Query Log Attack on Encrypted Databases

Tahmineh Sanamrad and Donald Kossmann

Department of Computer Science,
Swiss Federal Institute of Technology,
Zurich, Switzerland
{Tahmineh.Sanamrad,Donald.Kossmann}@inf.ethz.ch

Abstract. Encrypting data at rest has been one of the most common ways to protect the database data against honest but curious adversaries. In the literature there are more than a dozen mechanisms proposed on how to encrypt data to achieve different levels of confidentiality. However, a database system is more than just data. An inseparable aspect of a database system is its interaction with the users through queries. Yet, a query-enhanced adversary model that captures the security of user interactions with the encrypted database is missing. In this paper, we will first revisit a few well-known adversary models on the data encryption schemes. Also, to model the query-enhanced adversaries we additionally need new tools, which will be formally defined. Eventually, this paper introduces query-enhanced adversary models which additionally have access to the query logs or interact with the database in different ways. We will prove by reduction that breaking a cryptosystem by a query-enhanced adversary is at least as difficult as breaking the cryptosystem by a common adversary.

Keywords: Database Encryption, Query Rewrite Function, Query-Enhanced Adversary Models

1 Introduction

In the recent past, there has been significant interest in processing queries on encrypted data without decrypting the data. The goal is to protect the confidentiality of data against honest but curious attackers who have powerful access rights on the machines that host and process the data; e.g., system administrators with root privileges. There has been a lot of work on proposing query-able data encryption schemes such as [1, 4–7, 11].

However, data is not the only source of information for the adversary in a database system. Interactions with a database through queries and transactions might also lead to confidentiality leaks. Thus, in order to prove the level of data confidentiality in a database system, it does not suffice to only look at the adversary models which solely operate on the data. Therefore, new and enhanced adversary models are required to capture the confidentiality of the database query logs. The goal in this paper is to introduce such enhanced adversary models that independent of the encryption scheme try to break the cryptosystem by looking at the query logs. So far there has been no single work dedicated to model the query log attack on encrypted databases.

In this paper we assume to have a client-server architecture, where the client is trusted and the database server is completely untrusted. We assume to have a thin encryption layer residing on a trusted party, either on the client itself or on a trusted security middleware. This thin encryption layer sits between the client and the untrusted database server. The main task of the encryption layer is to adjust the plaintext queries written by the clients in such a way that the encrypted database can process them. Upon receiving the query results, the encryption layer decrypts the result sets and sends them back to the client. Client is assumed to be unaware of the encryption layer in between, i.e. the encryption layer is transparent to the client. This model is assumed by almost all the database systems supporting encryption such as [2, 6, 8, 10, 12, 14].

Now the main problem is how secure are these rewritten queries that are submitted to the untrusted database server. The encryption layer rewrites the queries based on the encryption scheme used to encrypt the data. There are however a number of examples on how the queries may leak additional information to the adversary. For example, some queries can reveal secrets about their underlying encryption scheme. Additionally, the query log itself provides the adversary with additional information about the client submitting the query, timestamp and submission frequency.

Since in a database system, data and queries are intertwined we need to carefully separate these two during security analysis. In the beginning of this paper we will explain the assumed architecture in more detail. Then, we will revisit the well-known adversary models on encrypted data, namely Ciphertext-only Attack, Known-Plaintext Attack and Chosen-Plaintext Attack. More concretely, this paper makes the following contributions::

- First contribution of this paper is the introduction and formal definition of new tools required to analyze a query-enhanced adversary, such as a one-way Query Rewrite Function (QRF) that takes an encryption scheme and an original query from the client as input and outputs a rewritten query for the encrypted database, a Query Simulator (QSim) that can simulate rewritten queries just by looking at the query logs and the encrypted data, and a Query Generator (QGen) that can generate original queries out of plaintext values.
- Second contribution of this paper are the query-enhanced adversary models.
- Third contribution is to prove that regardless of the chosen encryption scheme, the security guarantee that a database encryption scheme can give against a query-enhanced adversary is at least as much as the security guarantee that the database encryption scheme can give against a common adversary.

