# Optimized and Controlled Provisioning of Encrypted Outsourced Data

Anis Bkakria
Télécom Bretagne
Rennes, France
anis.bkakria@telecom-bretagne.eu

Andreas Schaad
SAP AG
Karlsruhe, Germany
andreas.schaad@sap.com

Florian Kerschbaum
SAP AG
Karlsruhe, Germany
florian.kerschbaum@sap.com

Frederic Cuppens
Télécom Bretagne
Rennes, France
frederic.cuppens@telecom-bretagne.eu

Nora Cuppens-Boulahia
Télécom Bretagne
Rennes, France
nora.cuppens@telecom-bretagne.eu

David Gross-Amblard
Université de Rennes 1
Rennes, France
david.gross-amblard@irisa.fr

## ABSTRACT

Recent advances in encrypted outsourced databases support the direct processing of queries on encrypted data. Depending on functionality (i.e. operators) required in the queries the database has to use different encryption schemes with different security properties. Next to these functional requirements a security administrator may have to address security policies that may equally determine the used encryption schemes. We present an algorithm and tool set that determines an optimal balance between security and functionality as well as helps to identify and resolve possible conflicts. We test our solution on a database benchmark and business-driven security policies.

## Categories and Subject Descriptors

H.2.0 [**Database Management**]: General—*Security, Integrity, and Protection*; D.4.6 [**Operating Systems**]: Security and Protection—*Access control*

## Keywords

Encrypted Database, Policy Configuration, Encryption Algorithm

## 1. INTRODUCTION

The IT world is facing an architectural shift where storage as well as processing capabilities are offered by cloud providers. We observe large platforms operated, for example, by Amazon, SAP or Microsoft successfully providing infrastructure, database-as-a-service, and entire cloud application offerings to some of the world's largest companies. Naturally, data security is a main concern and one general answer is to use at-rest encryption technology to protect outsourced data. This, however, renders any offerings such as database-as-a-service useless when assuming that cloud companies may not be fully trusted [26] as encrypted data cannot be directly processed. A bizarre architectural setup is the result where customers have to encrypt large data sets before provisioning them to the cloud, but then retrieve them back to on-premise and decrypt them if they want to run any complex queries. How to securely store as well as process data in a cloud is thus a major question for companies willing to migrate to and benefit from cloud offerings.

However, encryption schemes have been proposed recently that allow to execute particular query operators over encrypted data and recent work by [24] shows that the general direct processing of encrypted data is an achievable goal, something recently confirmed in a larger industrial perspective [15]. Following the idea of encrypting cleartext in so called "onions" allows to balance and match data processing functionality, i.e. each layer of an onion supports some SQL operations, with security, i.e. an onion structure introduces a total order with respect to the security properties of the chosen schemes. Yet, it is not practical to encrypt all columns in a table with the same onion structure. For example, columns may not require any encryption as they do not contain any sensitive material. Other columns may, for company specific compliance regulations, require to always be encrypted using a specific scheme when outsourced.

We believe that in order to further promote the wider industrial adoption of directly processing encrypted data, a more flexible configuration management is required before outsourcing the data from on-premise to a database-as-a-service cloud. In this paper, we first present a policy-based configuration framework for encrypted data allowing the security administrator to specify the security policy to be applied over the outsourced data. Second, we propose an algorithm allowing to detect conflicts between security and utility requirements. Third, we prove that selecting the optimal combination of encryption schemes that fit the defined policies with respect to the data owner's functional requirements (e.g. SQL that should be executed over the encrypted data) is NP-hard. Fourth, we therefore propose a heuristic, polynomial-time algorithm for finding a combination of en-

*SACMAT'14,* June 25–27, 2014, London, Ontario, Canada.
Copyright 2014 ACM 978-1-4503-2939-2/14/06 ...$15.00.
http://dx.doi.org/10.1145/2613087.2613100.

cryption schemes that satisfies a policy $P$ and provides the best security level.

The rest of this paper is organized as follows, Section 2 describes the problem treated in this paper. Section 3 presents the modeling of the used system and the modeling of the policy to be applied over the outsourced database. We show, for a given policy, how to detect the conflict between security and utility requirements involved in the policy and how to choose the combination of encryption schemes that enforces it. Section 4 presents a use case showing the application and the benefits of our approach in practice. Section 5 discusses related work. Finally, Section 6 reports our conclusions.

## 2. PROBLEM DESCRIPTION

### 2.1 Adjustable Database Encryption

Encrypted databases can execute SQL queries over encrypted data. In this case data is never decrypted inside the database server, but always remains encrypted. The key to the encryption and decryption functions solely resides at the client.

The main idea to processing queries in this way is property-preserving encryption. In property-preserving encryption a function $f(E(x), E(y))$ on ciphertexts $E(x)$, $E(y)$ returns the same result as $f(x, y)$. Hacigümüs et al. have described this concept for deterministic encryption and equality as a function [16]. They realized that many database operators, particularly selection and join, often use equality. Each data value is separately deterministically encrypted. Those database operators can then be used unmodified on encrypted data.

A limitation of the initial approach was that inequality comparisons (range queries) were insufficiently supported. Agrawal et al. introduced order-preserving encryption [2]. Order-preserving encryption is property-preserving encryption for greater-than-or-equal comparisons. Using order-preserving encryption one can implement a large subset of SQL queries.

The security of order-preserving encryption and even deterministic encryption is still much debated. It is therefore better to choose the most secure encryption for a set of queries. If this set is unknown, then all data needs to be encrypted order-preservingly. Popa et al. presented a solution to this: adjustable (onion) encryption. Each data value is encrypted order-preservingly. This ciphertext is encrypted deterministically and the result is finally encrypted using standard randomized encryption secure against chosen plaintext attacks. Before a query is executed it is analyzed for the required encryption levels and the data values are adjusted (decrypted) to these levels. Hence, the most secure encryption can be chosen automatically.

### 2.2 Functional Requirements

As already mentioned the set of queries executed on the database pose a set of functional requirements. These requirements are captured as the functions executed on the ciphertext by the database operators.

In many cases a large subset of the queries to be executed is known. For example, when an application uses the database, one can analyse this application and extract the queries (maybe except for parameters). In many cases one can simply resort to the prepared SQL statements.

If this subset of queries is known in advance, then it would be unwise to adjust the encryption during run-time. Although the adjustment process is performed only once, it can be quite costly. Each data value of an entire column needs to be decrypted which can sum to several MByte or even GByte of data.

Instead, the database can be encrypted to a "prepared" state and the adjustment process avoided. This leads to a significant shortening of the phase from a cold to a hot database. Real systems can go faster into production.

Our approach is the first to support this analysis. We choose the appropriate encryption levels depending on the functional requirements of a set of queries.

### 2.3 Security Levels

The encryption levels of adjustable encryption correspond to different security levels. We claim that randomized encryption is at least as secure as deterministic encryption which is at least as secure as order-preserving encryption. We argue as follows.

Randomized encryption (RND) is semantically secure, i.e., it is secure against chosen plaintext attacks. We use AES in CBC for this encryption level. Clearly, then chosen plaintexts attacks are prevented.

