only necessary to create an index for attributes involved in search and join predicates. However, without loss of generality, we assume that an index is created over each attribute of the relation.

3.1.1 Partition Functions

We explain what is stored in attribute A_i^S of R^S for each attribute A_i of R . For this purpose, we need to develop some useful notations. We first map the domain \mathcal{D}_i of values of attribute $R.A_i$ into partitions $\{p_1, \dots, p_k\}$, such that (1) these partitions taken together cover the whole domain, and (2) any two partitions do not overlap. Formally, we define a function *partition* as follows:

$$partition(R.A_i) = \{p_1, p_2, \dots, p_k\}$$

As an example, consider the attribute *eid* of the *emp* relation above. Suppose the values of domain of this attribute lie in the range $[0, 1000]$. We can divide the whole range into five partitions: $[0, 200]$, $(200, 400]$, $(400, 600]$, $(600, 800]$ and $(800, 1000]$. That is:

$$partition(emp.eid) = \{[0, 200], (200, 400], (400, 600], (600, 800], (800, 1000]\}$$

Different attributes of the same relation may be partitioned using different partition functions. Moreover, the partitions within the domain of the same attribute need not have the same size. It should be clear that the partition of attribute A_i corresponds to a splitting of its domain into a set of buckets.

3.1.2 Identification Functions

Furthermore, we define an identification function called *ident* to assign an identifier $ident_{R.A_i}(p_j)$ to each partition p_j of attribute A_i . Figure 1 shows the identifiers assigned to the five partitions of the attribute *emp.eid*. For instance, $ident_{emp.eid}([0, 200]) = 2$, and $ident_{emp.eid}((800, 1000]) = 4$.

The *ident* function value for a partition is unique, that is, $ident_{R.A_i}(p_j) \neq ident_{R.A_i}(p_l)$, if $j \neq l$. For this purpose, a collision-free hash function that utilizes properties of the partition may be used as an *ident* function. For example, in the case where a partition corresponds to a numeric range, the hash function may use the start and/or end values of a range as the input.

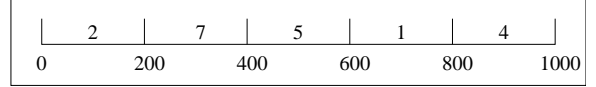


Figure 1: Identification function of *emp.eid*

3.1.3 Mapping Functions

Given the above partition and identification functions, we define a mapping function $Map_{R.A_i}$ that maps a value v in the domain of attribute A_i to the identifier of the partition to which v belongs:

$$Map_{R.A_i}(v) = ident_{R.A_i}(p_j)$$

where p_j is the partition that contains v . For the example above, the following table shows some values of the mapping function for attribute *emp.eid*.

<i>eid</i> value v	23	860	320	875
$Map_{emp.eid}(v)$	2	4	7	4

Let v be a value in the domain of the attribute A_i whose domain values exhibit total order. We further define two more mapping functions on the partitions associated with A_i :

$$Map_{R.A_i}^>(v) = \{ident_{R.A_i}(p_j) | p_j.high \geq v\}$$

$$Map_{R.A_i}^<(v) = \{ident_{R.A_i}(p_j) | p_j.low \leq v\}$$

Essentially, the result of $Map_{R.A_i}^>(v)$ is the set of identifiers corresponding to partitions whose ranges may contain a value not less than v . Likewise, $Map_{R.A_i}^<(v)$ is the set of identifiers corresponding to partitions whose ranges may contain a value not greater than v . In the example above, we have $Map_{emp.eid}^>(670) = \{1, 4\}$ and $Map_{emp.eid}^<(99) = \{2\}$.

3.1.4 Encryption Functions

We now have enough notations to specify how to store the encrypted relation R^S on the server. For each tuple $t = \langle a_1, a_2, \dots, a_n \rangle$ in R , the relation R^S stores a tuple:

$$\langle encrypt(\{a_1, a_2, \dots, a_n\}), Map_{R.A_1}(a_1), Map_{R.A_2}(a_2), \dots, Map_{R.A_n}(a_n) \rangle$$

where *encrypt* is the function used to encrypt a tuple of the relation. For instance, the following is the encrypted relation emp^S stored on the server:

<i>etuple</i> ^S	<i>eid</i> ^S	<i>ename</i> ^S	<i>salary</i> ^S	<i>addr</i> ^S	<i>did</i> ^S
11001100111...	2	19	81	18	2
10000000000...	4	31	59	41	4
11111010000...	7	7	7	22	2
10101010101...	4	71	49	22	4

The first column *etuple* contains the string corresponding to the encrypted tuples in *emp*. For instance, the first tuple in the relation *emp* is encrypted to $encrypt(23, Tom, 70K, Maple, 40) = 11001100111\dots$. The function *encrypt* can be implemented using any block cipher technique such as AES [2], RSA [16], Blowfish [17], DES [6], etc.