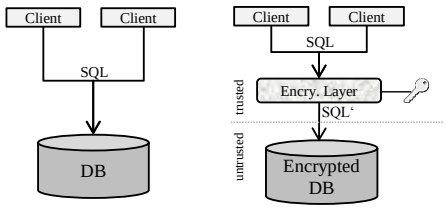
This paper is structured as follows: Section 2 revisits the common attacker scenarios on encrypted data. Afterward, Section 3 introduces the formal notions that will serve as building blocks in the query-enhanced adversary models. Eventually, Section 4 introduces the query-enhanced adversary models and proves by reduction that analyzing the security of the query logs can be replaced by the common security analysis on the underlying encryption scheme.

2 Preliminaries

In this section, we will first go over our assumptions regarding the system's architecture and security. Then, we revisit a few well-known attacks on the encryption schemes, namely Ciphertext-only Attack, Known-Plaintext Attack and Chosen-Plaintext Attack.

2.1 Client-Server Architecture

Figure 1a shows the traditional client-server architecture of running applications on top of a database system. The application or end user issues SQL statements to the database server. The database server executes these SQL statements and returns the results.



(a) Traditional (b) Encryption Layer

Fig. 1. Extended Client-server Database Architecture

Figure 1b shows an abstraction of the security-extended architecture. In this architecture, the application is unchanged and issues the same (unencrypted) SQL statements as in the traditional system of Figure 1a. In the following we will describe each component and its security assumptions in the security-extended architecture.

- **Encrypted DB** is the database server containing the encrypted data. In this paper the database server and all its components (e.g. main memory, CPU, ...) are assumed to be untrusted, however the server is not actively malicious.
- **Encryption Layer** implements the confidentiality. This layer can be seen as a trusted server namely a security middleware as in [6, 10, 13, 14], it can be seen as a secure co-processor within the database server as in [2, 3], or it can be an added module on the client's machine. The critical assumption is that this layer is trusted. The encryption layer is responsible for adjusting (rewriting) the queries to be processed on the encrypted data, and thereafter this layer needs to decrypt and if necessary post-process the query results.
- **Client** can be an end user or an application developer and is assumed to be trusted.

2.2 Attacks on Ciphers

In this section we go over the most common attack scenarios on the encryption schemes. These scenarios will be used in the reduction proofs in Section 4, where we define the query-enhanced adversaries.

Notation. Let x denote a plaintext value from the set of plaintext values, $\mathcal{X}$. Respectively, let y denote a ciphertext value from the set of ciphertext values, $\mathcal{Y}$. Let $\mathcal{E}nc(\tau, \mathbf{x})$ be the encryption function of an arbitrary encryption scheme, $\mathcal{ES}$ and τ the randomness element possibly required by the $\mathcal{E}nc$ function. $\mathbf{x} \xleftarrow{\$} \mathcal{X}$ simply means that a vector of plaintext elements have been chosen uniformly at random. The \$ implies a uniformly random selection from a set.

Experiment. An Experiment[1] is a probabilistic system that is connected to an Adversary[2], $\mathcal{A}$, on its left interface as shown in Figure 2 and at the right interface outputs a bit (0 or 1) indicating whether the Experiment is won. The Experiment is denoted as EXP throughout this paper.

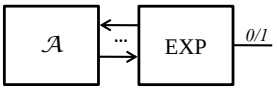


Fig. 2. An Experiment interacts with an adversary and in the end shows whether the adversary has succeeded.

Adversary. An adversary (attacker) is (or has access to) an algorithm that interacts with the experiment and its goal is to succeed in the experiment. The adversary is denoted as $\mathcal{A}$.

Advantage. The advantage of an adversary, $\mathcal{A}$, playing an experiment, EXP, is the success probability of $\mathcal{A}$ winning EXP, i.e. The experiment outputs 1 on its right interface. The advantage of $\mathcal{A}$ succeeding in EXP is denoted as $Adv^{\text{EXP}}(\mathcal{A})$.