Deterministic encryption (DET) allows chosen plaintext attacks, if the key is known or there is an encryption oracle. We only need symmetric encryption in encrypted database, such that it may be difficult to obtain the key or construct such an oracle. If a plaintext is encrypted and stored more than once, deterministic encryption also allows frequency attacks as in [18]. While not necessary, this may often – if not almost always – be the case in real databases. We therefore claim that deterministic encryption is less secure than randomized encryption. We use Pohlig-Hellman encryption, a symmetric key RSA variant, for this encryption level, in order to support proxy re-encryption [20].

Order-preserving encryption (OPE) is also deterministic, such that all attacks on deterministic encryption also work for order-preserving encryption. In addition, it preserves the order, which may enable many more attacks. It was concluded that order-preserving encryption leaks at least half of the plaintext bits [29]. Clearly, order-preserving encryption is the least secure choice. We use the scheme by Boldyreva et al. [5, 6] for this encryption level, which has been proven to be the optimally secure, immutable, order-preserving encryption scheme.

Next to these encryption levels we use homomorphic encryption (HOM) for aggregation. Specifically, we use Paillier encryption [22]. Homomorphic encryption is secure against chosen plaintext attacks as is randomized encryption. Since for processing queries both ciphertexts need to be offered in parallel, they can be safely assumed to provide the same security level. Furthermore, similar to onion encryption, homomorphic encryption can be downgraded to deterministic encryption. As in the approach by Bellare et al.[4], we can choose a deterministic randomization parameter. For downgrading we can simply select one ciphertext among the set of identical plaintexts. This has the added benefit that dictionary compression is as effective as on plaintext data [19].

## 2.4 Security Requirements and The Need for Policy Configuration

Considering the security levels from Section 2.3 The data owner may realize that certain queries may put his data at risk. These queries may adapt the encryption level to an unsafe state, e.g. order-preserving encryption, for a certain set of data. Even certain security standards, such as PCI-DSS, may require certain encryption levels.

Therefore the data owner may want to set certain policies on which encryption levels are allowed. He may want to prevent specific data from ever reaching a specific encryption state. For this he needs the approach for specifying policies we propose in this paper.

## 2.5 Policy Enforcement

The specified policies need to be enforced in the encrypted database. There is a crucial insight that enables prevention of certain encryption levels. If an encryption level is not present, it cannot be decrypted to. And vice versa, if an encryption should not be decrypted to, it does not need to be present. We therefore omit the encryption levels prevented by our policy. If one should not be able to decrypt to order-preserving encryption, the data value will not be encrypted order-preservingly. This has the positive side effect that ciphertexts may get smaller and encryption is more efficient.

The question remains what to do with queries that functionally require an encryption level that is prohibited by the security policy. In this case one ships the ciphertexts to the client, decrypts and executes the query on the client. The client query analysis algorithm of Kerschbaum et al. based on relational algebra, allows splitting a query into a local and a remote part [21]. This way only the minimally necessary part of the query according to the security policy will be executed on the client.

## 3. POLICY CONFIGURATION

In this section, we firstly present the modeling of the system and the specification of the policy. Afterwards, we present an algorithm allowing to detect conflicts between the constraints of the policy. We then propose an efficient algorithm allowing to enforce the policy while resolving the detected conflicts.

## 3.1 System modeling

In our approach, data to be outsourced is stored in a relational database $\mathcal{D}$, which is composed of a collection of relational tables $\mathcal{T} = \{T_1, \cdots, T_n\}$, with each of these relational tables $T_i$ containing a collection of attributes $\mathcal{A}_{T_i} = \{a_{1,i}, a_{2,i}, \cdots\}$. The system contains a toolbox $\mathcal{E}$ composed of a set of $m$ encryption schemes $\{E_1, \cdots, E_m\}$ that can be used to protect outsourced data. Each encryption scheme $E_i \in \mathcal{E}$ is characterized by a security level $l_i$ that provides and a set of functionalities $F_i \subseteq \mathcal{F}$ that satisfies. Let $\mathcal{F}$ be the set of functional requirements that can be required over the data to be outsourced and $\mathcal{L}$ be the set of security levels provided by $\mathcal{E}$.

## 3.2 Policy modeling

We model, in a quite simple and powerful way, the requirements defined by the data owner. Those requirements are expressed through security and utility constraints. Security constraints are composed of confidentiality constraints and security threshold constraints.

*Definition 1.* (Confidentiality constraint) Given a relational table $T_i \in \mathcal{T}$ containing a list of attributes $\mathcal{A}_{T_i}$, a confidentiality constraint defined over $T_i$ is a singleton set $CC = \{a\}$, where $a \in \mathcal{A}_{T_i}$.

Semantically speaking, a confidentiality constraint $CC$ states that the value assumed by the attribute in $CC$ is considered sensitive and therefore must be protected.

*Definition 2.* (Security threshold constraint) Given a relational table $T_i \in \mathcal{T}$ and an attribute $a \in \mathcal{A}_{T_i}$, a security threshold constraint $TC_a$ over the attribute $a$ is a security level $l$ in $\mathcal{L}$. A security threshold constraint defined over the attribute $a$ is well defined *iff* there exists a confidentiality constraint $CC$ such that $a \in CC$.

Security threshold constraints allow the data owner to specify a security level threshold for each sensitive attribute. The semantics of a security threshold constraint $TC$ is that the security level of the sensitive attribute $a$ must be at least as much secure as the security level $l$ of $TC$.

*Definition 3.* (Utility constraint) Given a relational table $T_i \in \mathcal{T}$ and an attribute $a \in \mathcal{A}_{T_i}$, an utility constraint $UC_a$ over the attribute $a$ is a set of functionality $F_a = \{f_1, \cdots, f_n\}$, where $F_a \subseteq \mathcal{F}$.

Confidentiality protection is provided at the expense of data utility. A utility constraint offers the data owner the ability to require that some functionalities on his data must be provided, otherwise the data is useless.

## 3.3 Policy conflict detection

Policy conflicts occur when the objectives of two or more constraints cannot be simultaneously satisfied. Conflict detection aims at checking whether a set of constraints contains conflicts. In our case, conflicts may occur between security constraints and utility constraints, more precisely, between security threshold constraints and utility constraints. To detect the conflicts, there are two steps. First, we must get for each security level $l \in \mathcal{L}$, the set of functionalities $F_l$ which are satisfied by encryption schemes providing security levels that are at least as much secure as $l$. Then, for each sensitive attribute having $TC_a = l_a$ as a security threshold constraint and $UC_a = F_a$ as an utility constraint, we check if the set of functionalities $F_{l_a}$ we got from the previous step for the level $l_a$ is a superset of $F_a$, and if not, we deduce that there is a conflict between $TC_a$ and $UC_a$. The set of conflicts in a defined policy are detected as described in Algorithm 1.

