Encryption Techniques for Secure Database Outsourcing

Sergei Evdokimov, Oliver Günther

Humboldt-Universität zu Berlin
Spandauer str. 1, 10178 Berlin, Germany
{evdokim,guenther}@wiwi.hu-berlin.de

Abstract. While the idea of database outsourcing is becoming increasingly popular, the associated security risks still prevent many potential users from deploying it. In particular, the need to give full access to one's data to a third party, the database service provider, remains a major obstacle. A seemingly obvious solution is to encrypt the data in such a way that the service provider retains the ability to perform relational operations on the encrypted database. In this paper we present a model and an encryption scheme that solves this problem at least partially. Our approach represents the provably secure solution to the database outsourcing problem that allows operations exact select, Cartesian product, and projection, and that guarantees the probability of erroneous answers to be negligible. Our scheme is simple and practical, and it allows effective searches on encrypted tables: For a table consisting of n tuples the scheme performs search in $O(n)$ steps.

1 Introduction

In this paper we consider the problem in which one party (Alice) owns a database and wants to outsource it to a second party (Bob), even though the trust of Alice in Bob is limited. Alice wants to be sure that the data she outsources is exposed neither to another party nor to Bob. Legal options, such as contracts, are available, but their effectiveness is often limited [**?**].

If, for example, the database is acquired by other company, it may be unclear whether the new owner is bound by the contract [**?**]. As Amazon says it: "In the unlikely event that Amazon.com Inc., or substantially all of its assets are acquired, customer information will of course be one of the transferred assets“. If the data were encrypted, this problem could not arise.

Ideally, Alice would like to have the data encrypted and only give the ciphertext to Bob, the database service provider. But if Bob is not trusted, he cannot participate in the encryption/decryption process. Usually Bob does not just store the data, but also processes non-trivial queries sent by Alice and therefore should be able to process these queries without decrypting the stored data. About 30 years ago, Rivest et al. [**?**] described a possible approach for solving such a problem they called *privacy homomorphism*. They proposed a scheme to encrypt data in such a way that certain operations can be performed on the ciphertext without decrypting it.

In this paper we present *privacy homomorphism* for the relational operations exact select, projection and Cartesian product. Additionally the scheme allows insert, exact delete, exact update and union with duplicates. Exact select, exact delete and exact update are variants of select, delete and update operations with condition predicates (WHERE-part of the corresponding SQL queries) restricted to a combination of equalities connected by AND or OR. The result of a union with duplicates is the union of two relations without duplicate tuples being removed.

Our approach displays the following key characteristics

- Our scheme is *provably secure* and can sustain a chosen-plaintext and a posteriori chosen-ciphertext attacks.

- Our scheme reveals *nothing but the number of tuples that share a queried value* while performing an exact select .

- Our scheme allows to *efficiently* perform the supported operations on an encrypted database. The scheme does not affect the time needed to perform projection, Cartesian product and insert operations. Checking whether a tuple satisfies an equality condition of an exact select requires $O(1)$ operations; therefore exact update, exact delete and exact select require $O(n)$ operations, where n is the number of tuples in the queried relation.

- Our scheme also avoids a problem of many previous solutions, such as the outsourcing approach of Hacıgümüş et al. [**?**] or the search algorithms on encrypted data of Goh [**?**] and Song et al. [**?**]. All those solutions may return erroneous tuples that do not satisfy the select condition. This requires Alice each time to perform postfiltering of the received result set, which reduces the performance and complicates the development process of a client software. This especially becomes an issue when Alice uses a mobile device for accessing the encrypted database. The only scheme that allows to perform search on encrypted data and does not require postfiltering is described in [**?**]. This scheme, however, can hardly be applied to databases since a search on encrypted data is restricted to the search with predefined keywords, which constitutes a severe limitation. The scheme we are proposing also may include erroneous tuples in the result set of an exact select operation but the probability of such an error is negligible.

The structure of the paper is as follows. Section 2 gives the relevant definitions and Section 3 reviews related work. Section 4 introduces the encryption scheme constituting a database privacy homomorphism and proves its security. Section 5 shows how to perform certain relational operations on an encrypted database. Section 6 contains some ideas on how to organize indexing of the encrypted database, and Section 7 presents our conclusions and ideas for future work.

2 Relevant Definitions Notions of Security

In this section we briefly introduce some cryptographic primitives and definitions used in the paper. We use the standard cryptography definitions; see, e.g., [**?**],[**?**].

By $\{0,1\}^n$ we define the set of all binary strings of length n. By $k \overset{R}{\leftarrow} \mathcal{K}$ we say that k is randomly and uniformly chosen from set $\mathcal{K}$.

Definition 1 (pseudo-random function). *A mapping $F : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{Y}$, where $\mathcal{K} = \{0,1\}^n$, is a* pseudo-random function *if for every PPT oracle algorithm A, every positive polynomial $p(n)$, and all sufficiently large n, the advantage $AdvA < 1/p(n)$. The advantage is defined as*

$$\mathrm{Adv}A = |Pr[A^{F_k} = 1] - Pr[A^{\phi} = 1]|,$$

where ϕ is a function chosen randomly and uniformly from the set of all functions that map $\mathcal{X}$ to $\mathcal{Y}$.

A function that after a certain point decreases faster than one over any polynomial is called *negligible*. Thus, it also can be said that the advantage is negligible.

Consider now set of plaintexts $\mathcal{X} = \{0,1\}^m$, set of ciphertexts $\mathcal{Y} = \{0,1\}^l$ and set of keys $\mathcal{K} = \{0,1\}^n$.

Definition 2 (symmetric encryption scheme).
An encryption scheme *is a triple $(\mathcal{K}, E, D)$, where $E : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{Y}$ is a PPT algorithm (encryption algorithm) that maps a key $k \in \mathcal{K}$ and a plaintext $x \in \mathcal{X}$ into a corresponding ciphertext $c \in \mathcal{Y}$ and $D : \mathcal{K} \times \mathcal{Y} \mapsto \mathcal{X}$ is a polynomial-time algorithm (decryption algorithm) that maps a key k and a ciphertext c into a corresponding plaintext x. It must hold that $D_k(E_k(x)) = x$. Keys are chosen randomly and uniformly from the key space $\mathcal{K}$. The bit length n of the keys is called* security parameter *of the scheme.*

There are also asymmetric encryption schemes that use two different keys: one for encryption and a second for decryption. In our paper we only use symmetric schemes.

The security of an encryption scheme is defined as follows:

Definition 3 (indistinguishability of encryptions).
An encryption scheme $(\mathcal{K}, E, D)$ is indistinguishably secure *if for every $x, y \in \mathcal{X}$, every PPT algorithm A, every positive polynomial p, and all sufficiently large n, the advantage $\mathrm{Adv}A_{xy} < 1/p(n)$. The advantage is defined as*

$$\mathrm{Adv}A_{xy} = |\Pr[A(E_k(x)) = 1] - \Pr[A(E_k(y)) = 1]|.$$

The definition of indistinguishability guarantees that in case a computationally bounded adversary obtains the ciphertext of plaintext x, the probability that she is able to infer any distinguishing property of the plaintext (except for the length of the plaintext) is negligible.

In our paper we will also use a construct called *pseudo-random permutation*, an indistinguishably secure encryption scheme that is a bijection and for which $\mathcal{X} = \mathcal{Y}$.

Definition 3 guarantees security only if a key is used once. In order to securely encrypt several messages, a new key should be generated for each new encryption. But often it should be possible to securely encrypt several messages using the same key. Encryption schemes that allow this are called indistinguishably secure for multiple messages:

Definition 4 (indistinguishability of encryptions for multiple messages).

An encryption scheme $(\mathcal{K}, E, D)$ *is* indistinguishably secure for multiple messages *if for every* $\bar{x} = (x_1, \ldots, x_t), \bar{y} = (y_1, \ldots, y_t)$*, every PPT algorithm* A*, every positive polynomial* p*, and all sufficiently large* n*, the advantage* $\mathrm{Adv}A_{\bar{x}\bar{y}} < 1/p(n)$*. The advantage is defined as*

$$\mathrm{Adv}A_{\bar{x}\bar{y}} = |\Pr[A(\bar{E}_k(\bar{x})) = 1] - \Pr[A(\bar{E}_k(\bar{y})) = 1]|.$$

$\bar{E}_k(\bar{x})$ denotes the sequence of ciphertexts that are produced by encrypting each x_i with encryption algorithm E_k: $\bar{E}_k(\bar{x}) = (E_k(x_1), \ldots, E_k(x_t))$.