Ciphertext-only Attack (CoA) In a *Ciphertext-only Attack* the attacker is given only a series of ciphertexts for some plaintext unknown to the attacker [9]. If an attacker can succeed in a *Ciphertext-only Attack*, it means that he could succeed in the following experiment:

Experiment 1 : $\mathbf{Exp}^{\text{CoA}}_{\text{ES}}(A)$

$\tau \xleftarrow{\$} \mathcal{T}$; $\mathbf{x} \xleftarrow{\$} \mathcal{X}$
$\mathbf{y} \leftarrow \mathcal{E}nc_{\text{ES}}(\tau, \mathbf{x})$
$x \xleftarrow{\$} A(\mathbf{y})$
if $x \in \mathbf{x}$ **then return** 1
else return 0

Experiment 1 chooses uniformly at random a vector of plaintext values, $\mathbf{x}$. It then encrypts them using the $\mathcal{E}nc_{\text{ES}}(\tau, \mathbf{x})$ function to obtain a vector of the corresponding ciphertext values, $\mathbf{y}$. The adversary then receives $\mathbf{y}$ and runs $A(\mathbf{y})$. The adversary succeeds

[1] An Experiment is also called a Game in some security literatures.
[2] An Adversary is also called a Winner in some security literatures.

if the plaintext value he returns, is in the plaintext vector chosen in the beginning by the experiment, in other words $\mathcal{A}$ succeeds if $x \in \mathbf{x}$. The advantage of the Ciphertext-only attacker on an arbitrary encryption scheme, $\mathcal{ES}$, is denoted as: $Adv_{\text{ES}}^{\text{CoA}}(A)$.

Known-Plaintext Attack (KPA) In a *Known-Plaintext Attack* the attacker is given a couple of plaintext-ciphertext pairs [9]. The goal of the attacker is to distinguish pairs of ciphertexts based on the plaintext they encrypt which were not initially given. Indistinguishability under *Known-Plaintext Attack* is captured through the following experiment:

Experiment 2 : $\text{Exp}_{\text{ES}}^{\text{IND-KPA}}(A)$

$\tau \xleftarrow{\$} \mathcal{T}; \mathbf{x} \xleftarrow{\$} \mathcal{X}$
$\mathbf{y} \leftarrow \mathcal{E}nc_{\text{ES}}(\tau, \mathbf{x})$
$(x_1, x_2) \xleftarrow{\$} \mathcal{X}\ s.t.\ x_1, x_2 \notin \mathbf{x}$
$b \xleftarrow{\$} \{0, 1\}$
$y_b \leftarrow \mathcal{E}nc_{\text{ES}}(\tau, x_b)$
$b' \xleftarrow{\$} A(\mathbf{x}, \mathbf{y}, (x_1, x_2), y_b)$
if $b' = b$ **then return** 1
else return 0

Experiment 2 chooses uniformly at random a vector of plaintext values, **x**. It then encrypts these values using the $\mathcal{E}nc_{\text{ES}}(\tau, \mathbf{x})$ function to obtain a vector of the corresponding ciphertext values, **y**. Then the experiment chooses randomly two plaintext values, $(x_1, x_2)\ s.t.\ x_1, x_2 \notin \mathbf{x}$, flips a coin and encrypts randomly one of them, $y_b = \mathcal{E}nc_{\text{ES}}(\tau, x_b)$. The attacker is then given both **x** and **y**, (x_1, x_2) and y_b. Based on y_b and the information he may extract from the known Plaintext-Ciphertext pairs,**x** and **y**, the attacker tries to find out whether x_1 or x_2 was encrypted. The probability that an attacker can succeed in this experiment is denoted as the IND-KPA advantage, $Adv_{\text{ES}}^{\text{IND-KPA}}(A)$ and is optimal if an attacker cannot do better than to randomly guess b, i.e. $Adv_{\text{ES}}^{\text{IND-KPA}}(A) \leq \frac{1}{2}$.

Chosen-Plaintext Attack (CPA) In a *Chosen-Plaintext Attack* the attacker is given plaintext-ciphertext pairs for the plaintext vector chosen by the attacker [9]. The goal of the attacker is to distinguish pairs of ciphertexts based on the plaintext they encrypt which were not initially chosen by the attacker. Therefore, the attacker, $\mathcal{A}$, consists of two functions $\mathcal{A} = (A_1, A_2)$. A_1 chooses a vector of plaintext values and A_2 tries to distinguish which plaintext was encrypted. Indistinguishability under *Chosen-Plaintext Attack* is captured through Experiment 3.