The other columns correspond to the indices of the attributes of the relation *emp*. For example, the value of the attribute *eid* in the second tuple is 860, and it belongs to the partition (800,1000]. Since this partition is identified to 4 as shown in Figure 1, we store the value “4” in the attribute eid^S for the second tuple of the encrypted relation R^S .

In general, we use the notation “*E*” to map a relation *R* to its encrypted representation. That is, given a relation $R(A_1, A_2, \dots, A_n)$, relation $E(R)$ is $R^S(etuple, A_1^S, A_2^S, \dots, A_n^S)$. Therefore, in the example above, $E(emp) = emp^S$.

It is now clear that to insert a new tuple, the client simply needs to determine the corresponding encrypted tuple (including the indices) using the function *encrypt* and the mapping functions, which is then sent to the server. Next, the server inserts the new encrypted tuple into the encrypted relation. Note that the server normally has no knowledge about the partition, identification and mapping functions.

3.1.5 Decryption Functions

Given the encryption function *E* that maps a relation to its encrypted representation, we define its inverse function *D* that maps the encrypted representation to its corresponding unencrypted relation. That is, $D(R^S) = R$. In the example above, $D(emp^S) = emp$. In other words, the operator *D* decrypts all strings in the column *etuple* to the unencrypted tuples and drops the auxiliary columns corresponding to the indices. The *D* operator may also be defined in a similar way on relations generated by query expressions. For example, consult the paper by Hacigümüş et al. [11] for details on how to define *D* on relations produced by joins.

3.2 Mapping Conditions

We now study how to translate specific query conditions in SQL operations (such as selections and joins) to corresponding conditions over the server-side representation. This translation function is called a mapping condition, Map_{cond} . Once we know how conditions are translated, we will be ready to discuss how relational operators are translated over the server-side implementation, and finally how query trees are translated.

In the following, we only present the translation of four most common types of query conditions. Translation

of other query conditions is discussed in full detail by Hacigümüş et al. [11].

In the discussion below, we use the following relations to illustrate the translation:

$$emp(eid, ename, salary, addr, did) \\ mgr(mid, did, mname)$$

The identification function of attribute *emp.eid* is inherited from Figure 1. The partition and identification functions of attributes *emp.did* and *mgr.did* are depicted in Figure 2.

emp.did					mgr.did		
2	4	3	1		9	8	
0	100	200	300	400	0	200	400

Figure 2: Identification functions of $\{emp, mgr\}.did$

Attribute = Value: Such a condition arises in selection operations. The mapping condition is defined as follows:

$$Map_{cond}(A_i = v) \Rightarrow A_i^S = Map_{A_i}(v)$$

For example, for the relation *emp*, we have:

$$Map_{cond}(eid = 860) \Rightarrow eid^S = 4$$

since *eid* = 860 is mapped to 4 by the mapping function of this attribute as shown in Figure 1.

Attribute ≤ Value: Such a condition arises in selection operations. The attribute must have a well-defined ordering over which the “≤” operator is defined. To translate this condition, we check if the attribute index A_i^S lies in any of the partitions that may contain a value v' where $v' \leq v$. Formally, the translation is:

$$Map_{cond}(A_i \leq v) \Rightarrow A_i^S \in Map_{A_i}^{\leq}(v)$$

For instance, for the relation *emp*, we have:

$$Map_{cond}(eid \leq 280) \Rightarrow eid^S \in \{2, 7\}$$

since all employee ids not greater than 280 belong to either partition [0,200] with identifier 2 or partition (200,400] with identifier 7. The query condition “Attribute ≥ Value” is symmetric with this condition.

Attribute1 = Attribute2: Such a condition might arise in a join (i.e., equijoin). The two attributes can be from two different relations, or from two instances of the same relation. This condition can also arise in a selection, in which the two attributes are from the same relation. The following is the translation:

$$Map_{cond}(A_i = A_j) \Rightarrow \\ \bigvee_{\phi} (A_i^S = ident_{A_i}(p_k) \wedge A_j^S = ident_{A_j}(p_l))$$

where ϕ is $p_k \in \text{partition}(A_i), p_l \in \text{partition}(A_j), p_k \cap p_l \neq \emptyset$. That is, we consider all possible pairs of partitions of A_i and A_j that overlap. For each pair (p_k, p_l) , we have a condition on the identifiers of these two partitions: $A_i^S = \text{ident}_{A_i}(p_k) \wedge A_j^S = \text{ident}_{A_j}(p_l)$. Finally we take the disjunction of these conditions. The intuition is that each pair of partitions may provide some values of A_i and A_j that can satisfy the condition $A_i = A_j$. For instance, we have

$$\text{Map}_{\text{cond}}(\text{emp}.did = \text{mgr}.did) \Rightarrow C_1$$

where $C_1 =$

$$\begin{aligned} & (\text{emp}^S.did^S = 2 \wedge \text{mgr}^S.did^S = 9) \\ \vee & (\text{emp}^S.did^S = 4 \wedge \text{mgr}^S.did^S = 9) \\ \vee & (\text{emp}^S.did^S = 3 \wedge \text{mgr}^S.did^S = 8) \\ \vee & (\text{emp}^S.did^S = 1 \wedge \text{mgr}^S.did^S = 8). \end{aligned}$$