*Example 1.* Let $\mathcal{L} = \{RND, DET, OPE\}$ be the set of security level that can be provided from the set of encryption schemes $\mathcal{E} = \{E_1, E_2, E_3\}$. Suppose that the $E_1$, $E_2$ and $E_3$ provide respectively $RND$, $DET$ and $OPE$, and satisfy respectively the functionalities $\emptyset$, $\{Equality,\ Join\}$ and $\{Min,\ Max\}$. Suppose that we want to enforce a policy composed of two constraints $TC_a = DET$ and $UC_a = \{Join,\ Min\}$. By performing the first step of Algorithm 1, we deduce that $F_{RND} = \emptyset$, $F_{DET} = \{Equality,\ Join\}$ and $F_{OPE} = \{Equality,\ Join,\ Min,\ Max\}$. The second step of Algorithm 1 gives that $UC_a \nsubseteq F_{DET}$, which allows to deduce that $TC_a$ and $UC_a$ are conflicting constraints.

```
input :
        $\mathcal{A}_s = \{a_1, \cdots, a_n\}$    /*sensitive attributes*/
        $\mathcal{C}_t = \{TC_{a_1}, \cdots, TC_{a_n}\}$ /*security threshold
constraints*/
        $\mathcal{C}_u = \{UC_{a_1}, \cdots, UC_{a_n}\}$ /*utility constraints*/
        $\mathcal{E} = \{E_1, \cdots, E_m\}$ /*encryption schemes*/
        $\mathcal{L} = \{l_1, \cdots, l_p\}$ /*security levels*/
output:
        $\mathcal{I}$    /*set of conflicts*/
Main
$\mathcal{I} = \emptyset$
/* First step */
foreach $l_i$ in $\mathcal{L}$ do
    $F_{l_i} = \emptyset$
    foreach $E_j$ in $\mathcal{E}$ do
        if ($l_j$ is more secure or equal $l_i$) then
            $F_{l_i} = F_{l_i} \cup F_j$
        end
    endfch
endfch
/* Second step */
foreach $a_k$ in $\mathcal{A}_s$ do
    if ( $UC_{a_k} \not\subseteq F_{TC_{a_k}}$ ) then
        $\mathcal{I} = \mathcal{I} \cup \{(a_k, UC_{a_k}, TC_{a_k})\}$
    end
endfch
```
**Algorithm 1:** Conflict detection

## 3.4 Policy satisfaction

The policy to be enforced over the outsourced database is composed of security and utility constraints. Those constraints can be satisfied through the application of encryption schemes. Our main challenge is to find for each sensitive attribute $a$ in the outsourced database, the *best combination of encryption schemes* that can satisfy the set of security and utility constraints defined over $a$.

*Definition 4.* (combination of encryption schemes) Let $\mathcal{E}$ be the set of available encryption schemes in the system, a combination of encryption schemes is a subset $C \subseteq \mathcal{E}$.

*Definition 5.* Let $C = \{E_1, \cdots, E_m\}$ be a combination of encryption schemes applied over the attribute $a$ and $l_i$ be the security level provided by the encryption scheme $E_i$, $1 \leq i \leq m$. The security level of the attribute $a$ provided by the application of $C$ is $l$, *iff* the following conditions hold:

- $l \in \{l_1, \cdots, l_m\}$
- $\forall l_j \in \{l_1, \cdots, l_m\}$, $l_j$ is at least as secure as $l$.

Note that the previous definition requires the security level provided by the combination of schemes in $C$ to be the lowest security level provided by the application of each encryption schemes in $C$. A strategy to find the combination of encryption schemes that satisfy the chosen policy consists of finding the *best combination of encryption schemes*, that is, it provides the highest level of protection for sensitive data, while minimizing the number of involved encryption schemes. We formalize this problem as follows:

*Problem 1.* (best combination of encryption schemes) Let $P$ be a policy, $\mathcal{C} = \{C_1, \cdots, C_n\}$ be a set of combinations of encryption schemes that satisfy the policy $P$, and $l_i$ be the security level provided by the application of the combination $C_i$, with $1 \leq i \leq n$. $C_k$ is the best combination of encryption schemes in $\mathcal{C}$ that satisfy $P$ *iff* the following conditions are satisfied:

- $\forall C_j \in \mathcal{C}$, $l_k$ is at least as secure as $l_j$.
- $\forall C_j \in \mathcal{C}$, $|C_k| \leq |C_j|$.

The problem of finding the best combination of encryption schemes is *NP-hard*. This is formally stated by the following theorem.

*Theorem 1.* The problem of finding the best combination of encryption schemes is *NP-hard*.

PROOF. We prove the previous theorem by a reduction from the NP-hard problem of minimum hypergraph coloring [13], which is formulated as follows: *given a hypergraph $G(V, E)$, determine a minimum coloring of $G$, that is, assign to each vertex in $V$ a color such that adjacent vertices have different colors, and the number of colors is minimized.*

We define the correspondence between finding the best combination of encryption schemes problem and the minimum hypergraph coloring problem as follows. Let $a$ be a sensitive attribute, $TC_a = l$ be a security threshold constraint defined over $a$, $UC_a = \{f_{a_1}, \cdots, f_{a_n}\}$ be a utility constraint defined over $a$, and $\mathcal{E}_l = \{E_1, \cdots, E_m\}$ the set of encryption schemes that provide a security level which is at least as secure as $l$. Any vertex $v_i \in V$ corresponds to a functionality $f_i \in \mathcal{F}$. We denote $e_a$ the edge in $G$ which connects $v_{a_1}, \cdots, v_{a_n}$, corresponds to the constraint $UC_a$. The combination of encryption schemes $C = \{E_{i_1}, \cdots, E_{i_p}\}$, where $C \subseteq \mathcal{E}$ and each $E_{i_j} \in C$ satisfies the set of functionalities $F_j = \{f_{j,1}, \cdots, f_{j,k_j}\}$, satisfies the constraint $UC_a$ correspond to a solution $S$ for the corresponding hypergraph coloring problem. More precisely, $S$ uses $p$ colors. Vertices $\{v_{1,1}, \cdots, v_{1,k_1}\}$ corresponding to the functionality satisfied by $E_{i_1}$ are colored using the first color, vertices $\{v_{q,1}, \cdots, v_{q,k_q}\}$ corresponding to the functionality satisfied by $E_{i_q}$ are colored using the $q$-th color, and vertices $\{v_{p,1}, \cdots, v_{p,k_p}\}$ corresponding to the functionality satisfied by $E_{i_p}$ are colored using the $p$-th color. Therefore, any algorithm finding the combination of encryption schemes that involved the minimal number of encryption mechanism while satisfying the constraint $UC_a$ can be used to solve the minimum hypergraph coloring problem.

Since the problem of finding the best combination of encryption schemes that satisfy a policy $P$ is NP-hard, we cannot expect to be able to solve an instance of arbitrary size of this problem to optimality. Thus, heuristic resolution strategies are widely exploited to solve such a problem with a reasonable computational effort.

## 3.5 Heuristic search

We propose a near-optimal heuristic for finding a combination of encryption schemes that satisfy a policy $P$. Our heuristic is based on a constructive method consisting of building a solution to the problem step by step from scratch. The used constructive method is based on choosing for each iteration, the *best satisfier* of the chosen policy.