The adversary, using A_1, chooses a vector of plaintext values, **x**, and gives it to the experiment. The experiment encrypts these values using the $\mathcal{E}nc_{\text{ES}}(\tau, \mathbf{x})$ function to obtain a vector of the corresponding ciphertext values, **y**. Then, the adversary again

Experiment 3 : $\mathbf{Exp}_{\text{ES}}^{\text{IND-CPA}}(A)$

$\tau \xleftarrow{\$} \mathcal{T}$; $\mathbf{x} \leftarrow A_1(\mathcal{X})$
$\mathbf{y} \leftarrow \mathcal{E}nc_{\text{ES}}(\tau, \mathbf{x})$
$(x_1, x_2) \leftarrow A_1(\mathcal{X})\ s.t.\ x_1, x_2 \notin \mathbf{x}$
$b \xleftarrow{\$} \{0, 1\}$
$y_b \leftarrow \mathcal{E}nc_{\text{ES}}(\tau, x_b)$
$b' \xleftarrow{\$} A(\mathbf{x}, \mathbf{y}, (x_1, x_2), y_b)$
if $b' = b$ **then return** 1
else return 0

chooses two plaintext values, $(x_1, x_2)\ s.t.\ x_1, x_2 \notin \mathbf{x}$ and gives it to the experiment. The experiment flips a coin and encrypts randomly one of them, $y_b = \mathcal{E}nc_{\text{ES}}(\tau, x_b)$. The attacker is then given both **x** and **y**, (x_1, x_2) and y_b. Based on y_b and the information he may extract from his chosen Plaintext-Ciphertext pairs,**x** and **y**, the attacker tries to find out whether x_1 or x_2 was encrypted. The probability that an attacker can succeed in this experiment is denoted as the IND-CPA advantage, $Adv_{\text{ES}}^{\text{IND-CPA}}(A)$ and is optimal if an attacker cannot do better than to randomly guess b, i.e. $Adv_{\text{ES}}^{\text{IND-CPA}}(A) \leq \frac{1}{2}$.

3 New Definitions

In order to define new and query-enhanced adversary models we need additional functions that can capture the query transformations in the client-server architecture mentioned in Section 2.1.

Notation. Let the set of all queries a client sends to the encryption layer be denoted as $\mathcal{Q}_x$ and the set of rewritten-queries by the encryption layer to be processed on the encrypted database be $\mathcal{Q}_y$. Respectively, an original query submitted from the client is denoted as $q_x \in \mathcal{Q}_x$ and similarly a rewritten query submitted to the untrusted and encrypted database server is denoted as $q_y \in \mathcal{Q}_y$. $\mathcal{Q}_x^s$ denotes a subset of queries from $\mathcal{Q}_x$ and respectively, $\mathcal{Q}_y^s \subseteq \mathcal{Q}_y$.

Query. In this paper whenever we talk about a query, an SQL query is meant. However, our query-enhanced adversary models can be used also for other type of queries (e.g. information retrieval queries).

Running Example. Consider a relation *customer(id,name,age,salary,city)* in the encrypted database. The following query, q_x, is sent by the client to the database server:

```
SELECT SUM(salary) FROM customer
  WHERE city = 'Zurich' and age <= 30
```

Since the database is encrypted, the client's query is intercepted by the encryption layer and rewrites the query in a way, so that it can be processed by the encrypted database.

Query Tokens. These tokens are the pieces of data in the SQL query. Query tokens can be in plaintext, for example given the query of our running example, the query tokens are 'Zurich' and 30. Query Tokens can also be in ciphertext.

Query Rewrite Function (QRF). The *Query Rewrite Function* is a function that takes an original query q_x, and an encryption scheme $\mathcal{ES}$ as input and outputs the rewritten query q_y, $QRF(\mathcal{ES}, q_x) = q_y$. Depending on the encryption scheme, q_x will be rewritten differently. For example, having data encrypted in the database with the deterministic AES, the query of our running example, q_x, is rewritten as follows $QRF(AES, q_x) = q_y$:

```
SELECT salary,age FROM customer
  WHERE city = Enc_AES('Zurich')
```

Why the rewritten query having a deterministic AES scheme looks like above is not in the scope of this paper. In brief, deterministic AES is neither homomorphic (no support for the SUM aggregate function), nor order preserving (no support for the range condition), but it is deterministic and therefore equality-preserving.