The indistinguishability definitions provided so far guarantee the protection only from a "passive" adversary. Such adversary simply eavesdrops ciphertexts and tries to get some information about the corresponding plaintexts. But in real applications the adversary can also be "active" and additionally cause the sender to encrypt a message of her choice (chosen-plaintext attack) or even cause the receiver to decrypt the ciphertext of her choice (chosen-ciphertext attack). Formally this is described as the ability of the adversary to query the encryption (decryption) oracle in case of a chosen-plaintext (chosen-ciphertext) attack.

Definition 5 (indistinguishability under chosen-plaintext attack (IND-CPA)).

An encryption scheme $(\mathcal{K}, E, D)$ *is* indistinguishably secure under a chosen-plaintext attack *if for every* $x, y \in \mathcal{X}$*, every PPT algorithm* A^{E_k} *with access to encryption oracle* E_k*, every positive polynomial* p*, and all sufficiently large* n*, advantage* $\mathrm{Adv}A^{E_k}_{xy} < 1/p(n)$*. The advantage is defined as*

$$\mathrm{Adv}A^{E_k}_{xy} = |\Pr[A^{E_k}(E_k(x)) = 1] - \Pr[A^{E_k}(E_k(y)) = 1]|.$$

According to [**?**], Definition 4 and Definition 5 are equivalent: If an encryption scheme is indistinguishably secure for multiple messages then the scheme is also IND-CPA secure.

A chosen-ciphertext attack can be represented as the following game:

1. The challenger generates key k: $k \overset{R}{\leftarrow} \mathcal{K}$.
2. The adversary asks the decryption oracle for the plaintexts corresponding to the ciphertexts of her choice.
3. The challenger generates two plaintext strings and gives the adversary the encryption of one of them.
4. The adversary may additionally ask the oracle for the decryption of some ciphertexts except for the decryption of the received challenge.

5. The adversary tries to guess which of the two strings he was given and halts.

The described attack is called *posteriori chosen-ciphertext attack* (IND-CCA2). When step 4 is omitted, the attack is called *a-priori chosen-ciphertext attack* (IND-CCA). It is clear that security against IND-CCA2 attack guarantees security against IND-CCA attack. Further in the paper, when speaking about chosen-ciphertext indistinguishability we will suggest IND-CCA2.

Definition 6 (posteriori chosen-ciphertext attack indistinguishability (IND-CCA2)).
An encryption scheme $(\mathcal{K}, E, D)$ *is* indistinguishable with respect to posteriori chosen-ciphertext attack *if for every* $x, y \in \mathcal{X}$*, every PPT algorithm* A^{D_k} *with access to decryption oracle* D_k*, every positive polynomial* p*, and all sufficiently large* n*, the advantage* $\mathrm{Adv}A_{xy}^{D_k} < 1/p(n)$*. The advantage is defined as*

$$\mathrm{Adv}A_{xy}^{D_k} = \left|\Pr\left[A^{D_k}(E_k(x)) = 1\right] - \Pr\left[A^{D_k}(E_k(y)) = 1\right]\right|.$$

Usually, in scenarios where a chosen-ciphertext attack is possible, a chosen-plaintext attack is possible too. Therefore, when speaking about chosen-ciphertext attacks we will also assume the possibility of a chosen-plaintext attack. Also, when speaking about indistinguishable security, we mean indistinguishable security for multiple messages or IND-CPA security.

3 Related Work and Security Analysis of Existing Approaches

As mentioned, the idea of a privacy homomorphism was first described by Rivest et al. [?]. There it was also mentioned that one of the most promising applications of privacy homomorphisms could be encryption of databases. If the privacy homomorphism preserved some of the relational operations, then it would be possible to process encrypted relations without decrypting them. For example, consider an encryption scheme that tuple by tuple deterministically encrypts all the attribute values of the database relations. Deterministic encryption means that each plaintext is bijectively mapped to the corresponding ciphertext. That allows to state that equality of the ciphertexts means equality of the corresponding plaintexts and, therefore, if the whole database is encrypted with such an encryption scheme it is possible to perform exact selects, unions, differences, Cartesian products and projections on the encrypted tables. Unfortunately, such a straightforward solution is vulnerable to statistical attacks and cannot be considered for any practical use.

In 2001 Hacıgümüş et al. [?] described an encryption scheme that allowed to perform all relational operations on an encrypted database and made the statistical attack on the scheme less obvious as in the example described above. According to the scheme, the domain of each attribute is partitioned into intervals, and each attribute value is mapped to the interval that contains it. Then the intervals are deterministically encrypted and attached to the secure encryptions of the tuples. The way the relational operations are carried out is similar to the deterministic privacy homomorphism described above. The only difference is that instead of operating with deterministically encrypted attribute values, the scheme uses the deterministically encrypted containing intervals - while the attributes are securely hidden. So, for example, an exact select operation will return the tuples with the attribute values contained in the interval that is stated as the argument of the select operation. This requires Alice (the user) to perform postfiltering in order to remove the tuples that have the attribute values that belong to the queried interval and are not equal to the argument of the select operation. On the other hand it makes the attack on the encryption scheme less straightforward. However, it is clear that an adversary or Bob still learns *something* about the data.

It is easy to show that both encryption algorithms do not comply with Definition 4 and, therefore, are not indistinguishably secure for multiple messages. By x_i, y_j we define tuples of the relations and by $\bar{x}, \bar{y}$ we define the relations consisting of these tuples. In case when the encryption scheme deterministically encrypts attribute values, the relation with identical tuples can easily be distinguished from the relation with the same number of different tuples: The encryption of the first set consists of the set of identical ciphertexts and the ciphertexts for the second set will be different. If we build algorithm A that outputs 1 when the ciphertexts are the same and 0 otherwise, the advantage for such tables will be 1, which is not negligible. Analogously one can distinguish tables encrypted with the scheme proposed by Hacıgümüş et al. Consider two tables:

ID	salary
171	4900
481	1200

Table 1

ID	salary
171	4900
481	4900

Table 2

According to the scheme, the salaries in the first table will be mapped to different intervals with high probability. The salaries in the second table will be mapped to the same interval. Since the intervals are encrypted deterministically, the ciphertexts that correspond to the intervals of the ”salary“ attribute of the first table will be different and the analogous intervals' encryptions for the second table will be identical. Hence, algorithm A can determine to which table corresponds the received ciphertext: If the ciphertexts that correspond to the “salary” intervals are different, A outputs 0; otherwise 1. The advantage for such an algorithm will again be non-negligible.

In modern cryptography, the weakest requirement for an encryption scheme to have any practical applications is IND-CPA security. In case of IND-CCA2 security, it may seem that the assumption of an adversary's ability to decrypt ciphertexts of her choice is very unlikely to be satisfied. However, the successful chosen-ciphertext attack on the widely used internet security protocol SSL discovered by Bleichenbacher [**?**] demonstrates the relevancy of IND-CCA2 security.

The encryption scheme that allows to perform exact selects on encrypted relations and is IND-CPA and IND-CCA2 secure is described in [?]. The scheme is based on encryption techniques that allow to perform searches on encrypted data [?],[?]. It uses the similarity between searching for text documents that contain a defined keyword and exact select operation for databases. The idea behind the scheme is to bijectively map tuples of the relation to text documents by treating each attribute value as a sequence of characters or ”word“, encrypt the resulting documents with the scheme that supports searches on encrypted data, and, instead of issuing exact selects, issue the corresponding search operations. E.g, Table 1 from the example above can be mapped to the following set of documents:

171#ID4900SL
1200SL481#ID

In this example each attribute value is mapped to the word consisting of 6 symbols where '#' is the padding symbol and ”ID“ and ”SL“ are identifiers that help to map the words back to the values of the corresponding attributes (ID and salary). The mapping of the tuples to the documents define the way exact selects are converted to the search operations: E.g., in order to process the exact select `SELECT * FROM Table1 WHERE salary=4900` Bob performs the search for documents that contain word ”4900SL“.