*Definition 6.* (best satisfier) Let $\mathcal{P}$ be a policy composed of two constraints: a security threshold constraint $TC_a = l$

and an utility constraint $UC_a = \{f_{a_1}, \cdots, f_{a_n}\}$. Both constraints are defined over the sensitive attribute $a$. Let $\mathcal{E} = \{E_1, \cdots, E_m\}$ be the set of available encryption schemes. $E_i \in \mathcal{E}$ is a best satisfier if the following conditions are satisfied:

- The security level $l_{E_i}$ is at least as secure as $l$.
- $\forall E_j \in \mathcal{E}, l_{E_j}$ is at least as secure as $l$ and $|F_{E_i} \cap UC_a| \geq |F_{E_j} \cap UC_a|$, where $F_E$ are the set of functionalities satisfied by $E$.

The second condition in the previous definition states that $E_i$ is the *best satisfier* if it satisfies the highest number of functionalities in $UC_a$ compared to other encryption schemes in $\mathcal{E}$ that satisfy $TC_a$.

Algorithm 2 shows our heuristic algorithm for computing for each sensitive attribute, a combination of encryption schemes that satisfy the constraints defined over it. The algorithm takes as input the set of attributes $\mathcal{A}$ in the database to be outsourced, the policy $\mathcal{P}$ to be enforced over the set of attributes $\mathcal{A}$, the set of available encryption schemes $\mathcal{E}$ that can be used to enforce the policy $\mathcal{P}$, the set of security levels $\mathcal{L}$, and returns as output the set of combinations of mechanisms $\mathcal{S}$ that efficiently enforce the policy $\mathcal{P}$. For conflicting constraints, the algorithm returns a set of propositions $\mathcal{CP}$ to aid in resolving the conflicts.

The algorithm first initializes $\mathcal{S}$, $\mathcal{CP}$, $\mathcal{A}_s$ to the empty set and execute the procedure *get_conflicting_constraints* which takes as parameters $\mathcal{P}$, $\mathcal{E}$, $\mathcal{L}$, and return the set of conflicts in the policy. The *get_conflicting_constraints* procedure is represented by the Algorithm 1. Based on the confidentiality constraints in $\mathcal{P}$, the algorithm performs the first **foreach** loop to get all sensitive attributes $\mathcal{A}_s$. Then, for each sensitive attribute $a_i$ having an unconflicting constraint it tries to get the best combination of schemes in terms of the provided security level. In order to meet the previous goal, we use the **while** loop to run down the set of security levels in $\mathcal{L}$ which are at least as secure as ($\geq_s$) $TC_i$ starting from the highest one. For each security level *sec_lev*, we get from $\mathcal{E}$ the set $\mathcal{E}_{sec\_lev}$ of encryption schemes that provide security levels which are at least as secure as *sec_lev* and which can satisfy functionalities in $UC_i$. Next, we copy the set of required functionalities $UC_i$ to $UC_{temp}$, and at each iteration of the next **while** loop, we get the *best satisfier* $E_{bs}$ from $\mathcal{E}_{sec\_lev}$ according to the Definition 6. $E_{bs}$ will be next added to the combination $Sol$, removed from $\mathcal{E}_{sec\_lev}$, and the required functionalities satisfied by $E_{bs}$ will be removed from $UC_{temp}$. This **while** loop is terminated if: (1) all required functionalities in $UC_{temp}$ are satisfied, in this case the set $Sol$ represents the combination allowing to satisfy the constraints defined over the attribute $a_i$; or (2) $\mathcal{E}_{sec\_lev}$ is empty, which means that there is no combination that satisfies $UC_i$ in the security level *sec_lev*.

For each attribute $a_i$ having a conflicting constraint, using the third outermost **foreach** loop, the algorithm gives additional proposition allowing to avoid the conflict. To meet this goal, we use the first **while** loop in the third outermost **foreach** loop to run down the set of security levels in $\mathcal{L}$ starting from $TC_i$. We perform the same operation as in the previous outermost **foreach** loop, except, for each *sec_lev*, we will add to the set of propositions $\mathcal{CP}$ the entry $(a_i, Prop, sat\_func, sec\_lev)$ stating that in the security level *sec_lev*, the combination of schemes $Prop$ is able to

```
input  : $\mathcal{A} = \{a_1, \cdots, a_n\}$   /*database attributes*/
         $\mathcal{P} = \{CC_1, \cdots, CC_l, TC_1, \cdots, TC_l, UC_1, \cdots, UC_l\}$
         $\mathcal{E} = \{E_1, \cdots, E_m\}$ /*encryption schemes*/
         $\mathcal{L} = \{l_1, \cdots, l_p\}$ /*security levels*/
output: $\mathcal{S}$   /*Solution*/
         $\mathcal{CP}$   /*Conflict resolution propositions*/
Main
$\mathcal{S} = \emptyset$
$\mathcal{CP} = \emptyset$
$\mathcal{A}_s = \emptyset$
$Conflicts = get\_conflicting\_constraints(\mathcal{P}, \mathcal{E}, \mathcal{L})$
foreach $CC$ in $\mathcal{P}$ do
    $\mathcal{A}_s = \mathcal{A}_s \cup CC$
endfch
foreach $a_i$ in $\mathcal{A}_s$ do
    if (not $(a_i, TC_i, UC_i)$ in $Conflicts$) then
        $sec\_lev = get\_the\_highest\_sec\_lev(\mathcal{L})$
        $Sol = \emptyset$
        while $sec\_lev \geq_s TC_i$ do
            $Sol = \emptyset$
            $\mathcal{E}_{sec\_lev} = \emptyset$
            foreach $E$ in $\mathcal{E}$ do
                if $(l_E \geq_s sec\_lev$ and $F_E \cap UC_i \neq \emptyset)$ then
                    $\mathcal{E}_{sec\_lev} = \mathcal{E}_{sec\_lev} \cup E$
                end
            endfch
            $UC_{temp} = UC_i$
            while $(UC_{temp} \neq \emptyset$ and $\mathcal{E}_{sec\_lev} \neq \emptyset)$ do
                $E_{bs} = get\_first\_elem(\mathcal{E}_{sec\_lev})$
                foreach $E$ in $\mathcal{E}_{sec\_lev}$ do
                    if $(|F_E \cap UC_i| \geq |F_{E_{bs}} \cap UC_i|)$ then
                        $E_{bs} = E$
                    end
                endfch
                $Sol = Sol \cup E_{bs}$
                $\mathcal{E}_{sec\_lev} = \mathcal{E}_{sec\_lev} \setminus \{E_{bs}\}$
                $UC_{temp} = UC_{temp} \setminus (F_{E_{bs}} \cap UC_{temp})$
            end
            if $(UC_{temp} = \emptyset)$ then
                break
            end
            if $(\mathcal{E}_{sec\_lev} = \emptyset)$ then
                $sec\_lev = get\_next\_best\_level(sec\_lev, \mathcal{L})$
            end
        end
        $\mathcal{S} = \mathcal{S} \cup \{(a_i, Sol, sec\_lev)\}$
    end
endfch
foreach $(a_i, TC_i, UC_i)$ in $Conflicts$ do
    $Prop = \emptyset$
    $sec\_lev = TC_i$
    while $sec\_lev \neq NULL$ do
        $Prop = \emptyset$
        $\mathcal{E}_{sec\_lev} = \emptyset$
        foreach $E$ in $\mathcal{E}$ do
            if $(l_E \geq_s sec\_lev$ and $F_E \cap UC_i \neq \emptyset)$ then
                $\mathcal{E}_{sec\_lev} = \mathcal{E}_{sec\_lev} \cup E$
            end
        endfch
        $UC_{temp} = UC_i$
        while $(UC_{temp} \neq \emptyset$ and $\mathcal{E}_{sec\_lev} \neq \emptyset)$ do
            $E_{bs} = get\_first\_elem(\mathcal{E}_{sec\_lev})$
            foreach $E$ in $\mathcal{E}_{sec\_lev}$ do
                if $(|F_E \cap UC_i| \geq |F_{E_{bs}} \cap UC_i|)$ then
                    $E_{bs} = E$
                end
            endfch
            $Prop = Prop \cup E_{bs}$
            $\mathcal{E}_{sec\_lev} = \mathcal{E}_{sec\_lev} \setminus \{E_{bs}\}$
            $UC_{temp} = UC_{temp} \setminus (F_{E_{bs}} \cap UC_{temp})$
        end
        $sat\_func = UC_i \setminus UC_{temp}$
        $\mathcal{CP} = \mathcal{CP} \cup \{(a_i, Prop, sat\_func, sec\_lev)\}$
        if $(UC_{temp} = \emptyset)$ then
            break
        end
        if $(\mathcal{E}_{sec\_lev} = \emptyset)$ then
            $sec\_lev = get\_next\_best\_level(sec\_lev, \mathcal{L})$
        end
    end
endfch
```

