Chip-Secured Data Access: Confidential Data on Untrusted Servers

Luc Bouganim, Philippe Pucheral

PRISM Laboratory – 78035 Versailles – France
<Firstname.Lastname>@prism.uvsq.fr

Abstract

The democratization of ubiquitous computing (access data anywhere, anytime, anyhow), the increasing connection of corporate databases to the Internet and the today's natural resort to Web-hosting companies strongly emphasize the need for data confidentiality. Database servers arouse user's suspicion because no one can fully trust traditional security mechanisms against more and more frequent and malicious attacks and no one can be fully confident on an invisible DBA administering confidential data. This paper gives an in-depth analysis of existing security solutions and concludes on the intrinsic weakness of the traditional server-based approach to preserve data confidentiality. With this statement in mind, we propose a solution called C-SDA (Chip-Secured Data Access), which enforces data confidentiality and controls personal privileges thanks to a client-based security component acting as a mediator between a client and an encrypted database. This component is embedded in a smartcard to prevent any tampering to occur. This cooperation of hardware and software security components constitutes a strong guarantee against attacks threatening personal as well as business data.

1. Introduction

The rapid growth of ubiquitous computing impels mobile users to store personal data on the Web to increase its availability. In the same way, corporate databases are made more and more accessible to authorized employees over the Internet. Small businesses are prompted to delegate part of their information system to Web-hosting companies or Database Service Providers (DSP) that guarantee data resiliency, consistency and high availability [eCr02,CaB02,Qck02]. Customer information is also maintained on-line for the needs of e-commerce and e-business applications. Typically, Microsoft .NET Passport [Mic02] gathers customer information (identity, passwords, credit card numbers, profiling data) in an electronic wallet shared by all participating .NET Web sites. Consequently, the amount of sensitive information collected and shared in the marketplace is such that data confidentiality has become one of the major concerns of citizens, companies and public organizations, and constitutes a tremendous challenge for the database community.

Confidential data threatened by attackers is manifold: information related to the private life of individuals (e.g., agenda, address book, bookmarks, medical records, household expenses), credit card numbers, patents, business strategies, diplomatic or military secrets. Even ordinary data may become sensitive once grouped and well organized in databases. Customers have no other choice than trusting DSP's arguing that their systems are fully secured and their employees are beyond any suspicion. However, according to the "Computer Crime and Security Survey" published by the Computer Security Institute (CSI) and the FBI [FBI01], the theft of intellectual property due to database vulnerability costs American businesses $103 billion annually and 45% of the attacks are conducted by insiders.

Traditional database security policies rely on user authentication, communication encryption and server-enforced access controls [BPS96]. Unfortunately, these mechanisms are inoperative against most insider attacks and particularly against database administrator attacks. Several attempts have been made recently to strengthen server-based database security policies thanks to database encryption [Ora99, Mat00, HeW01].

This paper first characterizes the intrinsic limits of these server-based solutions with respect to the different types of attacks that can be conducted. With these limitations in mind, we state the dimensions of the *data confidentiality problem*.

While client-based security policies have been historically disregarded considering the vulnerability of client environments [Rus01], we argue that the emergence of smartcard secured client devices fundamentally changes the problem statement. Initially developed by Bull to secure the French banking system, smartcards have been

**Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002**

used successfully around the world in various applications managing secured data (such as banking, pay-TV or GSM subscriber identification, loyalty, healthcare, insurance, etc.). Unfortunately, smartcards suffer from intrinsic hardware constraints that confine their applicability in terms of data management to secure portable folders (e.g., healthcare folder) [ISO99, PBV01].

We capitalize on the security properties of the smartcard to devise a solution to the data confidentiality problem, named *Chip-Secured Data Access (C-SDA)*. C-SDA takes the form of a security software embedded in a smartcard. This software acts as an incorruptible mediator between a client and a server hosting an encrypted database. The confidence in C-SDA relies on the fact that data encryption, query evaluation and access right management are insulated in a smartcard and cannot be tampered by anyone, including the cardholder. Dedicated query evaluation techniques are proposed to tackle the strong smartcard hardware constraints. We show the conclusive benefit of associating software and hardware security to preserve data confidentiality.