Disadvantages of the proposed method include the necessity of postfiltering of an exact select results (since the schemes [?,?] allow with high probability the inclusion of erroneous tuples in the result of a search operation) and the infeasibility of projection and Cartesian product, due to the impossibility to concatenate and split encrypted tuples.

In [?] Yang et al. proposed the encryption scheme similar to the one we discuss in this paper. In their work they introduce own security model and base the security analysis of the scheme on the different notion of security. However, though the approach they take for building the encryption scheme is correct, the analytical part of the paper contains several serious flaws. So, as it can be easily illustrated by a counterexample, the definition of security on which the authors base their reasoning in fact does not require a database to be encrypted at all. Additionally, the authors mistakenly suppose that their scheme does not include erroneous tuples in the resulting set of a processed query. For the more detailed analysis of this work refer to Appendix B.

4 Secure Database Encryption

In this section we show how to construct an encryption scheme that can serve as a privacy homomorphism for a well-defined subset of relational operations. First we show how to perform encryption and decryption of a database, then we provide the proof of IND-CPA security of the scheme. Algorithms for the relational operators follow in Section 5.

Attribute of R	Attribute of R^E	Type of Attribute
ID	f4FR32	int
Name	aSC3f7	string[100]
	...	
Address	sF3nD4	String[200]

Table 1. Corresponding attributes and data types

4.1 Construction

We build our scheme as the combination of *cryptographic primitives*. The term cryptographic primitive describes an elementary cryptographic algorithm that satisfies certain security requirements and is used as a building block for encryption schemes. When implementing the encryption scheme as a computer program, the primitives are substituted with their implementations that are *believed* to satisfy necessary security requirements (DES, RSA, MD5, SHA etc.). By saying ”believed“ we mean that so far there were no successful attacks on these implementations. In case a security breach is found, the compromised implementation can be substituted by another construct that possesses the needed properties and is considered as secure.

Our encryption scheme uses the following cryptographic primitives:

- $(\mathcal{K}, E, D)$, $\mathcal{K} = \{0,1\}^m$, $\mathcal{X} = \{0,1\}^m$, $E : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{Y}$ is a symmetric encryption schema that is IND-CPA secure and key space and space of plaintexts are identical.
- $(\mathcal{K}^0, E^0, D^0)$, $E^0 : \mathcal{K}^0 \times \mathcal{X} \mapsto \mathcal{Y}^0$ is a symmetric encryption schema that is IND-CPA secure.
- $P : \mathcal{K}' \times \mathcal{X} \mapsto \mathcal{X}$, $\mathcal{X} = \{0,1\}^m$ is a pseudo-random permutation. Since $\mathcal{K} = \mathcal{X}$ we can also write $P : \mathcal{K}' \times \mathcal{X} \mapsto \mathcal{K}$

The indistinguishable security of encryption scheme $(\mathcal{K}, E, D)$ means that the scheme is probabilistic: Same plaintexts may be encrypted as different ciphertexts. Otherwise it would always be possible to distinguish a set of ciphertexts that are encryptions of the identical plaintexts from a set that contains encryptions of different plaintexts. On the contrary, the pseudo-random permutation P is deterministic and maps identical plaintexts to identical ciphertexts.

Key generation. Alice generates the encryption key $\hat{k}$ that is a triple (k_0, k_1, k_2), where $k_0 \overset{R}{\leftarrow} \mathcal{K}_0$, $k_1 \overset{R}{\leftarrow} \mathcal{K}'$, $k_2 \overset{R}{\leftarrow} \mathcal{K}'$: k^0 is the key for encryption scheme $(\mathcal{K}^0, E^0, D^0)$, k_1, k_2 are the keys for pseudo-random permutation P (k_1, k_2 are chosen independently).

Encryption. Suppose that Alice wants to encrypt a relational database that consists of several relations. The idea behind the scheme is to augment encryptions of every attribute value with an additional piece of information, viz., a search tag that will allow Bob to execute search on the ciphertexts without getting any information about the corresponding plaintext values.

Each relation is encrypted separately, so we describe the encryption algorithm for an arbitrary attribute value of a relation $R(a_1 : D_1, \dots, a_l : D_l)$. Without loss of generality we suppose that $D_i \cap D_j = \emptyset, i \neq j$.[1] The encryption algorithm maps the relation R to an encrypted relation R^E that has the same number of attributes but the domains of the attributes are changed to binary strings. Since the information about the domains will be not available after encryption, Alice is responsible for saving this information and performing correct type conversions during the decryption process (this will be discussed later in more detail).

Before starting the encryption, Alice generates key $\hat{k}$ and then performs tuple-by-tuple encryption of relation R, separately encrypting each attribute value. Let $x \in D_i$ be a plaintext value of attribute a_i. The encryption algorithm treats plaintext x as a binary value and encrypts it by performing the following steps:

1. Plaintext x is encrypted with encryption function E^0 and key k_0: $c = E^0_{k_0}(x)$.
2. Pseudo-random permutation P generates key k_s: $k_s = P_{k_1}(x)$. Key k_s will be used for generating the search tag.
3. Plaintext x is deterministically encrypted by pseudo-random permutation P with key k_2: $s = P_{k_2}(x)$
4. Using ciphertext s and key k_s the search tag is generated: $t = E_{k_s}(s)$.
5. The output of the algorithm is the pair (t, c).

With $\hat{E}$ denoting the encryption algorithm, whole procedure can be described as

$$\hat{E}_{\hat{k}}(x) := (E_{P_{k_1}(x)}(P_{k_2}(x)), E^0_{k_0}(x)), \tag{1}$$

where $\hat{k} = (k_0, k_1, k_2)$.

After the encryption procedure was applied to each attribute value of tuple $< a_1 : x_1, \dots, a_l : x_l >$, the resulting ciphertexts form a new tuple $< a^E_1 : (t_1, c_1), \dots, a^E_l : (t_l, c_l) >$ that belongs to relation R^E. In order to hide the structure of the database, the names of the attributes should be changed ($a_i \neq a^E_i$). To correctly decrypt the encrypted relation, Alice should store the information about the correspondences between the attributes of relation R and the attributes of the relation R^E. Also, as mentioned earlier, the encryption changes the domains of the attributes to a raw binary data. The information about the domains of original attributes should also be maintained by Alice (Table 1).

In order to use the described encryption scheme for encrypting values of different attributes, the domains of relation R^E should be of the same length. That means that, before being encrypted, the values should be padded up to the length of the domain that has the longest binary representation. Note that it is very unlikely that an attribute containing very long values will be used by an exact select (e.g., attributes that contain full address, long text, multimedia data etc.). Such attributes should either be split into several shorter attributes or encrypted with a conventional secure encryption scheme if no select queries are expected for them.

[1] If not, then elements of each domain D_i can be appended with bits that uniquely identify attribute a_i within the table.

Decryption. The decryption is performed by decrypting the attribute values of every tuple of relation R^E and filling relation R with the corresponding plaintexts tuples taking into account the information from Table 1. The decryption of ciphertext (t, c) is performed straightforwardly:

$$\hat{D}_{\hat{k}}(t, c) := D_{k_0}(c) = x, \tag{2}$$

where $\hat{k} = (k_0, k_1, k_2)$.

Using the information stored in Table 1 the plaintext is converted to the appropriate type and saved as the value of the corresponding attribute.

The final scheme is defined as $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$, where $\hat{E}$ is defined according to (1), $\hat{D}$ is defined according (2) and $\hat{\mathcal{K}} = (\mathcal{K}^0 \times \mathcal{K}' \times \mathcal{K}')$.

4.2 Proofs of Security

Theorem 1. *Encryption scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ is IND-CPA secure.*

See Appendix A.1 for a proof of the theorem.