**Algorithm 2:** Policy satisfaction

satisfy the set of functionalities $sat\_func$ required for the attribute $a_i$. These propositions may help the security administrator (data owner) to choose, from his point of view, the best trade off between security and utility.

*Theorem 2.* (Complexity) Given a set of $p$ attributes $\mathcal{A}$, a policy $\mathcal{P}$ composed of $n$ confidentiality constraints, $n$ security threshold constraints, $n$ utility constraints, a set of $m$ encryption schemes $\mathcal{E}$, and a set of $r$ security levels, the complexity of the policy satisfaction algorithm (Algorithm 2) is $O(m^2 \cdot n \cdot r + r \cdot m + 2n)$.

PROOF. (sketch) We suppose that we have $p$ attributes having unconflicting constraints and $q$ attributes having conflicting constraints, with $p + q = n$. According to Algorithm 1, the execution of the function $get\_conflicting\_constraints$ costs $O(r \cdot m + n)$. In Algorithm 2, the first *foreach* loop costs $O(n)$, the second *foreach* loop costs in the worst case $O(p \cdot r \cdot m^2)$, and the third *foreach* loop costs in the worst case $O(q \cdot r \cdot m^2)$. Finally, the overall time complexity of the Algorithm 2 is $O(m^2 \cdot n \cdot r + r \cdot m + 2n)$.

## 4. USE CASE

In this section, we present the use case. For our case study, we use a scenario based on the TPC-H [1] benchmark database. We first give an overview of the TPC-H benchmark database structure. Afterwards, we present the set of encryption schemes that can be used in our scenario, a set of functionalities required for processing the data, and policies to be applied over the TPC-H database. Finally, we illustrate the use of our previously presented policy satisfaction algorithm to enforce the chosen policy over the TPC-H database.

### 4.1 TPC-H database

The TPC-H database is composed of 8 tables. Each attribute in TPC-H tables represents data for industrial resource management. TPC-H provides 22 queries consisting of different kind of SQL operations such as select, join, order by, etc. Figure 1 represents the conceptual model of the TPC-H database which includes foreign key relationships.

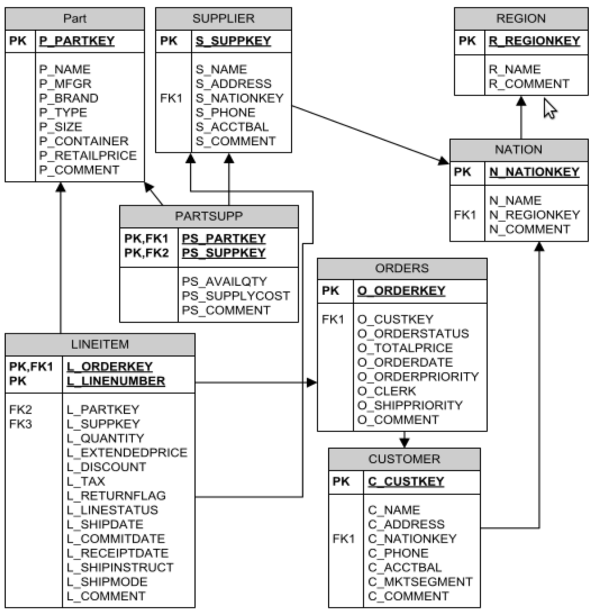

**Figure 1: TPCH database**

### 4.2 System design

As described in 3.1, the used system is composed of a relational database $\mathcal{D}$, a set of security layers $\mathcal{L}$, a set of functional requirements $\mathcal{F}$, and a toolbox $\mathcal{E}$. In our case study, $\mathcal{D}$ represents the TPC-H benchmark database, $\mathcal{L}$ will be composed of three security layers as explained in 2.3: $RND$ (random layer), $DET$ (deterministic layer) and $OPE$ (order preserving layer). As we work with relational databases, the set of utility requirements are composed of some SQL operators that can be used to query the database. In addition, we define the functionalities *computation* representing the numeric computation over the attributes (e.g., SET ATTR = ATTR + 30), and *order search* represeting the SQL operators ($>, \geq, <, \leq, between, min/max, order\ by$). Thus $\mathcal{F} = \{equality, join, group\ by, average, sum, computation, like,$ $order\ search\}$. The toolbox $\mathcal{E}$ is composed of the following encryption schemes. For each encryption scheme, we extract and specify the provided security level and the set of satisfied functionalities as presented in 3.1.

**AES-CBC.** When used in CBC chaining mode, AES provides a probabilistic encryption which is semantically secure. Thus, it provides the security level *RND*. Despite that this encryption scheme does not leak any information about the plaintext values, it does not allow any efficient computation over encrypted data. Therefore, $l_{AES} = RND$ and $F_{AES} = \emptyset$.

**Paillier [22].** It is based on secure probabilistic encryption which enables to perform computation aver encrypted data. A Paillier cryptosystem provides indistinguishability under an adaptive chosen-plaintext attack (IND-CPA). It provides the security level *RND* and allows to perform *sum*, *avg* operations over the encrypted data. Thus, $l_{Plr} = RND$ and $F_{plr} = \{sum, avg, computation\}$.

**SSE [28].** SSE is a symmetric searchable encryption which is semantically secure (as long as there is no search token). It allows to perform search over encrypted data which gives the ability to perform MySQL's *like* operator. Based on these properties, the SSE can be specified by $l_{SSE} = RND$ and $F_{SSE} = \{like\}$.