The contribution of this paper is twofold. First, it clearly states the dimensions of the data confidentiality problem and explains to which extent existing security solutions fail in addressing some of these dimensions. Second, it proposes a novel database security model where confidentiality is delegated to a tamper-resistant client device. This model is being validated in the context of a BtoB project supported by the French ANVAR agency (Agence Nationale pour la VAlorisation de la Recherche). This project will give us the opportunity to assess the functionality and performance of C-SDA on a real world application.

This paper is organized as follows. Section 2 characterizes the attacks that can be conducted against confidential data, analyzes the strengths and weaknesses of existing secure database solutions and concludes with a precise formulation of the *data confidentiality problem*. Section 3 introduces the Chip-Secured Data Access approach and shows how it answers each dimension of the data confidentiality problem. Section 4 addresses query management issues. Section 5 concentrates on data encryption and access right management. Section 6 develops a complete scenario illustrating the behavior of C-SDA on a concrete example. Finally, section 7 concludes the paper and sketches future research directions.

2. Data confidentiality problem

In this section, we first introduce the distinction between data privacy and data confidentiality. Then, we characterize the class of attacks that are commonly directed against databases. We discuss afterwards how server-based and client-based approaches resist to these attacks. We conclude by a precise formulation of the *data confidentiality problem* addressed in this paper.

2.1. Data privacy vs. data confidentiality

This paper concentrates on a particular aspect of database security, that is *data confidentiality*. Data confidentiality refers to the ability to share sensitive data among a community of users while respecting the privileges granted by the data owner to each member of the community. Any user external to the community is assumed to have no privilege at all. A special case of data confidentiality is *data privacy*. Data privacy means that the data owned by an individual will never be disclosed to anyone else.

Privacy is easier to enforce than confidentiality since sharing is precluded. The simplest and most effective way to ensure data privacy is to encrypt the user's data thanks to a symmetric key algorithm (e.g., DES [NIS93]). The user being the unique holder of the cipher key, no one else can access the clear text form of the data. Several Storage Service Providers propose to manage encrypted backups for personal data [Sky02]. They guarantee that data is encrypted at all times from transmission of a customer's computer to their server and back and remains safe from unauthorized access even by their staff.

Data privacy solutions cover only a restricted range of applications considering that even private data is subject to sharing (e.g., patient's medical records are shared by doctors, customer's information is shared by e-commerce sites). Thus, the remainder of the paper focuses on the more general problem of data confidentiality and places much emphasis on access right management.

2.2. The attackers

In the light of the preceding section, we can identify three classes of attackers that can threaten data confidentiality:

- *Intruder:* a person who infiltrates a computer system and tries to extract valuable information from the database footprint on disk (DBMS access controls are bypassed).
- *Insider*: a person who belongs to a community of users properly identified by the computer system and the database server and who tries to get information exceeding her own access rights.
- *Administrator*: a person who has enough (usually all) privileges to administer a computer system (System Administrator) or a DBMS (Database Administrator or DBA). These privileges give her the opportunity to tamper the access right definition and to spy on the DBMS behavior.

An Intruder who usurps successfully the identity of an Insider or an Administrator will be considered as such in the rest of the paper.

2.3. Weaknesses of server-based security policies

Traditional database security policies rely on three well established principles [BPS96]: (1) user identification and authentication, that can be supported by mechanisms ranging from simple login/password methods up to smartcard or biometrics device-based methods;

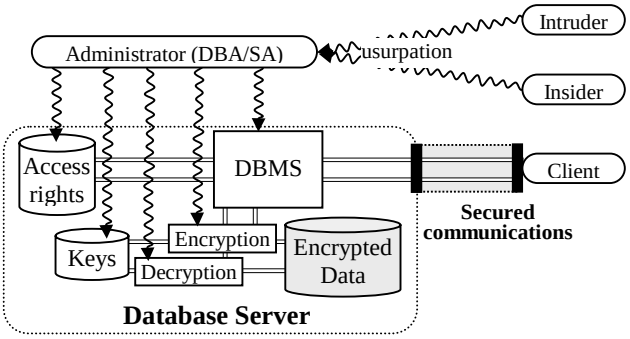

Figure 1: *Database Server approach*

(2) network encryption, that guarantees the confidentiality and the integrity of client/server communications; and (3) server-enforced access control and privilege management. Although these mechanisms are clearly required, they fail to answer all threats identified earlier for two obvious reasons. The first reason is that the confidence on the server never exceeds the confidence the user is ready to place in the DBA. This confidence may vary depending on the users, the Web-hosting companies or the countries but, as far as data confidentiality is concerned, this confidence is generally quite low. The second reason is the increasing number of commercial or institutional sites that are hacked, demonstrating the difficulty of making the hosting computing system secure enough to prevent any intrusion.