Even though the encryption scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ provides IND-CPA security, the scheme is vulnerable to IND-CCA2 attack. Even if we strengthen the security of cryptographic primitives and require IND-CCA2 security for encryption schemes $(\mathcal{K}, E, D)$ and
$(\mathcal{K}^0, E^0, D^0)$ the resulting scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ will still be vulnerable to a posteriori chosen-ciphertext attack that can allow an adversary to recover the plaintext from a given ciphertext.

Theorem 2. *Encryption scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ is not IND-CCA2 secure.*

Proof sketch. For our scheme, where $\hat{E}_{\hat{k}}(x) = (t, c)$, the distinguishing algorithm proceeds as follows:

1. The algorithm queries the encryption oracle for x and gets ciphertext (t', c').
2. The algorithm queries the decryption oracle for (t', c). This query is allowed and returns some α (note that if the algorithm is input x, then $\alpha = x$).
3. If $\alpha = x$ the algorithm outputs 1; otherwise 0.

Clearly, the advantage of the algorithm is non-negligible. □

The scheme can be easily modified to be IND-CCA2 secure. There exist standard techniques that make an IND-CPA secure encryption scheme secure against CCA2 attack. The underlying idea is to make it infeasible for an adversary having access to a decryption oracle to forge a legitimate ciphertext. One of the possibilities is to augment the ciphertext with a tag containing “Message Authentication Code” (MAC). A ciphertext is considered legitimate if in a pair (c, MAC), MAC is the valid authentication code of c. The simplest way for generating MAC for a ciphertext is to input the ciphertext into a pseudo-random function and use the output as the authentication code.

We define the IND-CCA2 secure version of encryption scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ as $(\hat{\mathcal{K}}', \hat{E}', \hat{D}')$ and construct it as follows:

Let $F : \mathcal{K}^F \times \mathcal{Y} \times \mathcal{Y}^0 \mapsto \mathcal{Y} \times \mathcal{Y}^0$ or $F_{k_F}(t,c) = a$, $k_F \in \mathcal{K}^F$, $t \in \mathcal{Y}$, $c \in \mathcal{Y}^0$, $a \in \mathcal{Y} \times \mathcal{Y}^0$.

Key generation. $\hat{k}' \overset{R}{\leftarrow} \hat{\mathcal{K}}'$, where $\hat{\mathcal{K}}' = \hat{\mathcal{K}} \times \mathcal{K}^F = \mathcal{K} \times \mathcal{K}^p \times \mathcal{K}^p \times \mathcal{K}^F$.

Encryption. $\hat{E}'_{\hat{k}'}(x) = (\hat{E}_{\hat{k}}(x), F_{k_F}(\hat{E}_{\hat{k}}(x))) = (t, c, F_{k_F}(c,t)) = (t,c,a)$, where $\hat{k}' = (\hat{k}, k_f) = (k, k_1, k_2, k_F)$.

Decryption. $\hat{D}'_{\hat{k}'}(t,c,a) = \hat{D}_{\hat{k}}(t,c) = D_k(c)$ if $F_{k_F}(t,c) = a$ otherwise the ciphertext is not legitimate and is thus rejected.

According to [**?**], the encryption scheme $(\hat{\mathcal{K}}', \hat{E}', \hat{D}')$ is IND-CCA2 secure.

Since the only difference between schemes $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ and $(\hat{\mathcal{K}}', \hat{E}', \hat{D}')$ is the authentication tag that is simply attached to the ciphertext, all the operations that are feasible under scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ will remain feasible under scheme $(\hat{\mathcal{K}}', \hat{E}', \hat{D}')$. Note that unlike scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ that does not require search tag for decryption, in order to perform decryption of ciphertext (t,c,a), the scheme $(\hat{\mathcal{K}}', \hat{E}', \hat{D}')$ needs all the members of the triple in order to check the legitimacy of the ciphertext. That means that if a database is encrypted with scheme $(\hat{\mathcal{K}}', \hat{E}', \hat{D}')$, the complete triples (t,c,a) should be sent to Alice, thus tripling the amount of transferred data compared to the case when the scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ is used.

5 Operations on Encrypted Relational Databases

In this section we discuss the relational operations that are feasible under the proposed scheme and security implications that arise when some of operations are performed.

5.1 Allowed Operations

The encryption schema described above allows to perform the following subset of relational operations on encrypted relations: exact select, projection, Cartesian product and equijoin. Also the scheme allows to perform union with duplicates, exact update, exact delete and insert.

Exact Select. The proposed encryption scheme allows to perform exact selects (`SELECT...`
`FROM...WHERE <attribute_name>=<value>`) on the encrypted relation without decrypting it. Exact selects with more than one selection attribute connected by AND or OR are discussed at the end of this section.

Suppose, that exact select $\sigma_{a_i.x_q}$ should be performed on relation R that is encrypted and stored as R^E. Then the following actions should be performed:

1. Alice transforms the query $\sigma_{a_i.x_q}$ into the following triple
$$(q, k_q, a_i^E) = (P_{k_2}(x_q), P_{k_1}(x_q), a_i^E), \tag{3}$$
where a_i^E is the name of the attribute of relation R^E that corresponds to attribute a_i. The corresponding attributes are taken from the structure analogous to Table 1.

2. Tuple by tuple, Bob checks every value (t, c) of attribute a_i^E for the following equality:
$$D_{k_q}(t) = q. \tag{4}$$
The tuples that satisfy the equality are marked.
3. After all the tuples of the relation R^E are checked, Bob sends the marked tuples to Alice. The search tags of the attribute values are not needed for the decryption and can thus be discarded. That would reduce the amounts of the data transferred to Alice by about half.
4. Using key k_0, Alice decrypts the received ciphertexts.

Recall that, when encrypting plaintext x, the encryption algorithm $\hat{E}$ generates a key $k_s = P_{k_1}(x)$ and a ciphertext $s = E_{k_s}(P_{k_2}(x))$. If the ciphertext (t, c), whose search tag was checked at step 2, is the encryption of x_q, then $k_s = k_q$, $s = q$, and equality (4) holds true due to

$$D_{k_q}(t) = D_{k_q}(E_{k_s}(s)) = D_{k_q}(E_{k_s}(P_{k_2}(x_q))) = P_{k_2}(x_q) = q.$$

Therefore, all the tuples that have encryption of x_q as the value of attribute a_i^E will be marked and included in the result set.

Note that the triple provided by Alice does not contain any plaintext values. That allows Bob to perform search for $a_i.x_q$ without $a_i.x_q$ itself being revealed.

However, we cannot call this scheme privacy homomorphism in a strict sense, since the set of marked tuples may contain tuples that do not belong to the actual solution. This can happen due to following collision:

$$D_{P_{k_1}(x_q)}(E_{P_{k_1}(x)}(P_{k_2}(x))) = P_{k_2}(x_q), \tag{5}$$

where $x_q \neq x$, $\hat{k} = (k_0, k_1, k_2)$.

In general the probabilities of such collisions vary depending on encryption scheme $(\mathcal{K}, E, D)$. A good candidate to minimize this probability is the IND-CPA secure one-time pad based encryption scheme constructed as follows:

– Key generation: $k \overset{R}{\leftarrow} \mathcal{K}$.
– Encryption: $E_k(x) := (r, f_k(r) \oplus x)$, where $f : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{X}$ is a pseudo-random function, $r \overset{R}{\leftarrow} \mathcal{X}$.
– Decryption: $D_k(r, c) := f_k(r) \oplus c$.

The scheme is simple, efficient and, according to [?], IND-CPA secure.