Condition1 \wedge Condition2: The translation of this condition is given as follows:

$$\begin{aligned} & \text{Map}_{\text{cond}}(\text{Condition1} \wedge \text{Condition2}) \Rightarrow \\ & \text{Map}_{\text{cond}}(\text{Condition1}) \wedge \text{Map}_{\text{cond}}(\text{Condition2}) \end{aligned}$$

Translation of “Condition1 \vee Condition2” is similar.

3.3 Implementing Relational Operators over Encrypted Relations

We are now ready to describe how individual relational operators can be implemented in the database-service-provider model. The strategy is to partition the computation of the operators across the client and the server. Specifically, we attempt to compute a superset of answers generated by the operator using the attribute indices stored at the server. These answers are then filtered at the client after decryption to generate the true results.

In the discussion below, we consider three most common relational operators only: selection, join, and projection. For the rest of the relational operators such as grouping, aggregation, sorting and set difference, consult the original work by Hacigümüş et al. [11]. In addition, we use the operator notations in Garcia-Molina et al. [9].

The Selection Operator (σ): Consider a selection 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, \dots, A_n of R . As discussed before, our strategy is to partially compute the selection operator at the server using the indices associated with the attributes in C , and push the results to the client. The client decrypts the results and filters out tuples that do not satisfy C . Specifically, the operator can be rewritten as follows:

$$\sigma_C(R) = \sigma_C \left(D(\sigma_{\text{Map}_{\text{cond}}(C)}^S(R^S)) \right)$$

In the above notation, we adorn the σ operator that executes at the server with a superscript “ S ” to highlight the fact that the selection operator executes at the server. All non-adorned operators are assumed to execute at the client. We explain the above implementation using an example $\sigma_{\text{eid} < 395 \wedge \text{did} = 140}(\text{emp})$. Based on the definition of mapping conditions discussed in Section 3.2, the above selection operation is translated into $\sigma_C \left(D(\sigma_{C'}^S(\text{emp}^S)) \right)$, where the condition C' on the server is:

$$C' = \text{Map}_{\text{cond}}(C) = (\text{eid}^S \in \{2, 7\} \wedge \text{did}^S = 4)$$

The Join Operator (\bowtie): Consider a join operation $R \bowtie_C T$ on two relations R and T . The join condition C could be either equality conditions (i.e., equijoins), or could be more general conditions (i.e., theta-joins). The above join operation can be implemented as follows:

$$R \bowtie_C T = \sigma_C \left(D(R^S \bowtie_{\text{Map}_{\text{cond}}(C)}^{S} T^S) \right)$$

As before, the “ S ” adornment on the join operator emphasizes the fact that the join is to be executed at the server. For instance, the join operation $\text{emp} \bowtie_{\text{emp}.did = \text{mgr}.did} \text{mgr}$ is translated to:

$$\sigma_C \left(D(\text{emp}^S \bowtie_{C'}^S \text{mgr}^S) \right)$$

where the condition C' on the server is condition C_1 defined in Section 3.2.

The Projection Operator (π): Since the attribute values of each tuple in a relation R are encrypted together into a single string in the *etuple* attribute of relation R^S at the server, a projection π is not implemented at the server. As a result, to compute $\pi_L(R)$, where L is a set of attributes, the strategy is to transmit the entire relation R^S to the client, decrypt the relation at the client, and then compute the projection. That is,

$$\pi_L(R) = \pi_L \left(D(R^S) \right).$$

For instance, we have $\pi_{\text{eid}}(\text{emp}) = \pi_{\text{eid}} \left(D(\text{emp}^S) \right)$.

3.4 Query Splitting

The highest layer of Hacigümüş’ scheme is an algebraic framework for splitting the computation of any SQL query Q across the server and the client. The server uses the implementation of the relational operators discussed in Section 3.3 to compute as much of the query as possible, relegating the remainder of the computation to the client. The objective is to come up with the “best” plan for Q that minimizes the execution cost. The basic idea is to reconstruct the query tree using standard rewrite

rules in relational algebra. Since our discussion about potential security holes in Hacigümüş’ scheme in subsequent sections does not involve this algebraic framework for query splitting, we omit discussion of this topic here. Interested readers are advised to consult Section 5 of Hacigümüş et al. [11]. In fact, we believe that this algebraic framework for query splitting strengthens the security of Hacigümüş’ scheme. Intuitively, it makes sense to believe that Hacigümüş’ scheme is insecure with simple queries such as those involving a single relational operator only, not with complex queries on which the algebraic technique for query splitting is applied to “mess up” the query trees.