Recent attempts have been made to reinforce the server security by encrypting the database. Some commercial DBMSs provide encryption packages to this end [Ora00]. However, if encryption provides an effective answer to attacks conducted on the database footprint by an Intruder, it does not enforce data confidentiality on its own. Indeed, the server being still responsible for query execution and access right management, encryption makes just a bit more tedious the Administrator attacks. In these solutions, the management of cryptographic keys is under the application's responsibility and data is decrypted on the fly by the server at query evaluation time. Thanks to her privileges and to the DBMS auditing tools, the DBA can change the encryption package, can get the cryptographic keys, can modify the access right definition and can even snoop the memory to get the data while it is decrypted. Thus, as Oracle confesses, encryption is not the expected "armor plating" because the DBA (or an Intruder usurping her identity) has all privileges (see Figure 1).

Solutions complementary to database encryption have been recently investigated to guard the DBMS from the DBA. Protegrity [Mat00] introduces a clear distinction between the role of the DBA, administering the database resources, and the role of the SA (Security Administrator), administering user privileges, encryption keys and other related security parameters. This distinction is also made effective at the system level by separating the database server from the security server. The gain in confidence comes from the fact that an attack requires a conspiracy between DBA and SA. Anyway, one must keep in mind that data is still decrypted by the database server at query execution time. An alternative to this approach is to design a secure DBMS engine that restricts DBA privileges in order to make the aforementioned attacks inoperative [HeW01]. This raises the following question "can a DBA administrate a DBMS with restricted privileges?". Unfortunately, DBMS vendors answer today negatively. In addition, this solution suffers from the same security breach as Protegrity regarding data decryption on the server.

The proliferation of solutions to increase database security exemplifies the acuity of the problem. However, existing solutions fail in answering the data confidentiality requirements listed below:

Data confidentiality requirements

1. confidential data must be managed by an auto-administered DBMS to cast off the DBA privileges,
2. this DBMS must be hosted by an auto-administered computing system to cast off the system administrator privileges,
3. this computing system must constitute a Secure Operating Environment (SOE)[1] to cast off any Intruder action.

The traditional database server approach suffer from a strong handicap to meet these requirements because existing DBMSs, as well as the computing systems they rely on, are far too complex, first to be auto-administered and second to constitute a SOE. The first assumption is strengthen by the analysis done in [ChW00] which measures the distance separating current technologies from future self-tuning and zero-admin DBMSs[2]. The worrying numbers regularly published by the Computer Security Institute and the FBI on database vulnerability [FBI01] truly confirms the second assumption.

2.4. Client-based security policies

The weaknesses of the server-based approach to meet the data confidentiality requirements led us to devise client-based solutions. As a preliminary remark, let us notice that the solution presented in section 2.1 to enforce data privacy is typically client-based since the server does nothing but storing encrypted data. Unfortunately, these solutions do not support sharing. Enforcing data confidentiality in a client-based approach means delegating the sharing control to the client devices. However, client-based approaches have been historically disregarded considering that users have themselves the opportunity to hack the client system, and then the sharing control in our context, with total impunity [Rus01].

1 A Secure Operating Environment was defined by [HeW01] as an environment able to manipulate secret data without causing secret leak.

2 Although security is not the concern of [ChW00], the following sentence from the authors is eloquent "*tuning is a nightmare and auto-tuning is wishful thinking at this stage*".

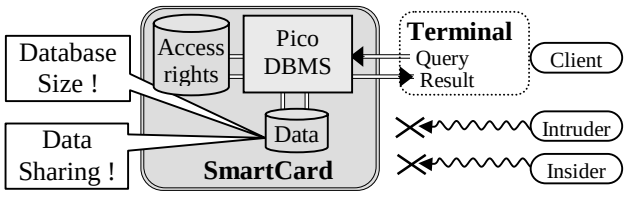

Figure 2: *PicoDBMS, a smartcard client-based approach*

The emergence of smartcard secured client equipment's drastically changes these conclusions. We illustrate the *smartcard-client-based* approach below through practical examples, and discuss to which extent they meet the data confidentiality requirements.