**Pohlig-Hellman.** This is a deterministic encryption scheme allowing logarithmic time equality checks over ciphertexts. Pohlig-Hellman encryption cannot achieve the classical notions of security of probabilistic encryption because it leaks which encrypted values correspond to the same plaintext value. It provides the security level $DET$ and allows to perform *equality*, *join*, and *group by* over the encrypted data. Thus, $l_{PH} = DET$ and $F_{PH} = \{equality, join, group\ by\}$.

**Boldyreva [5, 6].** Boldyreva propose an order-preserving, deterministic encryption which allows performing order operations over encrypted data. As mentioned in 2.3, in addition to the information leaked by having the deterministic property, it reveals the order between encrypted values. The encryption scheme provides the security level $OPE$ and allows to perform *equality*, *join*, *group by*, and *order search* operations. Thus, $l_{Bdv} = OPE$ and $F_{Bdv} = \{equality, join, group\ by, order\ search\}$.

## 4.3 The policy

In our scenario a security administrator (data owner) of the TPC-H benchmark database requires that the following security rules must be enforced:

1. The given discount for any Order should always remain *top secret.*
2. The account balance for a customer as well as our suppliers should always remain *top secret.*
3. The Name and Address of our suppliers should be *confidential.*
4. The supply cost of individual suppliers must be *confidential.*
5. Any pricing information must in general remain *secret.*
6. All other information in the database should be *unclassified.*

The security administrator used four levels to classify the data. The *top secret* classification levels means that any leaked information about the data will cause grave damage. The *secret* level means that some information about the data values can be leaked if they do not lead to reveal its values. The *confidential* level means that additional information about the data values can be leaked if they do not lead to reveal the values itselves. A *Unclassified* level implies that the data are not sensitive.

According to the properties of the security levels in $\mathcal{L}$ described in 2.3, we associate the *top secret* classification levels to the *RND* security level, the *secret* classification level to the *DET* security level, and the *confidential* classification level to the *OPE* security level. The previous rules are specified as follows:

**Rule 1.** It involves the attribute *L_DISCOUNT* of the table *LINEITEM*. This rule is specified using the folowing confidentiality and security threshold constraints:

- $CC_1 = \{L\_DISCOUNT\}$, $TC_1 = RND$.

**Rule 2.** This rule involves the attributes *C_ACCTBAL* and *S_ACCTBAL* from the tables *CUSTOMER* and *SUPPLIER*. It is specified using the following constraints:

- $CC_2 = \{C\_ACCTBAL\}$, $TC_2 = RND$.
- $CC_3 = \{S\_ACCTBAL\}$, $TC_3 = RND$.

**Rule 3.** It involves the attributes *S_NAME*, *S_ADDRESS*, and *S_NATIONKEY* from the table *SUPPLIER*. It is specified using the following constraints:

- $CC_4 = \{S\_NAME\}$, $TC_4 = OPE$.
- $CC_5 = \{S\_ADDRESS\}$, $TC_5 = OPE$.
- $CC_6 = \{S\_NATIONKEY\}$, $TC_6 = OPE$.

**Rule 4.** It involves the attribute *PS_SUPPLYCOST* from the table *SUPPLYCOST*. This rule is specified using the following constraints:

| Sensitive attributes | Functionalities |
|---|---|
| *L_DISCOUNT* | *computation*(Q1,Q3,Q4) *sum*(Q1,Q3,Q4) *order search*(Q3) |
| *C_ACCTBAL* | *group by*(Q5) *sum*(Q8) |
| *S_ACCTBAL* | *order search*(Q2) |
| *S_NAME* | *order search*(Q2,Q7) *group by*(Q7) |
| *S_ADDRESS* | *like*(Q4) |
| *S_NATIONKEY* | *join*(Q2,Q4) |
| *PS_SUPPLYCOST* | *equality*(Q2) |
| *P_RETAILPRICE* | |
| *L_EXTENDEDPRICE* | *sum*(Q1,Q3) *computation*(Q3,Q4) |
| *O_TOTALPRICE* | *group by*(Q6) *order search*(Q6) |

**Table 1: Required functionalities for sensitive attributes**

- $CC_7 = \{PS\_SUPPLYCOST\}$, $TC_7 = OPE$.

**Rule 5.** This rule involves the attributes *P_RETAILPRICE*, *L_EXTENDEDPRICE* and *O_TOTALPRICE* from tables *PART*, *LINEITEM* and *ORDERS*. It is specified using the following constraints:

- $CC_8 = \{P\_RETAILPRICE\}$, $TC_8 = DET$
- $CC_9 = \{L\_EXTENDEDPRICE\}$, $TC_9 = DET$
- $CC_{10} = \{O\_TOTALPRICE\}$, $TC_{10} = DET$

The security administrator gives examples of queries which should be executes efficiently over the TPC-H database. From these set of queries, we extract only the queries involving sensitive attributes described in the policy, which are illustrated in Figure 2. These queries enable us to extract the set of functionalities required for each sensitive attribute in the TPC-H database. Table 1 shows, for each sensitive attribute, the queries on which the attribute is involved and the set of required functionalities. These functional requirements are specified using the following utility constraints:

- $UC_1 = \{computation, sum, order\ search\}$
- $UC_2 = \{group\ by, sum\}$
- $UC_3 = \{order\ search\}$
- $UC_4 = \{order\ search, group\ by\}$
- $UC_5 = \{like\}$
- $UC_6 = \{join\}$
- $UC_7 = \{equality\}$
- $UC_8 = \emptyset$
- $UC_9 = \{sum, computation\}$
- $UC_{10} = \{group\ by, order\ search\}$