4 Semantic Security

The notion of *semantic security*, or *indistinguishability of encryptions*, introduced by Goldwasser and Micali [10], plays an important role in defining security in cryptography.

Definition 4.1 *An encryption scheme E is not semantically secure if we can find two messages m_0 and m_1 in the message space such that, given a ciphertext $c = E(m_i)$ for some $i \in \{0, 1\}$, we can determine whether $i = 0$ or 1 with probability greater than $1/2$.*

Informally speaking, an encryption scheme E is not semantically secure if we can find two messages m_0 and m_1 such that we can distinguish between encryptions of m_0 and m_1 (with a probability better than guessing). We can easily extend the definition of semantic security to the context of Hacigümüş’ scheme.

Definition 4.2 *Hacigümüş’ scheme is said to be not semantically secure if there exist two tuples t_0 and t_1 from the same relation $R(A_1, A_2, \dots, A_n)$, such that given the values of attribute A_k of t_0 and t_1 for some fixed k , and an encrypted tuple $t^S = (\text{encrypt}(t_i), a_1^S, a_2^S, \dots, a_n^S)$ for some $i = 0$ or 1 , we can determine whether $i = 0$ or 1 with probability greater than $1/2$.*

Notice that we only provide the adversary with one attribute value from t_0 and one attribute value from t_1 in Definition 4.2 (these two attribute values are from the same attribute A_k of R). If the adversary knew all the attribute values of t_0 and t_1 , he would have $2n$ pieces of data and Definition 4.2 would not be comparable to Definition 4.1 in terms of security strength.

In this section, we treat the server as the adversary and show that Hacigümüş’ scheme is *not* semantically secure if the adversary knows the schema of the relation R . Although the client does not normally reveal the schema of R to the server, it is plausible practically to assume that the server may obtain the schema of R by

other means. For example, a database administrator at the client site can easily obtain the schema of R , and he might sell this information to the server. Before showing that Hacigümüş’ scheme is semantically insecure, let us explain why we are interested in semantic security.

4.1 Why Semantic Security?

Semantic security is a strong definition of security. However, is it too strong that it is practically a useless notion? While this might be true in a general setting, semantic security may become useful in some specific environments. The impracticality of the definition of semantic security in a general setting seems to be due to its assumption that the adversary knows that the encrypted tuple t^S corresponds to either t_0 or t_1 . However, a golden rule of thumb in computer security is that security through obscurity never works. We can never tell whether the adversary knows that t^S corresponds to one of t_0 or t_1 in every database application.

Example 4.3 *Suppose that Alice works as a teller in a bank. The database system of this bank maintains a relation $\text{acc}(\text{accid}, \text{balance})$, whose encrypted representation is stored remotely at a database-service-provider site. Suppose that Alice also works as a part-time database administrator at the service-provider site. She discovers from the database recovery system that the encrypted relation was modified once at 18:03 EST on Jan 1, 2003, during which one new encrypted tuple t^S is inserted into the encrypted relation. Later she rushes back to the bank, checks the transaction records (hardcopies), and discovers that two new accounts were opened at 18:03 EST on Jan 1, 2003. She then knows with high probability that t^S corresponds to either of these two new accounts.*

Another interesting scenario is that if the adversary’s goal is to determine the decryption of a_k^S and the domain of attribute A_k contains two values only, then semantic security becomes an extremely useful notion.

Example 4.4 *Suppose a database in a voting system maintains a relation $\text{ballots}(\text{voter}, \text{vote})$, whose encrypted representation is stored remotely at the federal government site. In this environment, semantic insecurity essentially means that the server is able to determine whether any encrypted tuple t^S corresponds to a YES vote or a NO vote, which is disastrous for the functionality of a secure voting system.*

4.2 Proof by Example

We are now ready to show that Hacigümüş’ scheme is insecure if the adversary (i.e., the server) knows the schema of relation R . We prove this by means of a practical example.

Example 4.5 Suppose that a bank maintains a relation $R = acc(accid, balance)$ (where $k = 1$ and therefore $A_k = balance$), whose encrypted representation is stored at a database-service-provider site, which is assumed to know the schema of R . Furthermore, suppose that $t_0 = (accid_0 = 1234 - 34 - 56, balance_0 = \$1,000,000K)$ and $t_1 = (accid_1 = 5678 - 78 - 90, balance_1 = \$10K)$ are two tuples in the relation $R = acc$. Let's say $Map_{cond}(balance = \$1,000,000K) \Rightarrow balance^S = 9$ and $Map_{cond}(balance = \$10K) \Rightarrow balance^S = 11$.