Smartcard is undoubtedly the most secure and cheap computing device today. The strength of smartcard applications regarding data confidentiality is threefold: (1) existing smartcard applications are simple enough to require zero-administration once downloaded in the card, (2) thanks to its hardware architecture making tampering extremely difficult [AnK96, ScS99], the smartcard is probably the best representative of SOE, (3) the high cost of an attack and its practical difficulty (holding the card) must be weighted up with its benefit (the data of a single user can be revealed). A common assumption is that a system can be considered secure if the cost of hacking it exceeds the value of the disclosed information. Conversely, the cost of security for the user is negligible considering the price of a smartcard (a few dollars).

Smartcards become more and more versatile thanks to the emergence of the JavaCard standard [Sun99] and to their increasing computing power. Thus, complex applications can now be downloaded and coexist in smartcards. Simple smartcard applications do not require administration because they are in some sense pre-administered (data schema, user and access rights are hard-coded). The side effect is the lack of extensibility. To circumvent this limitation, ISO has recently promoted a database approach for smartcards, named SCQL [ISO99], which allows for the dynamic declaration of data, users and access rights. Thus, smartcard embedded databases require administration but this task is handled by the cardholder (the data owner) instead of by a DBA, thereby preserving data confidentiality (see Figure 2). The problem of designing database engines dedicated to smartcards (called *smartcard DBMSs* in the sequel) has been extensively studied in [PBV01] and the feasibility of the approach has been recently demonstrated [ABB01]. While smartcard DBMSs pave the way for complex secured client-based applications, they suffer from a tiny storage capacity[3], which confines them to specific applications (typically secured portable folders).

Interesting attempts have been made to push away the smartcard storage limit. The first solution, due to the *WebCard* project [Van98], consists of storing in the smartcard URLs referencing huge, but unprotected, external data. *The Vault* [Big98] extends the WebCard approach by encrypting the documents referenced by URLs. Undoubtedly, the Vault meets the requirements of some applications but it does not constitute a solution from the database point of view. Indeed, the on-board database is seen as a catalog of large encrypted documents rather than as a regular database holding numerous fine-grain objects that can be shared and queried.

2.5. Problem definition

From the preceding discussions, we can identify the different dimensions of the *data confidentiality problem* addressed in this paper.

Data confidentiality problem

- *Privacy and confidentiality*: privacy of personal data and confidentiality of shared data must be guaranteed against attacks conducted by Intruders, Insiders and Administrators.
- *Storage capacity*: the system must not limit the volume nor the cardinality of the database.
- *Sharing capacity*: if required, any data may be shared among multiple authorized users.
- *Query capacity*: any data, whatever its granularity, may be queried through a predicate-based language (typically SQL).
- *Pertinence*: the system must guarantee an acceptable response time to each user, must be scalable and must be economically viable to meet the requirements of large public applications.

3. C-SDA baseline

Before discussing the principles of Chip-Secured Data Access (C-SDA), we first analyze how smartcard client-based solutions answer each dimension of the data confidentiality problem:

- *Privacy and confidentiality:* enforced by the fact that the smartcard is a SOE hosting the data as well as the DBMS engine and that this DBMS is self or user-administered.
- *Storage capacity*: limited by the smartcard stable storage capacity.
- *Sharing capacity*: limited by the need to share physically the same card[4].
- *Query capacity:* depends on the power of the embedded database engine. While query capacity is limited to simple selection in the SCQL standard [ISO99], PicoDBMS [PBV01] demonstrates the feasibility of

[3] Existing smartcards provide around 128KB of EEPROM stable memory, while stable storage is rapidly growing, it will remain quite limited compared with traditional computers.

[4] Typically, a smartcard medical folder has vocation for being shared among multiple users (patient, doctors, pharmacists, …) but a single user is active at a time.

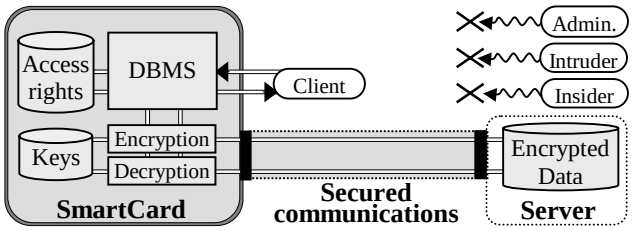

Figure 3: *C-SDA sphere of confidentiality*

embedding powerful query engines supporting selection, join, grouping and aggregate calculus.