Query Generator (QGen). The *Query Generator* is a function that takes a vector of plaintext query tokens, **x**, as input and generates a set of SQL queries Q_x^s using the tokens in **x**. *QGen* is independent of the encryption scheme, $\mathcal{ES}$. *QGen* has an inverse function, $\mathbf{x} = QGen^{-1}(Q_x^s)$ which outputs the plaintext query tokens of its input set. Figure 3, shows how *QGen* works for our running example.

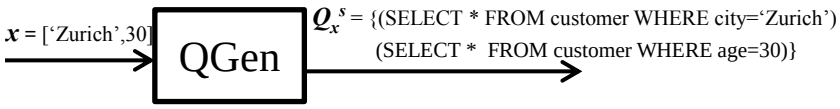


Fig. 3. QGen takes plaintext query tokens as input and builds a set of SQL queries out of them.

Query Simulator (QSim). Parallel to *QGen* but in the ciphertext space, the *Query Simulator* takes a vector of ciphertext query tokens, **y**, as input and generates a set of SQL queries, Q_y^s, using the tokens in **y**. *QSim* is independent of the encryption scheme and is allowed to use the query logs in Q_y. *QSim* has an inverse function too, $\mathbf{y} = QSim^{-1}(Q_y^s)$. Figure 4, shows how *QSim* works for our running example, assuming that $\mathcal{E}nc_{\text{AES}}(\text{'Zurich'}) = EG42KL23$.

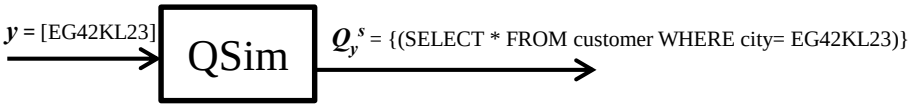


Fig. 4. QSim takes ciphertext query tokens as input and builds a set of rewritten SQL queries out of them.

Remark 1 *QGen and QSim are functions that do not change their input,* ***x*** *and* ***y*** *respectively, but wrap them in an SQL query.*

Figure 5, illustrates the above introduced functions. These functions will serve as building blocks in our adversary models and proofs.

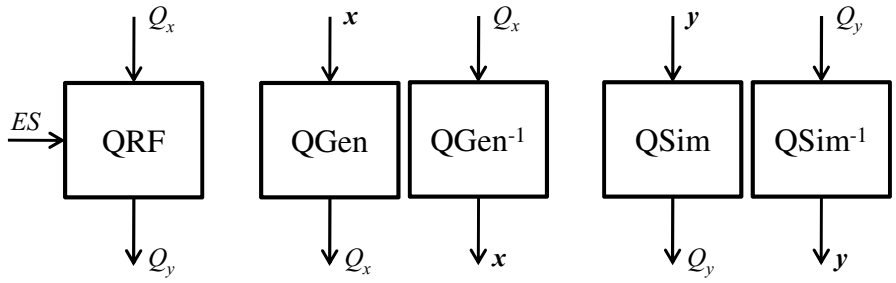


Fig. 5. Basic Functions used in the Query Log Adversary Models

In the remainder of this section, we introduce and formally define the adversary models. We prove that each adversary model on the query logs can be reduced to a known adversary model on the underlying encryption scheme.

4 Adversary Models

Database Adversary: Let us define the database adversary, A^{DB}, to be an adversary that has access to everything on the database server, namely the set of rewritten queries, $\mathcal{Q}_y$, the encrypted data, $\mathcal{Y}$, and eventually the encrypted result sets which are the result of running $\mathcal{Q}_y$ on $\mathcal{Y}$. We also assume that the database schema is public, i.e. a database adversary knows about the tables, attributes, attribute types, foreign keys and so on.

Proof by Reduction: To prove that Problem B (with unknown complexity) is at least as hard as Problem A (with known complexity), one solves Problem A using the solver of Problem B. It suffices to find an efficient[3] transformation, ϕ between the Solver of Problem B, $\mathcal{T}$ into the Solver of Problem A, $\mathcal{S}$, i.e. $\mathcal{S} = \phi(\mathcal{T})$.