Now, as required by Definition 4.2, assume that the adversary (i.e. the server at the service-provider site) is given (1) $balance_0 = \$1,000,000K$, (2) $balance_1 = \$10K$, and (3) an encrypted tuple $t^S = (encrypt(t_i), accid_i^S, balance_i^S)$ corresponding to either t_0 or t_1 (i.e., $i \in \{0, 1\}$). The adversary can now determine whether $i = 0$ or 1 by performing some frequency analysis. First of all, although the server does not know the mapping functions, he knows that the identifier of $balance = \$1,000,000K$ is different from the identifier of $balance = \$10K$ with overwhelming probability, because these two numerical values are far apart from each other within the domain of attribute "balance". Secondly, since there should be logically much more clients with balance around $\$10K$ than $\$1,000,000K$, it follows that if $a_i^S = 11$ (i.e. $i = 1$ with balance $\$10K$), then the server should be able to find lots of encrypted tuples in R^S whose values in the attribute index $balance^S$ are equal to $a_i^S = 11$. Likewise, if $a_i^S = 9$ (i.e. $i = 0$ with balance $\$1,000,000K$), then the server should find very few encrypted tuples in R^S whose values in the attribute index $balance^S$ are equal to $a_i^S = 9$. Using this logic, the server can simply count the number of encrypted tuples whose values in A_k^S are equal to a_k^S , and determine that $i = 1$ if such a number is large and $i = 0$ otherwise, with a high probability of success.

For example, suppose the server performs a frequency analysis and finds that the encrypted relation R^S has 4 accounts with $balance^S = 9$, 10456 accounts with $balance^S = 11$, perhaps thousands with a few other values for $balance^S$, and maybe even a few with another value that represents a balance of 0 (but the server does not know this correspondence). Now, the server examines $t^S = (accid_i^S, balance_i^S)$. If $balance_i^S = 9$, it is highly likely that t^S corresponds to an account that has extremely high balance (i.e., $i = 0$). On the other hand, if $balance_i^S = 11$, it is likely that t^S corresponds to an account that has a "common" balance (i.e., $i = 1$).

Since the server can determine whether $i = 0$ or 1 with high probability of success in this example, we conclude that Hacigümüş' scheme is not semantically secure, under the assumption that the server knows the schema of the relation R .

The example above illustrates how we can use frequency analysis to defeat semantic security in specific environments. Next, we explore a formal theoretical proof that Hacigümüş' scheme is not semantically secure if the adversary is given the probability distribution of the attribute index A_k^S (we inherit the notations used in Definition 4.2). Since values of A_k^S are images of a collision-free hash function, the domain of A_k^S can be viewed as a proper subset of \mathbb{Z} (the set of integers). Thus, we assume that A_k^S exhibits a probability distribution on \mathbb{Z} .

4.3 Theoretical Proof

Let p be the probability distribution of A_k^S on \mathbb{Z} , i.e., $0 \leq p(i) \leq 1$, for $i \in \mathbb{Z}$, and $\sum_{i \in \mathbb{Z}} p(i) = 1$. We can view the aforementioned theoretical proof as an interactive game between us (i.e., the client) and the adversary (i.e., the server). Suppose that we randomly choose two encrypted tuples $t_0^S = (etuple_0, \alpha_1^S, \alpha_2^S, \dots, \alpha_n^S)$ and $t_1^S = (etuple_1, \beta_1^S, \beta_2^S, \dots, \beta_n^S)$ from the encrypted relation R^S according to the probability distribution p of the attribute index A_k^S . Let X and Y be two random variables representing the values of α_k^S and β_k^S respectively. Therefore, X and Y exhibit probability distribution p . Next, we decrypt these two encrypted tuples to obtain $t_0 = (\alpha_1, \alpha_2, \dots, \alpha_n)$ and $t_1 = (\beta_1, \beta_2, \dots, \beta_n)$. We reveal α_k and β_k to the adversary (this matches the first assumption in Definition 4.2). Now, we toss a fair coin c . If $c = 0$, we give t_0^S to the adversary; if $c = 1$, we give t_1^S to the adversary (this matches the second assumption in Definition 4.2). Let $t_c^S = (etuple_c, a_1^S, \dots, a_n^S)$ be the encrypted tuple the adversary receives. The adversary's task is to determine whether t^S corresponds to t_0 or t_1 . It is now clear that Hacigümüş' scheme is not semantically secure if the adversary can give the correct answer with probability greater than $1/2$.