- *Pertinence:* well suited in terms of performance (the smartcard DBMS is mono-user and works on a reduced set of data), of scalability (one smartcard per user) and of price (a few dollar per smartcard).

Given these statements, solving the data confidentiality problem sums up to bypass the storage and sharing limitations without hurting the other dimensions. The concept of server typically addresses the storage and sharing issues. Thus, let us consider to which extent the sphere of security provided by the smartcard could be extended to a remote server holding encrypted data. As discussed in section 2.3, the first security breach of the server-based approach comes from the fact that data is decrypted by the server at query execution time. Assuming that the DBMS query engine remains hosted by the smartcard, this eliminates the need to decrypt data on the server side. The second security breach of the server-based approach comes from the fact that access rights are enforced by the server and administered by an untrusted DBA. Let us assume that the DBMS access right manager remains hosted by the smartcard, the DBA (or an Intruder usurping her identity) is no longer able to abuse them.

Can we infer from the preceding assumptions that a server acting as an encrypted repository for a smartcard DBMS can integrate the smartcard's sphere of security (i.e., while keeping the level of confidence unchanged)? The answer is obviously 'no' since the server is not hosted by a SOE. Typically, an Intruder may conduct destructive or deny of service attacks on the server. However, privacy and confidentiality are preserved thanks to encryption[5].

In the same spirit, since the data that flows from the server to the smartcard DBMS is encrypted, can we infer that the communication channel is part of the smartcard's sphere of security? Again, the answer is 'no' since the communication channel may undergo several forms of attacks. At first sight, privacy and confidentiality are preserved anyway. However, an Insider may compare the encrypted data issued from the server with the query result that appears in plain text on its terminal. This may help her to conduct a *known plain text cryptanalysis* in order to deduce the encryption keys hosted by the smartcard. Thanks to these keys, the Insider may attempt to access data exceeding her own access rights. Indeed, the Insider may have the privilege to see the result of a query computed by the smartcard DBMS on data that is outside the scope of her privilege[6]. Consequently, re-encrypting the communication with a session key protocol (e.g., SSL) is necessary to enforce confidentiality[7].

The baseline of C-SDA is then to build a sphere of confidentiality encompassing the smartcard DBMS, the server and the communication channel linking them. The resulting functional architecture is pictured in Figure 3 and roughly works as follows. Each smartcard is equipped with a database engine managing access rights, query evaluation and encryption. When the user issues a query, the smartcard DBMS first checks the user's access rights and, in the positive case, gets the data from the server, decrypts it, executes the query and delivers the result to the terminal.

The server component of the C-SDA architecture is an answer to the storage and sharing dimensions of the data confidentiality problem. However, one may wonder about the impact of this answer on the other dimensions of the problem. The main question is whether the smartcard DBMS technology can conciliate complex queries, large volumes of data and performance, considering the inherent hardware constraints in the smartcard. The second question relates to data confidentiality and concerns the level of confidence that can be placed in data encryption (with respect to data hosted by the smartcard) and the granularity of sharing compatible with encryption. The next sections investigate these two issues.

4. Query Management

In order to evaluate the technical soundness of the C-SDA architecture in terms of query evaluation feasibility and efficiency, we first recall the smartcard characteristics that are relevant to this issue. Then, we propose a query evaluation principle that matches these smartcard characteristics whatever the volume of data involved in a query.

4.1. Smartcard characteristics

Current smartcards include in a monolithic chip, a 32 bits RISC processor at about 30 MIPS, memory modules (of about 96 KB of ROM, 4 KB of static RAM and 128 KB of EEPROM), a serial I/O channel (current bandwidth is around 9.6Kbps but the ISO standard allows up to 100Kbps) and security components preventing tampering [ISO98]. ROM is used to store the operating system, fixed data and standard routines. RAM is used to manage the execution stack of programs and to calculate results. EEPROM is used to store persistent information.

[5] The confidence that can be placed on data encryption itself will be more deeply discussed in section 5.

[6] For instance, a user may be authorized to consult the result of an aggregation without be aware of the elementary values from which this aggregation is computed.

[7] A side effect of an SSL-like protocol is to guarantee at the same time a mutual identification/authentification of the client and the server as well as the integrity of messages.