In order to use this scheme as $(\mathcal{K}, E, D)$ we require $k, r, x \in \{0,1\}^m$ and $f : \{0,1\}^m \times \{0,1\}^m \mapsto \{0,1\}^m$. Using this implementation of $(\mathcal{K}, E, D)$ we can rewrite (5) as

$$f_{P_{k_1}(x_q)}(r) \oplus f_{P_{k_1}(x)}(r) \oplus P_{k_2}(x) = P_{k_2}(x_q) \Leftrightarrow f_{P_{k_1}(x_q)}(r) \oplus f_{P_{k_1}(x)}(r) = P_{k_2}(x_q) \oplus P_{k_2}(x).$$

Consider the ideal case where instead of pseudo-random functions $f_{P_{k_1}(x_q)}, f_{P_{k_1}(x)}$ random functions ϕ, ψ are used. Then

$$\Pr[\phi(r) \oplus \psi(r) = P_{k_2}(x_q) \oplus P_{k_2}(x), x \neq x_q,\, r \overset{R}{\leftarrow} \mathcal{X}] = \frac{1}{2^m}.$$

The probability that the collision (5) will not occur is the probability of the inverse event or

$$\Pr[\phi(r) \oplus \psi(r) \neq P_{k_2}(x_q) \oplus P_{k_2}(x),\, x \neq x_q, r \xleftarrow{R} \mathcal{X}] = 1 - 2^{-m}.$$

In order to estimate the probability that there will be no collisions when equality (4) is checked for a set of different values $\{x_1, \dots, x_{t(m)})\}$, where $x_i \neq x_q$, $x_i \neq x_j$, $i \neq j$ and t is a positive polynomial, we note that in the ideal case, for each x_i the random function ϕ_i is chosen independently and thus the events that correspond to the collisions for each x_i are also independent. Therefore the probability that, when performing an exact select σ_{x_q} on values $\{x_1, \dots, x_{t(m)})\}$, no collisions occur is

$$(1 - \Pr[\phi(r) \oplus \psi(r) = P_{k_2}(x_q) \oplus P_{k_2}(x), x \neq x_q,\, r \xleftarrow{R} \mathcal{X}])^{t(m)} = (1 - 2^{-m})^{t(m)}.$$

Analogously, to each new query there corresponds a randomly chosen function ψ_i. The probability of event $\Re$ that corresponds to the absence of collisions when querying $t(m)$ values with $s(t)$ different queries is

$$\begin{aligned}\Pr(\Re) &= (1 - \Pr[\phi(r) \oplus \psi(r) = P_{k_2}(x_q) \oplus P_{k_2}(x), x \neq x_q,\, r \xleftarrow{R} \mathcal{X}])^{t(m)s(m)} \\ &= (1 - 2^{-m})^{t(m)s(n)}.\end{aligned}$$

The lower bound of probability $\Pr(\Re)$ can be estimated as

$$\Pr(\Re) = \left(1 - \frac{1}{2^m}\right)^{t(m)s(m)} > \left(1 - \frac{t(m)s(m)}{2^m}\right) = \left(1 - \frac{p(m)}{2^m}\right) > \left(1 - \frac{1}{p(m)}\right) \tag{6}$$

for sufficiently large m and positive polynomial p. The probability that there will be at least one collision is $1 - \Pr(\Re) < 1/p(m)$, which is negligible.

This estimation was performed for the case in which functions ψ, ϕ are chosen randomly and uniformly. If instead we use pseudo-random functions $f_{P_{k_1}(x_q)}, f_{P_{k_1}(x)}$, then analogously to Lemma A.1.4 (see Appendix A.1) it can be shown that the pair $(f_{P_{k_1}(x_q)}, f_{P_{k_1}(x)})$ is indistinguishable from the pair $(\psi,\, \phi)$. Suppose that there exist a set of values and a set of queries, such that probability $1 - \Pr(\Re)$ is non-negligible. Then using these sets we can build an algorithm that distinguishes between $(\psi,\, \phi)$ and $(f_{P_{k_1}(x_q)}, f_{P_{k_1}(x)})$ with non-negligible probability. Due to the polynomial sizes of the sets the algorithm works in polynomial time. That contradicts the indistinguishability of $(\psi,\, \phi)$ and $(f_{P_{k_1}(x_q)}, f_{P_{k_1}(x)})$. Therefore the probability of collision in the non-ideal case is also negligible. For real applications that means that with sufficient key lengths in most of the cases queries results will not contain any erroneous tuples. However, since the possibility of an error is not excluded, in some applications the client still may have to recheck the result set in order to ensure its correctness.

To get an impression on what can it mean for real applications consider the following example: If the encryption scheme $(\mathcal{K}, E, D)$ uses the random function ϕ, $m = 128$, and $t(m)s(m) = 10^{20}$ then according to (6) $\Pr(\Re) > 1 - 2.9 \cdot$

10^{-19}. That means that if Alice issues 10^{10} different exact selects $\sigma_{a.x_i}$, $i = 1,..,10^{10}$, $x_i \neq x_j$, $i \neq j$, and the attribute a of the queried relation contains 10^{10} different values, the probability that there will be at least one erroneous tuple in the result set is bigger than $1 - 2.9 \cdot 10^{-19}$. When instead of the random function a cryptographic primitive is used (e.g. HMAC-SHA-2 [**?**]) the actual probability might become lower, but still the presented technique may serve as a good approach for estimating the chances of an erroneous tuple being included in the result set.

In order to process an exact select $\sigma_{a_i.x}$ for a relation consisting of u tuples, Bob should only check whether equality (4) holds true for the value of attribute a_i of every tuple. Every check requires $O(1)$ operations and therefore processing of the query for the whole relation will be done in $O(u)$ operations.

Projection. Since the attributes of the relation are encrypted separately, in order to perform projection $\pi_{a_i,\ldots a_j}(a_1,\ldots a_l)$, Alice simply provides the name of the corresponding encrypted attributes and Bob performs $\pi_{a_i^E,\ldots a_j^E}(a_1^E,\ldots a_l^E)$ on the encrypted relation.

Cartesian product. Cartesian product of two encrypted relations is carried out just as with unencrypted relations - by returning all combinations of tuples of the encrypted relations. Again, this is possible because the attributes are encrypted separately and, as a result, ciphertexts can be concatenated.

Equijoin. The encryption scheme allows to perform equijoin as a combination of of Cartesian product and exact select. The feasibility of equijoin makes it possible to preserve the foreign key associations between the relations.

Union with duplicates. The union of two encrypted relations is performed by simply including the tuples of both relations in the resulting one. Note that duplicate removal is not possible because Bob has no means to determine on his own whether two ciphertexts correspond to identical or different tuples.

Exact update. Exact update is feasible due to the feasibility of exact select and separate encryption of the attribute values: Exact select allows to specify the tuples that should be updated and separate encryption allows to replace the encrypted attribute values of the tuples with the new ones. For example, consider the following update query: `UPDATE table1 SET salary = 3500 WHERE name = "John Smith"`. Alice transforms the query into tuple $(c, a_c^E, s, k_s, a_i^E)$ and sends it to Bob. The last three values allow to run the exact select query for getting the tuples to be updated. The first two values are the encryption of the new attribute value (3500) and the attribute name of the encrypted relation that corresponds to the one that should be updated (salary).

Exact delete. In order to run exact delete, Alice sends to Bob a triple (s, k_s, a_i^E) so that Bob can find the tuples to be deleted and then remove them from the encrypted relation.

Insert. To insert a tuple, Alice encrypts it and sends it to Bob, who simply appends it to the corresponding relation.

Logical operations. It is also possible to run operations with conditions consisting of several equalities connected by AND or OR. In case of a pair of equalities connected by a logical operation α, Alice sends a pair of triples connected by α to Bob: $(s_i, k^{s_i}, a_i^E)\,\alpha\,(s_j, k^{s_j}, a_j^E)$, where $\alpha \in \{\text{AND}, \text{OR}\}$. If $\alpha = \text{AND}$, Bob marks the tuple when (4) holds true for both triples. If $\alpha = \text{OR}$, Bob marks the tuple when (4) holds true for one of the triples (conditions built of more than two equalities connected by AND or OR can be treated in an analogous manner).

When there is a negation of the equality condition (NOT operation), Bob marks those tuples for which (4) does not hold.