## 4.4 Policy enforcement results

Using the Algorithm 2, we get from the toolbox, for each sensitive attribute, the encryption scheme or the combination of encryption schemes that satisfies the policy. The results of the application of Algorithm 2 over our use case are the followings:

```
Q1:
SELECT L_RETURNFLAG, L_LINESTATUS,
  SUM(L_QUANTITY) AS SUM_QTY,
  SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE,
  SUM(1-L_DISCOUNT) AS SUM_DISC_PRICE,
  AVG(L_QUANTITY) AS AVG_QTY,
FROM LINEITEM
WHERE
  L_SHIPDATE <= '2010-01-15'
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS

Q2:
SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY,
  P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM PART, SUPPLIER, PARTSUPP, NATION, REGION
WHERE
  P_PARTKEY = PS_PARTKEY AND
  S_NATIONKEY = N_NATIONKEY
  PS_SUPPLYCOST = 1000
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME

Q3:
SELECT SUM(L_DISCOUNT) AS REVENUE
FROM LINEITEM
WHERE L_SHIPDATE >= '2010-01-01' AND
  L_SHIPDATE < '2010-01-01'
  AND L_DISCOUNT BETWEEN .06 - 0.01 AND .06 + 0.01
  AND L_QUANTITY < 24

Q4:
SELECT N_NAME AS NATION,
  L_EXTENDEDPRICE*(1-L_DISCOUNT) AS AMOUNT
FROM PART, SUPPLIER, LINEITEM, NATION
WHERE S_SUPPKEY = L_SUPPKEY
  AND S_NATIONKEY = N_NATIONKEY
  AND S_ADDRESS LIKE '%%RENNES%%'
Group By N_NAME.

Q5:
SELECT TOP 20 C_NAME, C_ACCTBAL,
  N_NAME, C_ADDRESS, C_PHONE, C_COMMENT
FROM CUSTOMER, ORDERS, LINEITEM, NATION
WHERE C_CUSTKEY = O_CUSTKEY AND
  L_ORDERKEY = O_ORDERKEY AND
  L_RETURNFLAG = 'R'
GROUP BY C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE
ORDER BY C_NAME.

Q6:
SELECT C_NAME, O_ORDERDATE,
  O_TOTALPRICE, SUM(L_QUANTITY)
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY AND
  O_ORDERKEY = L_ORDERKEY
GROUP BY C_NAME, C_CUSTKEY, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC.

Q7:
SELECT TOP 100 S_NAME, COUNT(*) AS NUMWAIT
FROM SUPPLIER, LINEITEM L1, ORDERS, NATION
WHERE S_SUPPKEY = L1.L_SUPPKEY AND
  O_ORDERKEY = L1.L_ORDERKEY AND
  L1.L_RECEIPTDATE> L1.L_COMMITDATE
GROUP BY S_NAME
ORDER BY NUMWAIT DESC, S_NAME.

Q8:
SELECT CNTRYCODE, COUNT(*) AS NUMCUST,
  SUM(C_ACCTBAL) AS TOTACCTBAL
FROM
    (SELECT SUBSTRING(C_PHONE,1,2) AS
    CNTRYCODE, C_ACCTBAL
    FROM CUSTOMER
    WHERE
    SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29'))
GROUP BY CNTRYCODE
```

**Figure 2: Queries involving sensitive attributes**

1. C_ACCTBAL: conflict detected ($TC_2$ and $UC_2$)
   Conflicts resolution propositions:
   - $[Paillier]$ (RND), satisfied utility requirements: $\{sum\}$ (Q8)
   - $[Paillier, Pohlig{-}Hellman]$ (DET), satisfied utility requirements: $\{group\ by, sum\}$ (Q8, Q5)
2. L_EXTENDEDPRICE: $[Paillier]$ (RND), satisfied utility requirements: $\{sum, computation\}$ (Q1,Q3,Q4).
3. PS_SUPPLYCOST: $[Pohlig - Hellman]$ (DET), satisfied utility requirements: $\{equality\}$ (Q2).
4. L_DISCOUNT: conflict detected ($TC_1$ and $UC_1$)
   Conflicts resolution propositions:
   - $[Paillier]$ (RND), satisfied utility requirements: $\{sum, computation\}$ (Q1,Q4).
   - $[Paillier, Boldyreva]$ (OPE), satisfied utility requirements: $\{sum, order\ search, computation\}$ (Q1,Q3,Q4).
5. S_ADDRESS: $[SSE]$ (RND), satisfied utility requirements: $\{like\}$ (Q4).
6. S_NAME: $[Boldyreva]$ (OPE), satisfied utility requirements: $\{order\ search, group\ by\}$ (Q2,Q7).
7. S_NATIONKEY: $[Pohlig{-}Hellman]$ (DET), satisfied utility requirements: $\{join\}$ (Q2,Q4).
8. S_ACCTBAL: conflict detected ($TC_3$ and $UC_3$)
   Conflicts resolution propositions:
   - $[AES - CBC]$ (RND), satisfied utility requirements: $\emptyset$.
   - $[Boldyreva]$ (OPE), satisfied utility requirements: $\{order\ search\}$ (Q2).
9. P_RETAILPRICE: $[AES - CBC]$ (RND).
10. O_TOTALPRICE: conflict detected ($TC_{10}$ and $UC_{10}$)
    Conflicts resolution propositions:
    - $[Pohlig - Hellman]$ (DET), satisfied utility requirements: $\{group\ by\}$.
    - $[Boldyreva]$ (OPE), satisfied utility requirements: $\{group\ by, order\ search\}$ (Q6).

Result 1 shows the satisfaction of the constraints defined over the attribute C_ACCTBAL. A conflict between the constraints $TC_2$ and $UC_2$ has been detected. Thus, our algorithm gives the data owner two propositions in order to resolve the conflict. The first proposition states that the data owner can preserve the $RND$ security level through the application of the *Paillier* encryption scheme, however only the *sum* functionality will be provided and therefore the query Q5 cannot be executed efficiently over the encrypted data. The second proposition gives the data owner the ability to decrease the required threshold security level to $DET$ in order to allows the application of the combination $[Paillier, Pohlig - Hellman]$ which satisfies the required utility constraints. Result 2 states that the encryption scheme *Paillier* can be applied to enforce the set of security and utility requirements defined over the attribute L_EXTENDEDPRICE. Result 3, shows that security and utility constraints defined over the attribute PS_SUPPLYCOST can be enforced through the application of $Pohlig-Hellman$ encryption scheme. Result 4 shows that there is a conflict between the constraints $TC_1$ and $UC_1$ and proposes two solution to reconcile the conflict. Result 5 states that the encryption scheme $SSE$ can be applied to enforce the set of security and utility requirements defined over the attribute S_ADDRESS. Result 6 shows that the set of security and utility constraints defined over the attribute S_NAME can be enforced via the application of $Boldyreva$ encryption scheme. Result 7 states that the encryption scheme $Pohlig - Hellman$, when applied, can enforce the of security and utility requirements defined over the attribute S_NATIONKEY. For this result, we remark that our algorithm has chosen the best encryption scheme in terms of provided security level, as the $Boldyreva$ encryption scheme can also be used to enforce security and utility requirements defined over the attribute S_NATIONKEY. Result 8 shows the conflict detected between $TC_3$ and $UC3$ and proposes two solutions two overcome the conflict. Result 9 confirms that the application of $AES - CBC$ can enforce the constraints defined over the attribute P_RETAILPRICE. Finally, result 10 shows that there is a conflict between $TC_10$ and $UC_10$ and proposes two solution allowing to reconcile the conflict.

It is important to note that sequentially applying a combination of encryption schemes (e.g. [Paillier and Pohlig-Hellman] in one "onion") over an attribute may not provide the functionalities provided by each encryption scheme. This problem can be resolved by duplicating the values of the attribute over which the two mechanisms are to be applied and apply each mechanism separately.

## 5. RELATED WORK

### 5.1 Encrypted Databases

Hacigümüs et al. first introduced the concept of executing queries over encrypted data [16]. They used deterministic encryption for equality queries and binning for range queries. The binning concept was superseded by order-preserving encryption by Agrawal et al. [2]. Order-preserving encryption allows the processing of range queries also on encrypted data. Damiani et al. have presented an algorithm for optimizing the trade-off between security and performance of post-processing in encrypted databases [10]. Their algorithm combines deterministic ciphertexts if the security benefit outweighs the performance penalty. Popa et al. have recently presented CryptDB which introduces adjustable encryption as we use it [24]. This removes the need for analyzing all queries in advance, but also poses the policy configuration we solve in this paper.