EEPROM has very fast read time (60-100 ns/word) comparable to RAM, but a dramatically slow write time (1 to 5 ms/word).

The main constraints of current smartcards are therefore: (i) the very limited storage capacity; (ii) the very slow write time in EEPROM and (iii) the extremely reduced size of the RAM. On the other hand, smartcards benefit from a very high security level and from a very powerful CPU with respect to the other resources. This makes the smartcard an asymmetric computing architecture which strongly differs from any other computing devices.

The current trends in hardware advances are on: (i) augmenting the CPU power to increase the speed of cipher algorithms, (ii) augmenting the capacity of the stable storage and (iii) augmenting the communication bandwidth between the chip and the card-reader[8]. More details on existing smartcard platforms and their evolution can be found in [Tua99, PBV01].

4.2. Query evaluation principle

A naive interpretation of the C-SDA architecture depicted in Figure 3 is to consider that the server acts as a persistent encrypted virtual memory which is accessed by the smartcard DBMS during query evaluation, any time a data item is requested for computation. Such an architecture would suffer from disastrous performance because it would incur a prohibitive communication cost (one call per data item) and I/O cost (traditional server optimizations become irrelevant). It may even happen that the same data be loaded several times from the server if the smartcard DBMS cannot keep enough local resources to cache it. Last but not least, the smartcard hardware constraints impose to design very specific query evaluation strategies. While ad-hoc strategies have been shown convenient in the context of small-embedded databases, their algorithm complexity renders them totally inappropriate for large databases [PBV01].

Thus, new query evaluation strategies that better exploit the computational resources available on the server and even on the terminal must be devised. This leads to split a query Q into a composition of the form $Q_s \circ Q_c \circ Q_t$, where Q_s, Q_c and Q_t denote respectively the sub-query evaluated on the server, the card and the terminal. The imbalance between the smartcard, the server and the terminal in terms of computing resources advocates pushing the biggest part of the computation down into Q_s and Q_t. However, the imbalance between these same components in terms of security leads to the following compromise:

- *Server subquery (Q_s):* the server must execute the largest part of the query as far as confidentiality is not compromised. That is, any predicate that can be evaluated on the encrypted form of the data must be pushed down to the server. To simplify things, we consider below that predicates based on an equality comparator {=, ≠} satisfy this condition[9]. In the sequel, we call these predicates *equi-predicates* in opposition to *inequi-predicates* based on inequality operators {>, ≥, <, ≤}.
- *Smartcard subquery (Q_c):* the smartcard DBMS is responsible for filtering the result of Q_s to evaluate all predicates that cannot be pushed down to Q_s and for computing aggregation functions if required. The terminal cannot participate to this evaluation because the data flow resulting from Q_s may go beyond the user's access rights.
- *Terminal subquery (Q_t):* due to the confidentiality consideration mentioned earlier, the terminal can only evaluate the part of the query related to the result presentation. Typically, it can handle the sort and the distinct operators, if requested by the user.

The challenge in decomposing Q into $Q_s \circ Q_c \circ Q_t$ is twofold. First, the global evaluation must meet the *pertinence* requirement in terms of performance and scalability. Second, Q_c must accommodate the smartcard's hardware constraints. Query evaluation on the smartcard precludes the generation of any intermediate results since: (i) the RAM capacity cannot accommodate them, (ii) RAM cannot overflow into EEPROM due to the dramatic cost of EEPROM writes and (iii) intermediate results cannot be externalized to the terminal without hurting confidentiality.

To explain how this challenge can be tackled, we will consider unnested SQL queries composed by the traditional Select, From, Where, Group by, Having and Order by clause and we will reason about them in terms of relational algebra. Let us first introduce some notations:

R, S, …U: relations involved in the query

R.a : attribute *a* from relation R

$\pi_{p,fp}$: projection operator, where *p* denotes the list of attributes to be projected and f_p denotes the list of aggregate functions to be computed before projection

χ: cartesian product operator

σ_q: selection operator, where *q* denotes the selection qualification: q is expressed in conjunctive normal form as follows: $C_1 \wedge C_2 \ldots \wedge C_n$, each condition C_i being of the form $(P_1 \vee P_2 \ldots \vee P_k)$, each predicate P_j being of the form (R.a θ value) or (R.a θ S.b), with $\theta \in \{=, \neq, >, \geq, <, \leq\}$.
C^q denotes the set $\{C_1, C_2 \ldots, C_n\}$ of conditions participating in q.
P^{Ci} denotes the set $\{P_1, P_2, P_k\}$ of predicates participating in C_i.