Denote the adversary's response by $R \in \{t_0, t_1\}$. Furthermore, let us denote $t_{min} = (\gamma_1, \gamma_2, \dots, \gamma_n) \in \{t_0, t_1\}$ such that $p(\gamma_k^S) = \min(p(\alpha_k^S), p(\beta_k^S))$, and $t_{max} = (\gamma_1, \gamma_2, \dots, \gamma_n) \in \{t_0, t_1\}$ such that $p(\gamma_k^S) = \max(p(\alpha_k^S), p(\beta_k^S))$. If $X \neq Y$ and the adversary knows (1) p , (2) $p(\alpha_k^S)$, and (3) $p(\beta_k^S)$, then he can apply the following strategy: choose a random cutoff point $z \in \mathbb{Z}$ according to the distribution p (let Z be the random variable representing the cutoff point). Now, if $p(a_k^S) < p(z)$, he would say that t_c^S corresponds to the tuple $R = t_{min}$; if $p(a_k^S) \geq p(z)$, he would say that t_c^S corresponds to $R = t_{max}$. The adversary answers correctly if $R = t_c$. We prove in the following that by using this strategy, the adversary gives a correct answer with probability greater than $1/2$, thus establishing the fact that Hacigümüş' scheme is not semantically secure.

Before presenting the proof, let us point out that this theoretical proof is strongly correlated with Example 4.5.

This theoretical proof shows that Hacigümüş’ scheme is not semantically secure under the assumptions that $X \neq Y$ and that the adversary knows p , $p(\alpha_k^S)$ and $p(\beta_k^S)$. The assumption that $X \neq Y$ is analogous to the fact that the identifiers of $balance = \$1,000,000K$ and of $balance = \$10K$ are not the same in Example 4.5. The adversary’s knowledge of p is analogous to the fact that the server can perform frequency analysis on the stored indices and estimate the probability distribution of $balance^S$ in Example 4.5. The adversary’s knowledge of $p(\alpha_k^S)$ and $p(\beta_k^S)$ is analogous to the adversary’s deduction that tuples with balance $\$1,000,000K$ are much fewer than tuples with balance $\$10K$ in Example 4.5.

Lemma 4.6 *For all $z \in \mathbb{Z}$, $Pr[R = t_c | Z = z] \geq 1/2$.*

Proof. Since either $\min(X, Y) < z$ or $\max(X, Y) \geq z$ must occur, we have that

$$Pr[\min(X, Y) < z] + Pr[\max(X, Y) \geq z] \geq 1.$$

Therefore, we have

$$\begin{aligned} & Pr[R = t_c | Z = z] \\ = & Pr[t_c = t_{min}]Pr[\min(X, Y) < z] \\ & + Pr[t_c = t_{max}]Pr[\max(X, Y) \geq z] \\ = & \frac{1}{2}Pr[\min(X, Y) < z] + \frac{1}{2}Pr[\max(X, Y) \geq z] \\ = & \frac{1}{2}(Pr[\min(X, Y) < z] + Pr[\max(X, Y) \geq z]) \\ \geq & \frac{1}{2}. \square \end{aligned}$$

Lemma 4.7 *There exists an integer ω such that $Pr[R = t_c | Z = \omega] > 1/2$.*

Proof. Since p models the probability distribution of the values of the attribute index A_k^S , which are images of a hash function (the identification function), and every hash function has a finite range, it follows that p is bounded. That is, there exists some greatest integer ω such that $Pr[X = \omega] > 0$ and $Pr[X \leq \omega] = 1$. Now with non-zero probability, we have that $Z = \omega$, so we have $Pr[R = t_c | Z = \omega] = \frac{1}{2}(Pr[\min(X, Y) < \omega] + Pr[\max(X, Y) \geq \omega])$ from the proof of Lemma 4.6. However, since we assume that $X \neq Y$, we have $Pr[\min(X, Y) < \omega] = 1$, whence

$$Pr[R = t_c | Z = \omega] = \frac{1}{2} + \frac{1}{2}Pr[\max(X, Y) \geq \omega] > \frac{1}{2}$$

since $Pr[X = \omega] > 0$ implying that $Pr[\max(X, Y) \geq \omega] > 0$. \square

Theorem 4.8 *$Pr[R = t_c] > 1/2$.*

Proof. Observe that

$$Pr[R = t_c] = \sum_{z \in \mathbb{Z}} Pr[Z = z]Pr[R = t_c | Z = z].$$

Since Lemma 4.6 assures us that $Pr[R = t_c | Z = z] \geq 1/2$ for all $z \in \mathbb{Z}$, and Lemma 4.7 tells us that there is some $\omega \in \mathbb{Z}$ such that $Pr[R = t_c | Z = \omega] > 1/2$, it follows that the weighted average above must also be $> 1/2$. We conclude that $Pr[R = t_c] > 1/2$, whence Hacigümüş’ scheme is not semantically secure. \square

5 Small-Domain Hash

Apparently, if the server knows the partition and identification functions, it can do many things, e.g. performing statistical analysis on the indices. This is the reason why the partition and identification functions would like to be kept secret at the client site. In Section 3.1.2, we mention that the identification functions can be implemented using a collision-free hash function that utilizes properties of the partition.