4.1 Query-Only Attack

Query-only Attack (abbreviated as QoA) is when the database adversary A^{DB} has only access to the rewritten query logs namely, $\mathcal{Q}_y$. The advantage of a database adversary, A^{DB}, to succeed in a *Query-only Attack*, is defined as his probability to win the Experiment 4:

$$Adv_{\text{ES}}^{\text{QoA}}(A^{\text{DB}}) = Pr[Exp_{\text{ES}}^{\text{QoA}}(A^{\text{DB}}) = 1]$$

The experiments $Exp_{\text{ES}}^{\text{QoA}}(A^{\text{DB}})$ is defined as follows:

In the following we will prove that the *Ciphertext-only Attack* discussed in detail in Section 2 can be reduced to a Query-only Attack.

[3] Polynomial-time in the size of the input

Experiment 4 : $\mathbf{Exp}^{\text{QoA}}_{\text{ES}}(A^{\text{DB}})$

$\mathbf{x} \xleftarrow{\$} \mathcal{X}$
$Q_x^S \leftarrow QGen(\mathbf{x})$
$Q_y^S \leftarrow QRF(\mathcal{ES}, Q_x^S)$
$x_t \xleftarrow{\$} A^{\text{DB}}(Q_y^S)$
if $x_t \in \mathbf{x}$ **then return** 1
else return 0

Lemma 1 *Given an encryption scheme $\mathcal{ES}$ and a subset of rewritten queries, $\mathcal{Q}_y^S \in \mathcal{Q}_y$, $\mathcal{ES}$ is at least as safe against Query-only Attack as $\mathcal{ES}$ is safe against Ciphertext-only Attack.*

$$Adv_{ES}^{QoA}(A^{DB}) \leq Adv_{ES}^{CoA}(A)$$

Proof. Let $\mathcal{ES}$ be an arbitrary encryption scheme. Suppose A^{DB} is an adversary with non-trivial *QoA* advantage against $\mathcal{ES}$. We construct a *Ciphertext-only* adversary A against $\mathcal{ES}$. As per definition in the Ciphertext-only experiment1, A is given a vector of encrypted values, $\mathbf{y}$. A runs $\mathcal{QS}im(\mathbf{y})$ to simulate Q_y^S. Eventually, A runs $A^{\text{DB}}(Q_y^S)$. A's communication with A^{DB} mimics the QoA experiment. Clearly, A is efficient since *QSim* is sublinear to the size of its input.

4.2 Known-Query Attack

Known-Query Attack (abbreviated as KQA) is when the database adversary A^{DB} has access to a number of (q_x, q_y) pairs. For example assume q_x is:

```
SELECT SUM(salary) FROM customer
  WHERE city = 'Zurich' and age <= 30
```

then q_y using a deterministic $\mathcal{ES}$ will be something like:

```
SELECT salary, age FROM customer
  WHERE city = Enc_ES('Zurich')
```

Remark 2 *Query Logs additionally provide a database adversary with information about the clients submitting the queries, timestamp of the query submitted and their frequency. An adversary that has background knowledge about the business logic can use these additional log information to guess the original queries submitted by the clients. In general, statistical attacks on query logs can be classified as KQA.*

The advantage of a database adversary, A^{DB}, to succeed in a *Known-Query Attack*, is defined as his probability to distinguish the rewrite of two queries with the same structure as shown in the Experiment 5.

$$Adv_{\text{ES}}^{\text{IND-KQA}}(A^{\text{DB}}) = Pr[Exp_{\text{ES}}^{\text{IND-KQA}}(A^{\text{DB}}) = 1]$$

The experiments $Exp_{\text{ES}}^{\text{IND-KQA}}(A^{\text{DB}})$ is defined as follows:

Experiment 5 : $\mathbf{Exp}_{\text{ES}}^{\text{IND-KQA}}(A^{\text{DB}})$

$Q_x^S \xleftarrow{\$} Q_x$
$Q_y^S \leftarrow QRF(\mathcal{ES}, Q_x^S)$
$(q_x^1, q_x^2) \xleftarrow{\$} Q_x \; s.t. \; q_x^1, q_x^2 \notin Q_x^S$
$b \xleftarrow{\$} \{0, 1\}$
$q_y^b \leftarrow QRF(\mathcal{ES}, q_x^b)$
$b' \xleftarrow{\$} A(Q_x^S, Q_y^S, (q_x^1, q_x^2), q_y^b)$
if $b' = b$ **then return** 1
else return 0

Lemma 2 *Given an encryption scheme $\mathcal{ES}$ and a set of original and rewritten query pairs, $(\mathcal{Q}_x^S, \mathcal{Q}_y^S)$, $\mathcal{ES}$ is at least as safe*[4] *against Known-Query Attack as $\mathcal{ES}$ is safe against Known-Plaintext Attack.*

$$Adv_{ES}^{IND\text{-}KQA}(A^{DB}) \leq Adv_{ES}^{IND\text{-}KPA}(A)$$

Proof. Let $\mathcal{ES}$ be an arbitrary encryption scheme. Suppose A^{DB} is an adversary with non-trivial *IND-KQA* advantage against $\mathcal{ES}$. We construct an *IND-KPA* adversary, A, against $\mathcal{ES}$. As per definition in the IND-KPA experiment (Experiment 2), A is given a set of plaintext-ciphertext pairs. A has also access to a limited QRF that only works for Q_x^S. A first runs $QGen(\mathbf{x}) = Q_x^S$ and then $QRF(\mathcal{ES}, Q_x^S) = Q_y^S$. Additionally, A receives x_1, x_2 and y_b, so he constructs $q_x^1 = QGen(x_1)$, $q_x^2 = QGen(x_2)$ and $q_y^b = QSim(y_b)$. Eventually, A runs $A^{\text{DB}}(Q_x^S, Q_y^S, (q_x^1, q_x^2), q_y^b)$. A's communication with A^{DB} mimics the IND-KQA experiment. Clearly, A is efficient since *QGen*, *QRF* and *QSim* are linear to the size of their input.

Indistinguishability against *Known-Plaintext Attack* (Section 2) can be reduced to indistinguishability against *Known-Query Attack* as Lemma 2 shows.

4.3 Chosen-Query Attack

Chosen-Query Attack (abbreviated as CQA) is when the database adversary, A^{DB}, has access to a Query Rewrite Function, $QRF(\mathcal{ES})$. For example, an adversary that can send arbitrary queries to the encryption layer and see the rewritten queries on the other end. The advantage of a database adversary, $A^{\text{CQA}} = \left(A^{\text{CQ}}, A^{\text{DB}}\right)$, to succeed in a *Chosen-Query Attack*, is defined as his probability to distinguish the rewrite of his chosen queries as shown in Experiment 6.

$$Adv_{\text{ES}}^{\text{IND-CQA}}(A^{\text{CQA}}) = Pr[Exp_{\text{ES}}^{\text{IND-CQA}}(A^{\text{CQA}}) = 1]$$

The indistinguishability against *Chosen-Plaintext Attack* (see Section 2) can be reduced to indistinguishability against *Chosen-Query Attack* as Lemma 3 suggests.

[4] Safe means indistinguishable in this experiment.

Experiment 6 : $\mathbf{Exp}_{\mathrm{ES}}^{\text{IND-CQA}}(A^{\mathrm{CQA}})$

$Q_x^S \xleftarrow{\$} A^{\mathrm{CQ}}(Q_x)$
$Q_y^S \leftarrow QRF(\mathcal{ES}, Q_x^S)$
$(q_x^1, q_x^2) \leftarrow A^{\mathrm{CQ}}(Q_x)\ s.t.\ q_x^1, q_x^2 \notin Q_x^S$
$b \xleftarrow{\$} \{0,1\}$
$q_y^b \leftarrow QRF(\mathcal{ES}, q_x^b)$
$b' \xleftarrow{\$} A(Q_x^S, Q_y^S, (q_x^1, q_x^2), q_y^b)$
if $b' = b$ **then return** 1
else return 0