[8] These trends are partly explained by market perspectives on delivering multimedia objects (e.g., an mp3 flow) that can be decrypted on the fly by the card of a subscriber.

[9] This assumption means that any couple of data subject to comparison is encrypted with the same key. Data encryption is more deeply detailed in section 5.

γ_g: grouping operator, where *g* denotes the list of attributes on which the grouping applies

$\eta_{c, f\eta}$: having operator, where *c* denotes the having qualification and f_η the list of aggregate functions on which *c* applies

ϕ: presentation operators: sort, duplicate removal

E (resp. D): encryption (resp. decryption) operator

According to the operational semantics of SQL, an unnested query Q is equivalent to the following formula:

$Q = \phi(\pi_{p, fp}(\eta_{c, f\eta}(\gamma_g(\sigma_q(R\chi S\chi \ldots U)))))$

Under the assumption made about database encryption, that is: $\forall d_i,d_j$, $E(d_i) = E(d_j) \Leftrightarrow d_i = d_j$, we can infer that the largest part of Q that can be delegated to the server is:

$Q_s = \pi_{ps}(\gamma_g(\sigma_{qs}(R\chi S\chi \ldots U)))$, with

$C^{qs} \subseteq C^q$ and $C_i \in C^{qs} \Rightarrow \forall P_k \in P^{Ci}, \theta \in \{=, \neq\}$,

$ps = p \cup g \cup l_p \cup l_\eta \cup l_{qc}$, where

l_p is the list of attributes referenced by f_p

l_η is the list of attributes referenced by f_η

l_{qc} is the list of attributes referenced by the conditions $\in (C^q - C^{qs})$

This leads to define Q_c and Q_t as follows:

$Q_c = \pi_{p,fp}(\eta_{c,f\eta}(\sigma_{qc}(D(Q_s))))$, with $C^{qc} = C^q - C^{qs}$

$Q_t = \phi(Q_c)$

Roughly speaking, this means that equi-selection, equi-join and group by are computed on the server while inequi-predicates, aggregation and predicates over aggregate results have all to be evaluated on the smartcard. Figure 4 sketches the algorithm in charge of the evaluation of Q_c in the smartcard. This algorithm is self-explanatory. It consumes one tuple at a time from Q_s and requires a single buffer to cache the tuple of Q_c under construction. Note that if an aggregation is to be computed, the tuples of Q_s have already been grouped by the server and then do not need to be reordered in the smartcard. Thus, it clearly does not produce intermediate results and fulfills the second part of the decomposition challenge. As far as performance and scalability are concerned, two remarks have to be made. First, the cost incurred by the security mechanism (i.e., decryption) is spread over all users' smartcards instead of being concentrated on the server, thereby improving scalability. Second, the in-card computation is not CPU bound (powerful processor, low algorithm complexity) nor memory bound (one tuple at a time) but communication bandwidth bound. Let us remind that the communication channel between a smartcard and the card reader range from 9.6Kbps to 115Kbps maximum. The output-channel is not the limiting factor because it can deliver the resulting tuples at a reasonable rate (i.e., up to bandwidth/sizeof($\pi_{p,fp}$(result)). However, the input-channel may become the bottleneck if the ratio $|Q_c|/|Q_s|$ is low, because this ratio decreases in the same proportion the output rate. To illustrate the problem, let assume an inequi-join between relations R and S having a selectivity factor of 0.01. All tuples resulting from the cartesian product $R\chi S$ computed in Q_s will traverse the input-channel while only 1% of relevant tuples will traverse the output-channel. Optimization techniques are clearly required to handle this problem. This issue is addressed in section 6.2