In order for a hash function $h : X \rightarrow Y$ to be secure, it has to be one-way, i.e., given $y \in Y$, it is infeasible to find an element $x \in X$ such that $h(x) = y$. Many cryptographic hash functions are proven to be secure based on the fact that their domains have infinite size and their ranges are large so that exhaustive search is computationally infeasible. However, even though a hash function can take an infinite number of inputs, if we apply it in specific applications in which the possible number of inputs to the hash function is small, then exhaustive search becomes feasible.

For example, suppose we use a hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^{160}$ as the identification function for an attribute *state*. Suppose that the domain of the attribute *state* is $\{1, 2, \dots, 50\}$. Then any computer can easily form a table showing the 50 possible outputs of h in this particular application of h in a few microseconds! Hacigümüş’ scheme has used the right approach of using hash functions to implement identification functions due to their one-wayness (so that the server, having the indices that are images of the mapping function, cannot find their preimages). However, it fails to take into account the situation that the domain of an attribute is discrete and small, in which case one can easily figure out the identification and mapping functions.

A question arises here. How can the server know what kinds of hash functions are used? Again, a golden rule of thumb in computer security is that security through obscurity never works. In fact, in cryptography, all cryptographic primitives such as the hash functions used in a cryptosystem are always assumed to be public. The programmer who codes the identification functions at

the client site know what kinds of hash functions are used.

6 Query Hiding

In Section 2, we introduce the notion of *instance-hiding schemes*. This notion can be extended to the concept of *hidden queries* [18], which means the database-service-provider has no knowledge about the query type. However, Hacigümüş’ scheme does not support hidden queries because the server can at least deduce the relational operations requested. For example, if the user requests the server to perform a selection operation on the encrypted relation, then the user’s SQL query must involve the selection operator. If the user requests the server to send back the entire encrypted relation, then the user’s SQL query must involve a single projection operator.

Furthermore, every kind of query condition is translated into a unique kind of mapping conditions, as shown in Section 3.2. For example, the query condition “Attribute = Value” is mapped to a simple condition $A_i^S = Map_{A_i}(v)$, but the query condition “Attribute1 = Attribute2” is mapped to a complicated disjunction of conjunctions of such simple conditions. If an attacker is only interested in the query types, this potential security hole in Hacigümüş’ scheme may give the attacker a big help.

7 Decryption by Field

As mentioned in Section 3.1.5, Hacigümüş’ scheme performs decryption by row. Therefore, even if a user at the client site needs only one particular attribute value from a tuple, the corresponding encrypted tuple has to be entirely decrypted and thus all the attribute values revealed. However, no matter how strong an access control infrastructure the client DBMS provides, once the entire encrypted tuple is decrypted at the client site and thus all the attribute values revealed, we cannot guarantee that the user obtains only those attribute values that he is allowed to access.

Users with different logical views of the data should be presented with appropriate views of the database by the DBMS. It is the DBMS’s job to present the user with only those fields which he requests and to which he has the access right. Unfortunately, software cannot be guaranteed to perform this function correctly, because no software is immune against all kinds of computer viruses or Trojan-horse programs [7]. In other words, no software can absolutely guard the system against all kinds of intrusion. Thus, it is the responsibility of the encryption scheme to prohibit attackers and intruders from illegit-

imately using those fields to which they have no access right. In other words, each field should be accessible under a different key.

However, as mentioned in Section 1, encrypting by field is not desirable in both the cryptographic and performance points of view. Therefore, what is essentially needed is an encryption scheme $(\mathcal{E}, \mathcal{D})$ such that, given a tuple $t = (a_1, a_2, \dots, a_n)$, (1) each separate field a_i is associated with a subkey pair (e_i, d_i) , (2) the encryption function \mathcal{E} is a single function of all the fields and their subkeys e_i , (3) we can decrypt a field a_i using the decryption function \mathcal{D} and the subkey d_i without decrypting any other field, and (4) the subkeys (e_i, d_i) are kept secret at the client site, and only those users who have the right to access the field a_i at the client site can own the subkey d_i . This can be described mathematically as:

$$\begin{aligned} etuple &= \mathcal{E}((e_1, a_1), (e_2, a_2), \dots, (e_n, a_n)) \\ a_i &= \mathcal{D}(d_i, etuple) \end{aligned}$$

Unfortunately, the block cipher techniques suggested by Hacigümüş et al. [11], such as AES, RSA, Blowfish and DES (see Section 3.1.4), do not fall into this category of encryption schemes. An example of such encryption schemes based on the Chinese Remainder Theorem (CRT) was proposed by Davida et al. [5].