Lemma 3 *Given an encryption scheme $\mathcal{ES}$ and a set of original and rewritten query pairs, $(\mathcal{Q}_x^S, \mathcal{Q}_y^S)$ where Q_x^S has been chosen by the adversary, $\mathcal{ES}$ is at least as safe against a Chosen-Query Attack as $\mathcal{ES}$ is safe against a Chosen-Plaintext Attack.*

$$Adv_{ES}^{\textit{IND-CQA}}(A^{\textit{CQA}}) \leq Adv_{ES}^{\textit{IND-CPA}}(A)$$

Proof. Let $\mathcal{ES}$ be an arbitrary encryption scheme. Suppose A^{CQA} is an adversary with non-trivial *IND-CQA* advantage against $\mathcal{ES}$. We construct a *CPA* adversary A against $\mathcal{ES}$. As per definition in the IND-CPA experiment (Experiment 3), A is given a set of plaintext-ciphertext pairs where plaintexts have been chosen by the attacker. A has also access to a full-fledged QRF. A first runs $QGen(\mathbf{x}) = Q_x^S$ and then $QRF(\mathcal{ES}, Q_x^S) = Q_y^S$. Additionally, A receives x_1, x_2 and y_b, so he constructs $q_x^1 = QGen(x_1)$, $q_x^2 = QGen(x_2)$ with the same structure and $q_y^b = QSim(y_b)$.
Eventually, A runs $A^{\mathrm{DB}}(Q_x^S, Q_y^S, (q_x^1, q_x^2), q_y^b)$. A's communication with A^{CQA} mimics the IND-CQA experiment. Clearly, A is efficient since *QGen*, *QRF* and *QSim* are linear to the size of their input.

5 Conclusion

In this paper we have shown why it is important to consider additional and enhanced adversary models when analyzing the security of an encrypted database. The reason is because an encrypted database does not only consist of data but also queries and therefore, the security of the query logs are as important as the security of the data. Along the way, we have introduced a few notions and tools such as a Query Rewrite Function, a Query Generator and a Query Simulator to be used in our query-enhanced adversary models. In the end, we proved by reduction that breaking a database encryption using a query-enhanced adversary is at least as hard as breaking the underlying encryption scheme using a normal adversary. More concretely, we could show in this paper that:

- An encrypted database is at least as secure against a Query-only Attack as its underlying encryption scheme is secure against a Ciphertext-only Attack
- An encrypted database is at least as secure against a Known-Query Attack as its underlying encryption scheme is secure against a Known-Plaintext Attack
- An encrypted database is at least as secure against a Chosen-Query Attack as its underlying encryption scheme is secure against a Chosen-Plaintext Attack

As already mentioned in the introduction, there are a dozen of database encryption systems and schemes proposed in different communities. Nevertheless, adversary models that capture the query log security have never been defined or proposed before. As a venue for future work, the conclusions in this paper can be easily used to analyze the query log security for any existing or upcoming database encryption system or scheme.

References

1. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *SIGMOD*, pages 563–574, 2004.
2. A. Arasu and S. Blanas. Orthogonal security with cipherbase. In *CIDR*, 2013.
3. S. Bajaj and R. Sion. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In *SIGMOD*, pages 205–216, 2011.
4. A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, pages 224–241, 2009.
5. A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, pages 578–595, 2011.
6. E. Damiani, S. De Capitani Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *CCS*, pages 93–102, 2003.
7. H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, pages 216–227, 2002.
8. S. Hildenbrand, D. Kossmann, T. Sanamrad, C. Binnig, F. Faerber, and J. Woehler. Query processing on encrypted data in the cloud. Technical Report 735, Department of Computer Science, Swiss federal Institute of Technology Zurich, 2011.
9. J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC Press, 2008.
10. R. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *SOSP*, pages 85–100, 2011.
11. R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–178, 1978.
12. T. Sanamrad, L. Braun, D. Kossmann, and V. Ramarathnam. POP: A new encryption scheme for dynamic databases. Technical Report 782, Department of Computer Science, Swiss federal Institute of Technology Zurich, 2013.
13. R. Sion. Secure data outsourcing. In *VLDB*, pages 1431–1432, 2007.
14. S. Tu, M. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. In *VLDB*, pages 289–300, 2013.