5.2 Security Analysis

It is important to understand that when an encryption scheme is a privacy homomorphism the indistinguishability alone may not guarantee the security of the encrypted data. In some cases in order to perform an operation on the encrypted data Alice has to provide Bob with additional input that is dependent on the encryption key or the data itself. To see how this can become a problem, consider a database privacy homomorphism that encrypts a table and queries with an indistinguishably secure encryption scheme using two independently generated keys - one for the table and another for the queries. In order to provide Bob with the ability to run queries issued by Alice the encrypted table is appended with the key used for encrypting the queries and each query is appended with the key used for encrypting the table. When Alice issues a query she encrypts the corresponding SQL statement with the appropriate key and sends it to Bob. Bob, in turn, by using the key he got with the encrypted table and the key that he has received with the query decrypts the table and the query, runs the query and send the result to Alice. As a result, on the one hand we have a database privacy homomorphism that securely encrypts the table and the queries and supports all possible relational operations. But on the other hand that homomorphism gives no security at all after an operation was performed.

Therefore, for a privacy homomorphism it is always necessary to estimate the amounts of information disclosed when performing operations feasible under this homomorphism. Concerning our case, all the feasible operations except for exact select and those that are based on it (exact delete, exact update) do not provide Bob with any data that depends on the keys or on the encrypted table. As for exact select, we can show that when such queries are processed nothing except for the frequencies of queried attribute values is revealed to Bob. Intuitively, that means that when given an encrypted table and a sequence of queries Bob cannot get significantly more information about the table than when he is given the encrypted table, knows queried attributes and knows which tuples each query returns.

To express this formally, consider a database privacy homomorphism $(\mathcal{K}, E, D)$ that allows exact selects. Let m_i be a message to which Alice maps exact select q_i and which is then given to Bob so that he could process this query, and let $R_{E_k(T)}(m_i)$ be a set of references pointing to the tuples of encrypted table $E_k(T)$ that constitute a resulting set of query m_i.

Definition 7. *An exact select for database privacy homomorphism* $(\mathcal{K}, E, D)$ reveals nothing except for the frequencies of queried attribute values *if for every*

PPT algorithm A there exist a PPT algorithm A' such that for every table T, every polynomial p', every sequence of exact selects $q_1, \ldots, q_t$, $t \leq p'(n)$, every polynomially-bounded function f, every polynomial p and all sufficiently large n

$$\begin{aligned}&\Pr\left[A(m_1, \ldots, m_t, E_k(T)) = f(T)\right]\\ &\quad < \Pr\left[A'(R_{E_k(T)}(m_1), \ldots, R_{E_k(T)}(m_t), E_k(T)) = f(T)\right] + \frac{1}{p(n)}\end{aligned}$$

And for our database privacy homomorphism $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ we can formulate the following theorem

Theorem 3. *Database privacy homomorphism $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ reveals nothing except for the frequencies of queried attribute values.*

See Appendix A.2 for a proof of the theorem.

6 Indexing and Hashing

Processing of an exact select operation requires to sequentially scan all the tuples of the queried relation. In large databases, this is not efficient, which raises the question of indexing.

If the database is securely encrypted and cannot be decrypted when a query is processed, the usual indexing algorithms are no longer applicable. For example, it is impossible to perform ordering of the ciphertexts according to their plaintext values. Indeed, let $\prec$ be a binary relation defined on the set of ciphertexts and $E_k(x) \prec E_k(y)$ if and only if $x < y$. Then given the encryptions of x and y such that $x < y$, the adversary will be able to determine which ciphertext corresponds to which plaintext by simply checking if $E_k(x) \prec E_k(y)$. This rules out B+ trees and in general all indexing algorithms that rely on the ordering of the values of the indexed attribute. Hashing, on the other hand, remains feasible even if the database is securely encrypted.

Note, however, that identical plaintext values may be encrypted into different ciphertexts. As a result they can have different hashes and therefore belong to different hash buckets. This problem can be be solved by calculating the hash of the attribute value *before* the value is encrypted and send the ciphertext together with the corresponding hash to Bob. Then, when Alice issues an exact select query $\sigma_{a_i.x_q}$ with a_i being a hash-indexed attribute, she sends Bob not the triple (4) but a quadruple (q, k_q, a_i^E, h_q), where $h_q = H(x_q)$. Care should be taken when choosing a hash function for indexing encrypted data. If Bob knows the hash function H, from a given ciphertext c and corresponding hash h he can deduce some knowledge about the plaintext. E.g., by comparing $H(x)$ with the hash he got from Alice he can infer if the ciphertext c can be the encryption of x: $H(x) \neq h$ means that c is not the encryption of x. However, if Alice uses a pseudo-random function for computing hashes, knowledge of the secret key is required for calculating valid hash values. If only Alice knows the key, Bob is not able to get any information about the plaintexts by using their hash values.

In addition, if the values of an indexed attribute are highly skewed, such a hash function smooths out the distribution and uniformly fills the buckets.

The major drawback of such an approach is that it preserves security only if the indexed attributes do not contain any duplicate values. As a counter example consider two tables, one with the indexed attribute values being identical for all tuples and another where the values of the corresponding indexed attribute are unique. According to our indexing algorithm, Bob partitions attribute values into buckets based on hashes precomputed by Alice. Since the same values will produce the same hashes, the indexed attribute of the first table will produce single bucket and the indexed attribute of the second table with high probability will produce more than one. Using this information the adversary will be able to distinguish these two tables even if they are encrypted with a secure encryption scheme. According to Definition 4 the resulting scheme is not indistinguishably secure. An analogous analysis is applied to the more common case when the indexed column contains only some duplicates. Therefore, only key attributes can be securely indexed this way.

In order to prevent the adversary from learning statistical information about encrypted values, all buckets that contain pointers to the indexed values can be padded up to the same length and encrypted. In order to process an exact select query on a hashed attribute, Bob sends Alice the corresponding encrypted bucket. Alice decrypts the bucket and sends the decrypted pointers to Bob, who performs the exact select on the tuples to which the pointers from the received bucket refer.

7 Conclusion

In this paper we presented an encryption scheme that allows the secure outsourcing of a substantial subset of relational database operators: exact select, Cartesian product, projection, exact update, exact insert and exact delete. Our approach represents the first solution to the database outsourcing problem that is provably secure and supports such an extensive set of relational operators. We conclusively proved the security of our scheme and showed how to reduce the probability of having erroneous tuples in the answer to an exact select query to a negligible level. Moreover, we presented some thoughts on how to perform indexing and hashing in the context of encrypted database outsourcing. The development of efficient and secure hashing and indexing schemes for encrypted database outsourcing remains an important topic for future research.

A Proofs of security

A.1 Proof of Theorem 1 (IND-CPA security)

First we prove several lemmas. Note that the notation used for defining cryptographic primitives within the lemmas is not related to the cryptographic primitives that are used for building encryption scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$.

Let $(\mathcal{K}, E, D)$, where $\mathcal{K} = \{0,1\}^m$, $\mathcal{X} = \{0,1\}^m$, $E : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{Y}$, be IND-CPA secure encryption scheme and $F : \mathcal{K}^F \times \mathcal{X} \mapsto \mathcal{K}$, where $\mathcal{K}^F = \{0,1\}^n$, be a pseudo-random function.

Lemma A.1.1. *The scheme $(\mathcal{K}', E')$, where $E'_{k'}(x) := E_{F_{k'}(x)}(x)$, is IND-CPA secure.*[2]

Proof sketch. IND-CPA security of scheme $(\mathcal{K}', E')$ means that for all x, y, all PPT algorithm A with access to the encryption oracle, and every positive polynomial p for all sufficiently large n, the advantage

$$\mathrm{Adv} A_{xy}^{E'_{k'}} = |\Pr[A^{E'_{k'}}(E'_{k'}(x)) = 1] - \Pr[A^{E'_{k'}}(E'_{k'}(y)) = 1]| < \frac{1}{p(n)}.$$