8 Dishonest Servers

Hacigumus et al. claimed in their paper [11] that Hacigumus’ scheme can guard against untrusted servers. Contrary to their claim, Hacigumus’ scheme is unable to ensure the correctness of the query result in presence of a dishonest server that does not answer the query correctly on purpose by returning a wrong set of tuples. Putting it in another way, the server can choose to execute a query different from the one sent, and the client has no way to detect that. This could have disastrous effect, as shown by the following example.

Example 8.1 *In presence of auditors, a huge bank needs to pay interests to clients whose balance exceeds \$1,000,000K. The bank stores the encrypted representation of the relation $R = acc(accid, balance)$ remotely at a database-service-provider site. It issues the query $\sigma_{balance \geq \$1,000,000K}(R)$. Suppose that $Map_{cond}(balance \geq \$1,000,000K) \Rightarrow balance^S \in \{2, 9\}$, and so the bank sends the translated query $\sigma_{balance^S \in \{2, 9\}}(R^S)$ to the server where the data are stored. In the worst case, the bank may bribe the database administrator of the database-service-provider site, who then configures the server so that no (encrypted) tuples are returned to the bank, indicating that there is no client whose balance exceeds \$1,000,000K.*

This example illustrates a misconception on which Hacıgümüş’ scheme is based on: it is dangerous to say that to achieve “total” data privacy (i.e. security against untrusted servers), the only requirement is that encrypted data may not be decrypted at the provider site. Definitely, encryption of the data is necessary to prevent the server from reading the actual data, but it is not a sufficient security measure. Analogous to an active adversary who can inject or delete messages sent over an insecure channel in the traditional two-party communication model, a malicious server can deliberately control the number of tuples returned to the client here.

9 Conclusion and Future Work

We have presented five potential security holes in Hacıgümüş’ scheme of executing SQL queries over encrypted data in the database-service-provider model; namely, that the scheme is not semantically secure, that an adversary can figure out the identification functions if the input domain is small, that the server can deduce the query type from the translated queries sent from the client, that the scheme does not support decryptions by field, and that the scheme cannot detect dishonest servers. These security holes may lead to serious security problems in specific environments. However, we want to emphasize here that we believe Hacıgümüş’ scheme is secure in a general setting. Nonetheless, it would be best if the security provided by Hacıgümüş’ scheme can be quantified exactly. We have shown that Hacıgümüş’ scheme is not perfectly secure, because it is not even semantically secure. However, can we come up with a tighter theoretical upper bound on the security strength of Hacıgümüş’ scheme?

References

[1] M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. In *J. Comput. and System Sci.*, 1989.

[2] AES. Advanced Encryption Standard. *National Institute of Science and Technology, FIPS 197*, 2001.

[3] D. Chaum, C. Crépeau, and I. Damgaard. Multi-party unconditionally secure protocols. In *Proc. of ACM STOC*, 1988.

[4] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proc. of IEEE FOCS*, 1995.

[5] G. Davida, D. Wells, and J. Kam. A database encryption system with subkeys. In *ACM Tran. on Database Systems*, 1981.

[6] DES. Data Encryption Standard. *FIPS PUB 46, Federal Information Processing Standards Publication*, 1977.

[7] F. Cohen. Computer viruses: theory and experiments. *Computers and Security*, 6:22-35, 1987.

[8] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: secure fault-tolerant protocols and the public-key model. In *Proc. of CRYPTO*, 1987.

[9] H. Garcia-Molina, J. Ullman, and J. Widom. *Database systems: the complete book*. Prentice Hall, 2002.

[10] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Com. Sys. Sci.* 28 (1984), pages 270-299.

[11] H. Hacıgümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of ACM SIGMOD*, 2002.

[12] H. Hacıgümüş, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of ICDE*, 2002.

[13] J. He and M. Wang. Cryptography and relational database management systems. In *Proc. of IDEAS*, 2001.

[14] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proc. of IEEE FOCS*, 1997.

[15] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169-178, 1978.

[16] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120-126, 1978.

[17] B. Schneier. Description of a new variable-length key, block cipher (blowfish), fast software encryption. In *Cambridge Security Workshop Proceedings*, 1994.

[18] D. Song, D. Wagner and A. Perrig. Search on encrypted data. In *Proc. of IEEE SRSP*, 2000.

[19] A. Yao. Protocols for secure computations. In *Proc. of IEEE FOCS*, 1982.