The cryptographic community has further developed the encryption schemes used for processing queries. Bellare et al. introduce a concept for deterministic, public-key encryption [4]. They also note that only deterministic encryption can achieve sub-linear search. Boldyreva et al. propose a new order-preserving encryption scheme [5, 6]. Their scheme is as secure as any immutable, order-preserving encryption scheme can be. Popa et al. introduced an ideal-secure order-preserving encryption scheme [23]. Nevertheless, this scheme and adjustable "onion" encryption are not efficiently combinable. Recently, Gentry [14] presented fully homomorphic encryption. Implementing search in fully homomorphic encryption is difficult, since the result of the match is still encrypted. Therefore in a perfectly semantically secure search the database has to return all records completely annihilating the advantage of searching on the database. Instead, searchable encryption offers an almost as secure alternative. Song et al. introduced searchable encryption [28]. For equality searches indices achieve sub-linear time in the average case [7]. Still, they require an index for each searchable attribute which is currently still too inefficient in practice. The scheme of Shi et al. [27] performs range searches using logarithmically sized ciphertexts. Semantically secure range searches still always require a linear scan, i.e. linear time.

### 5.2 Policy Configuration

There are a couple of approaches to enforce and manage access control policies in encrypted databases. De Capitani di Vimercati et al. proposed over-encryption [12]: the layering of encryption to enforce policies by the user and the database. Again, De Capitani et al. presented an approach to selectively update the access control policies with minimal effort [11]. Damiani et al. have proposed selective encryption in order to also outsource the access control [9]. They presented the key management for this approach in [8]. Atallah et al. presented an efficient key management scheme for hierarchies [3]. Ion et al. use proxy re-encryptable, searchable encryption in order to enforce multi-user access policies [17]. An approach recently implemented for web applications is using a different scheme by Popa et al. [25].

All of these approaches implement access control for different users via encryption. Our approach and use of policies is different, albeit we also enforce confidentiality constraints. We configure the use of encryption in order to prevent unintended disclosure against the database provider (not other users of the database).

## 6. CONCLUSIONS AND FUTURE WORK

Searchable, yet encrypted databases appear to be one promising building block of a secure cloud offering. In order to help companies migrate data from on-premise to the cloud, tools are needed to help decide about the best acceptable trade-off between functionality and security requirements. In this paper we presented a set of algorithms which help to analyze functionality and security requirements when configuring an encrypted database following an onion-based approach. We reasoned about their formal characteristics as well as discussed their application in an enterprise use case on basis of the TPC-H benchmark. The assumption that data may be labelled as we proposed may appear oversimplified, but industrial experience shows that even in complex applications this is sufficient to cover the evaluation results of a typical 3x3 risk matrix. This work complements some earlier work of ours where we could show that in the context of some representative industrial customer datasets in the best case only 8 percent of all data ever had to be encrypted using an order-preserving scheme while still supporting all the required queries [15]. Future work will now concentrate on further tool support for building the optimal encrypted structures when outsourcing encrypeted searchable data. Even with our proposed optimizations, the initial, onion-based encryption of the plaintext data is computationally expensive and we are thus investigating on how to parallelize this. Thirdly, we are implementing support for splitting queries in local and remote parts [21].

## 7. ACKNOWLEDGMENTS

- This work has received a French government support granted to the CominLabs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.
- This work has received a French government support granted to the Frag&Tag project and managed by the Dual Innovation Support Scheme (RAPID) under convention No 132906023.
- This work has received a support from EIT ICT Labs Doctoral School and the EIT ICT Labs Doctoral Training Center in Rennes.

## 8. REFERENCES

[1] Transaction processing performance council. benchmark h. http://www.tpc.org/.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD, 2004.

[3] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.

[4] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology*, CRYPTO, 2007.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *Proceedings of the 28th International Conference on Advances in Cryptology*, EUROCRYPT, 2009.

[6] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *Proceedings of the 31st International Conference on Advances in Cryptology*, CRYPTO, 2011.

[7] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5), 2011.

[8] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Key management for multi-user encrypted databases. In *Proceedings of the ACM Workshop on Storage Security and Survivability*, StorageSS, 2005.

[9] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Selective data encryption in outsourced dynamic environments. In *Proceedings of the Second International Workshop on Views on Designing Complex Architectures*, VODCA, 2007.

[10] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS, 2003.

[11] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. A data outsourcing architecture combining cryptography and access control. In *Proceedings of the ACM Workshop on Computer Security Architecture*, CSAW, 2007.

[12] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: Management of access control evolution on outsourced data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB, 2007.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[14] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Symposium on Theory of Computing*, STOC, 2009.

[15] P. Grofig, M. Härterich, I. Hang, F. Kerschbaum, M. Kohler, A. Schaad, A. Schröpfer, and W. Tighzert. Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data. In *Proceedings of the Conference of the GI Security Group*, SICHERHEIT, 2014.

[16] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM International Conference on Management of Data*, SIGMOD, 2002.

[17] M. Ion, G. Russello, and B. Crispo. Enforcing multi-user access policies to encrypted cloud databases. In *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY, 2011.

[18] M. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Proceedings of the 19th Network and Distributed System Security Symposium*, NDSS, 2012.

[19] F. Kerschbaum, P. Grofig, I. Hang, M. Härterich, M. Kohler, A. Schaad, A. Schröpfer, and W. Tighzert. Adjustably encrypted in-memory column-store. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, CCS, 2013.

[20] F. Kerschbaum, M. Härterich, P. Grofig, M. Kohler, A. Schaad, A. Schröpfer, and W. Tighzert. Optimal re-encryption strategy for joins in encrypted databases. In *Proceedings of the IFIP Conference on Data and Applications Security and Privacy*, DBSec, 2013.

[21] F. Kerschbaum, M. Härterich, M. Kohler, I. Hang, A. Schaad, A. Schröpfer, and W. Tighzert. An encrypted in-memory column-store: The onion selection problem. In *Proceedings of the 9th International Conference on Information Systems Security*, ICISS, 2013.

[22] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 18th International Conference on Advances in Cryptology*, EUROCRYPT, 1999.

[23] R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *34th IEEE Symposium on Security and Privacy*, S&P, 2013.

[24] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, SOSP, 2011.

[25] R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, M. F. Kaashoek, and H. Balakrishnan. Securing web applications by blindfolding the server. In *Proceedings of the USENIX Symposium of Networked Systems Design and Implementation*, NSDI, 2014.

[26] P. Samarati and S. De Capitani di Vimercati. Data protection in outsourcing scenarios: issues and directions. In *ASIACCS*, pages 1–14, 2010.

[27] E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *Proceedings of the 2007 Symposium on Security and Privacy*, S&P, 2007.

[28] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 21st IEEE Symposium on Security and Privacy*, S&P, 2000.

[29] L. Xiao, O. Bastani, and I.-L. Yen. Security analysis for order preserving encryption schemes. Technical Report UTDCS-01-12, Department of Computer Science, University of Texas Dallas, 2012.