For every algorithm A, we can define an oracle algorithm B that is identical to A except for the steps where A requests x from its oracle: B instead queries x from its oracle $O(\cdot)$ and then calculates $E_{O(x)}(x)$. Then, defining by $E_{F_{k'}}$ oracle $E'_{k'}(\cdot) = E_{F_{k'}(\cdot)}(\cdot)$, and by E_ϕ oracle $E_{\phi(\cdot)}(\cdot)$, where ϕ is a random function, the latter inequality can be rewritten as

$$\begin{aligned}\mathrm{Adv} A_{xy}^{E'_{k'}} \leq\ & |\Pr[B^{F_{k'}}(E_{F_{k'}(x)}(x)) = 1] - \Pr[B^{\phi}(E_{\phi(x)}(x)) = 1]| + |\Pr[A^{E_\phi}(E_{\phi(x)}(x)) = 1] \\ & - \Pr[A^{E_\phi}(E_{\phi(x)}(y)) = 1]| + |\Pr[B^{\phi}(E_{\phi(x)}(y)) = 1] - \Pr[B^{\phi}(E_{\phi(y)}(y)) = 1]| \\ & + |\Pr[A^{E_\phi}(E_{\phi(y)}(y)) = 1] - \Pr[A^{E_{F_{k'}}}(E_{F_{k'}(y)}(y)) = 1]| \\ & = \mathrm{Adv1} + \mathrm{Adv2} + \mathrm{Adv3} + \mathrm{Adv4}.\end{aligned}$$

Using the reduction argument and the facts that $(\mathcal{K}, E, D)$ is IND-CPA secure, F is a pseudo-random function, ϕ is a random function, it can be proved that $E_{F_{k'}(x)}(x)$ is IND-CPA indistinguishable from $E_{\phi(x)}(x)$, $E_{\phi(x)}(x)$ is IND-CPA indistinguishable from $E_{\phi(x)}(y)$, $E_{\phi(x)}(y)$ is IND-CPA indistinguishable from $E_{\phi(y)}(y)$, and $E_{\phi(y)}(y)$ is IND-CPA indistinguishable from $E_{F_{k'}(y)}(y)$:

$$\forall i,\ \mathrm{Adv}\, i < \frac{1}{4 \cdot p(n)},\ i \in \{1,2,3,4\} \Rightarrow \mathrm{Adv} A_{xy}^{E'} < \frac{1}{p(n)}.$$

Therefore $(\mathcal{K}', E')$ is IND-CPA secure. □

In the next lemma we prove IND-CPA security of the scheme analogous to the of from Lemma A.1.1, with the pseudo-random function F being substituted by the pseudo-random permutation $P : \mathcal{K}^p \times \mathcal{X} \mapsto \mathcal{X}$.

Lemma A.1.2. *If the encryption scheme $(\mathcal{K}, E, D)$, where $\mathcal{K} = \{0,1\}^m$, $\mathcal{X} = \{0,1\}^m$, $E : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{Y}$, is IND-CPA secure, then the scheme $(\mathcal{K}^p, E^p)$, where $\mathcal{K}^p = \{0,1\}^n$, $E^p_{k^p}(x) = E_{P_{k^p}(x)}(x)$ and $P : \mathcal{K}^p \times \mathcal{X} \mapsto \mathcal{K}$ is a pseudo-random permutation, is IND-CPA secure.*

[2] The scheme $(\mathcal{K}', E')$ is not an encryption scheme since the construction of E' does not suppose decryption.

Proof sketch. IND-CPA security of scheme $(\mathcal{K}^p, E^p)$ follows from Lemma A.1.1 and the fact that a pseudo-random function and a pseudo-random permutation with the same key and argument spaces are indistinguishable [**?**]. Otherwise assuming that the new scheme is not IND-CPA secure it can be shown that there exists a PPT oracle algorithm that is able to distinguish between F and P. □

Lemma A.1.3. *If the encryption scheme $(\mathcal{K}, E, D)$, where $\mathcal{K} = \mathcal{X}$, $E : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{Y}$, is IND-CPA secure, then the scheme $(\mathcal{K}^{pp}, E^{pp})$, where $\mathcal{K}^{pp} = \mathcal{K}^p \times \mathcal{K}^p$, $E^{pp}_{k^{pp}}(x) := E_{P_{k^1}(x)}(P_{k^2}(x))$, $k^{pp} = (k^1, k^2)$, $k^1, k^2 \in \mathcal{K}^p$ and $P : \mathcal{K}^p \times \mathcal{X} \mapsto \mathcal{K}$ is a pseudo-random permutation, is IND-CPA secure.*

Proof sketch. Suppose that the scheme $(\mathcal{K}^{pp}, E^{pp})$ is not IND-CPA secure. Then there exist plaintexts x, y and a PPT algorithm A that can distinguish between $E^{pp}_{k^1,k^2}(x)$ and $E^{pp}_{k^1,k^2}(y)$ with non-negligible probability. Considering that $P'_{k^1,k^2} := P_{k_1} \circ P^{-1}_{k^2}$ is a pseudo-random permutation, it can be shown that algorithm A applied to the scheme $(\mathcal{K}^p, E^p)$ from Lemma A.1.2, where $E^p_{k^1,k^2}(x) := E_{P'_{k^1,k^2}(x)}(x)$, can distinguish between $E^p_{k^1,k^2}(P_{k^2}(x))$ and $E^p_{k^1,k^2}(P_{k_2}(y))$ with non-negligible probability, since $E^p_{k^1,k^2}(P_{k^2}(x)) = E_{P_{k^1}(x)}(P_{k^2}(x))$ and $E^p_{k^1,k^2}(P_{k_2}(y)) = E^p_{k^1}(P_{k^2}(y))$. □

Lemma A.1.4. *If the encryptions schemes $(\mathcal{K}^1, E^1, D^1)$ and $(\mathcal{K}^2, E^2, D^2)$ are IND-CPA secure, then the encryption scheme $(\mathcal{K}^0, E^0, D^0)$, where $\mathcal{K}^0 = \mathcal{K}^1 \times \mathcal{K}^2 = \{0,1\}^n$ and*

$E^0_{k^0}(x) := (E^1_{k_1}(x), E^2_{k_2}(x))$, $k^0 = (k_1, k_2)$ and $k_1 \overset{R}{\leftarrow} \mathcal{K}^1$, $k_2 \overset{R}{\leftarrow} \mathcal{K}^2$, is also IND-CPA secure.

Proof sketch. IND-CPA security of scheme $(\mathcal{K}^0, E^0, D^0)$ means that for all x, y, all PPT algorithms A with access to the encryption oracle, and every positive polynomial p for all sufficiently large n, advantage $\mathrm{Adv}A^{E^0}_{xy} < 1/p(n)$:

$$\begin{aligned}\mathrm{Adv}A^{E^0}_{xy} &\leq |\Pr[A^{E^0_{k^0}}(E^1_{k_1}(x), E^2_{k_2}(x)) = 1] - \Pr[A^{E^0_{k^0}}(E^1_{k_1}(y), E^2_{k_2}(x)) = 1]| \\ &+ |\Pr[A^{E^0_{k^0}}(E^1_{k_1}(y), E^2_{k_2}(x)) = 1] - \Pr[A^{E^0_{k^0}}(E^1_{k_1}(y), E^2_{k_2}(y)) = 1]| \\ &= \mathrm{Adv1} + \mathrm{Adv2}\end{aligned}$$

Using the reduction argument and the facts that encryption schemes $(\mathcal{K}^1, E^1, D^1)$ and $(\mathcal{K}^2, E^2, D^2)$ are IND-CPA secure it can be proved that $(E^1_{k_1}(x), E^2_{k_2}(x))$ is IND-CPA indistinguishable from $(E^1_{k_1}(y), E^2_{k_2}(x))$ and $(E^1_{k_1}(y), E^2_{k_2}(x))$ is IND-CPA indistinguishable from $(E^1_{k_1}(y), E^2_{k_2}(y))$:

$$\forall i,\ \mathrm{Adv}\, i < \frac{1}{2 \cdot p(n)},\ i \in \{1, 2\} \Rightarrow \mathrm{Adv}A^{E^0}_{xy} < \frac{1}{p(n)}.$$

Therefore $(\mathcal{K}^0, E^0, D^0)$ is IND-CPA secure. □

Now we are ready to prove Theorem 1.

Proof. IND-CPA security of encryption scheme $(\hat{\mathcal{K}}, \hat{E}, \hat{D})$ follows from the IND-CPA security of encryption scheme $(\mathcal{K}, E, D)$, Lemma A.1.3 and Lemma A.1.4.

A.2 Proof of Theorem 3

Proof. Let $T = \{x_{ij}\}_{i \in \{1,\ldots,m\}, j \in \{1,\ldots,l\}}$ be a table with l attributes and m rows where the values are padded up to the same length and transformed to the binary format (for the brevity of notation we denote attribute a_j (a_j^E) as j (j^E)). Then $\hat{E}_{\hat{k}}(T) = \{\hat{E}_{\hat{k}}(x_{ij})\} = \{(t_{ij}, c_{ij})\}_{i \in \{1,\ldots,m\}, j \in \{1,\ldots,l\}}$ is table T in the encrypted form. By $E^q_{k_1,k_2}(\sigma_{j.x})$ we denote triple $(P_{k_2}(x), P_{k_1}(x), j^E)$ that corresponds to the encrypted query $\sigma_{j.x}$. In order to prove the theorem we have to show that for any PPT algorithm A there exists a PPT algorithm A' such that

$$\Pr\left[A((P_{k_2}(x_1), P_{k_1}(x_1), j_1^E), \ldots, (P_{k_2}(x_t), P_{k_1}(x_t), j_t^E), \hat{E}_{\hat{k}}(T)) = f(T)\right] < \frac{1}{p(n)} + \tag{$*$}$$

$$\Pr\left[A'((R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_1:x_1})), j_1^E), \ldots, (R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_t:x_t})), j_t^E), \hat{E}_{\hat{k}}(T)) = f(T)\right]$$

We build algorithm A' as a composition of algorithms A and B, where B receives $(R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_1:x_1})), j_1^E), \ldots, (R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_t:x_t})), j_t^E), \hat{E}_{\hat{k}}(T)$ as an input, generates a sequence of encrypted queries $(\alpha_1, \beta_1, j_1^E), \ldots, (\alpha_t, \beta_t, j_s^E)$ that is indistinguishable from the sequence $(P_{k_2}(x_1), P_{k_1}(x_1), j_1^E), \ldots, (P_{k_2}(x_s), P_{k_1}(x_s), j_s^E)$ and modifies $\hat{E}_{\hat{k}}(T)$ in such a way, that queries $(P_{k_2}(x_i), P_{k_1}(x_i), j_i^E)$ and $(\alpha_i, \beta_i, j_i^E)$ return tuples that reside at the same positions in the original and modified encrypted tables correspondingly.

As algorithm B begins, it randomly and uniformly chooses from $\mathcal{X}$ bit sequences α_1 and β_1 and for each $\hat{E}_{\hat{k}}(x_{ij_1}), i \in R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_1:x_1}))$ replaces its search tag with a new one computed as $t'_{ij_1} = E_{\alpha_1}(\beta_1)$. Then for each next pair $(R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_p:x_p})), j_p^E), p \in \{2, \ldots, t\}$ algorithm B checks if there exists $s < p$ such that $j_p^E = j_s^E$ and $R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_p:x_p}))$ contains the same references as $R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_s:x_s}))$. If no, then B randomly and uniformly chooses α_p from $\mathcal{X} \setminus \{\alpha_i \,|\, i < p\}$, β_p from $\mathcal{X} \setminus \{\beta_i \,|\, i < p\}$ and for each $\hat{E}_{\hat{k}}(x_{ij_p}), i \in R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_p:x_p}))$ replaces search tag with $t'_{i,j_p} = E_{\alpha_p}(\beta_p)$. If yes, then that means that s-th and p-th queries are identical. Therefore $\alpha_p = \alpha_s$, $\beta_p = \beta_s$ and the corresponding search tags are again replaced with $t'_{i,j_p} = E_{\alpha_p}(\beta_p)$. As the output B returns $(\alpha_1, \beta_1, j_1^E), \ldots, (\alpha_t, \beta_t, j_t^E)$, $\{(t'_{ij}, c_{ij})\}$ which is taken as the input for algorithm A.

Since random values $A'((R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_1:x_1})), j_1^E), \ldots, (R_{\hat{E}_{\hat{k}}(T)}(E^q_{k_1,k_2}(\sigma_{j_t:x_t})), j_t^E), \{(t_{ij}, c_{ij})\})$ and $A((\alpha_1, \beta_1, j_1^E), \ldots, (\alpha_t, \beta_t, j_t^E)), \{(t'_{ij}, c_{ij})\})$ are identically distributed we can rewrite $(*)$ as

$$\Pr\left[A((P_{k_2}(x_1), P_{k_1}(x_1), j_1^E), \ldots, (P_{k_2}(x_t), P_{k_1}(x_t), j_t^E), \{(t_{ij}, c_{ij})\}) = f(T)\right] < \tag{$**$}$$

$$\Pr\left[A((\alpha_1, \beta_1, j_1^E), \ldots, (\alpha_t, \beta_t, j_t^E)), \{(t'_{ij}, c_{ij})\}) = f(T)\right] + \frac{1}{p(n)}$$

Suppose now that there exist table T, queries $\sigma_{j_1.x_1}, \ldots, \sigma_{j_t.x_t}$ and function f such that inequality $(**)$ does not hold. Then by wrapping the expression $A(\ldots) = f(T)$ with algorithm U that outputs 1 when the equality holds and 0 otherwise and performing reductions similar to those that we used in the proof of Theorem 1 it can be shown that there exist a PPT algorithm that could distinguish between pseudo-random permutation P and a pemutation chosen randomly and uniformly from the set of all permutations defined on X. The existence of such algorithm will contradict to the pseudo-randomness of P what concludes the proof of the theorem.

B Comments on the Encryption Scheme Presented in [?]

Yang et al. propose their own security model for privacy-preserving query protocols. First, they introduce the notion of the *minimum information revelation* of exact select query q issued to table T, which is the set of coordinates of the cells satisfying the condition of the query. Denoting the minimum information revelation by $R_T(q)$ they present their version of the definition of the query protocol revealing nothing but $R_T(q)$:

Definition 8. *A one-round query protocol reveals nothing beyond the minimum information revelation if for any polynomial* poly() *and all sufficiently large* n*, there exists a PPT algorithm* S *(called a simulator) such that for any* $t < \mathrm{poly}(k)$*, any polynomial-size circuit family* $\{A_n\}$*, any polynomial* $p()$*, and any exact select queries* $q_1, \ldots, q_t$ *for the advantage defined as*

$$\begin{aligned}\mathrm{Adv}A = |&\Pr[A_n(q_1, \ldots, q_t, m_1, \ldots, m_t), E_k(T)) = 1] \\ &- \Pr[A_n(q_1, \ldots, q_t, S(R_{E_k(T)}(E'_{k'}(q_1)), \ldots, R_{E_k(T)}(E'_{E_{k'}(T)}(q_t)), E_k(T)))) = 1]|\end{aligned}$$

it holds that $\mathrm{Adv}A < 1/p(n)$*.*

However, this definition contains one serious flaw: It does not impose any requirements on the security of the encryption scheme that is used to encrypt table T. As an example, consider a protocol that performs no encryption at all and operates with plaintext tables and queries. In such protocol for any table T (query q_i) $E_k(T) = T$ ($m_i = q_i$) it is trivially to build simulator S that by observing $E_k(T)$ and $R_{E_k(T)}(m_1)), \ldots, R_{E_k(T)}(m_t))$ reconstructs queries $q_1, \ldots, q_t$ and returns them with $E_k(T)$ as the output. Clearly, with such simulator $\mathrm{Adv}A = 0$ – thus, the protocol which gives no security at all satisfies the proposed definition.

The encryption scheme and the querying algorithm proposed by Yang et al. exploits the approach similar to the one we proposed in Section 4. But by proving that the described query protocol satisfies Definition 8 Yang et al. claim that the protocol reveals only number of tuples sharing the queried value and the queried attribute. As we have just shown, this definition, actually, says nothing about the strength of the encryption and the level of security provided by the protocol.

Also, without any formal argumentation Yang et al. claim that their protocol returns those, and only those tuples that satisfy an issued exact select query. However, by applying the same reasoning as we did in Section 5.1 one can easily see that the protocol may allow erroneous tuples to be included in the resulting set.

It is worth to mention that the described issues as well address the query protocol with enhanced security, which Yang et al. construct to minimize the amount of information leaked when the table is being queried.