5. Confidentiality and encryption

This section fixes a set of encryption rules required to answer accurately the data confidentiality problem. Then, it shows how the smartcard device can be exploited to increase the privacy and confidentiality of a reduced set of

```
∅ → result                                          // buffer caching the tuple of Qc being constructed
                                                    // result owns one attribute for each element in ps
0 → groupCard                                       // cardinality of a group
For each t ∈ Qs, do
    D(t) → t                                        // decrypt tuple t
    If σqc(t) = true, then                          // check selection qualification
        If g = ∅ then                               // no grouping
            send πp(t) to the terminal
        Else                                        // group by clause
            If πg(t) ≠ πg(result), then             // a new group is detected
                If ηc, fη(result) = true then       // check having qualification
                    send πp, fp(result) to the terminal
                ∅ → result
                0 → groupCard
            For each aggi ∈ (fp ∪ fη), do           // t belongs to the same group as result
                aggi(t) → result.aggi               // evaluate aggregation functions[1]
            groupCard + 1 → groupCard
send πp, fp(result) to the terminal                 // send the last tuple under construction
```

[1] Each aggregate function to be computed uses one attribute of *result* as a state variable. Assume the *avg* function is to be computed, *avg(t)* → *result.avg* sums up the current attribute value of *t* into *result.avg* while π_{avg}*(result)* divides this sum by the cardinality *groupCard* of the current group.

Figure 4: *Q_c in-card algorithm*

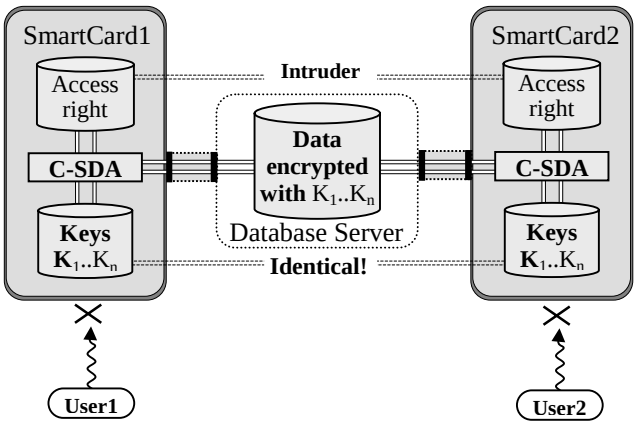

Figure 5: *Encryption keys versus access rights*

highly sensitive data. Finally, it addresses the management of access rights and concludes with a discussion on the limits of the solution.

5.1. Database encryption

From the beginning of the paper, we have considered implicitly that the whole database was encrypted. Obviously, only the confidential part of it needs to be encrypted. For the sake of simplicity, we will not discuss further the cohabitation between clear and encrypted data because it does not present a major technical difficulty. Thus, we concentrate in the sequel on the quality of the database encryption.

As stated in section 3, the level of confidence placed in C-SDA is strongly related to the confidence placed in the data encryption strategy. In our context, the following *data encryption rules* apply:

Key insulation rule: *encryption keys must remain confined in the smartcard.*
This rule is required to prevent any attack conducted by the DBA, an Intruder and even an Insider. Consequently, data encryption and decryption must be handled by the smartcard as well. Note that the cardholder herself has no way to access the encryption keys hosted by its own card. These keys remain under the exclusive control of the in-card C-SDA software.

Sharing rule: *encryption must remain orthogonal to access rights.*
As explained in section 2.1, encryption alone is sufficient to implement data privacy, assuming that each user encrypts her own data with a secrete key. Thus, encryption acts as a binary access right granting or revoking all privileges to the user depending on whether or not she knows the secrete key. On the contrary, data confidentiality requires sharing the same key(s) among a community of authorized users. Unfortunately, there is no bijection between encryption and access rights because these two mechanisms do not operate at the same level of granularity. Access rights are commonly attached to database views to share data at a very fine-grain level. The sharing is thus predicate-based. Achieving the same level of sharing with encryption alone would require defining as many encryption keys as possible SQL qualifications. Access rights can even be defined on virtual data (e.g., aggregate calculus) that obviously cannot be encrypted. Consequently, encryption rules must remain orthogonal to access right management. Assuming key K_i is used to encrypt data shared among multiple users, K_i must be hosted by the smartcard of each of these users but the key usage is restricted to the in-card C-SDA software that controls access rights (see Figure 5).

Computation rule: *encryption must preserve attribute equality comparisons.*
Encrypting the database on a tuple, column, or relation basis precludes any computation to occur on the server side without decrypting the data first. Thus, the encryption must be done on an attribute basis. In addition, as stated in section 4.2, the minimal assumption required to allow server computation without decryption is $\forall d_i,d_j$, $E(d_i) = E(d_j) \Leftrightarrow d_i = d_j$. Obviously, this assumption is required only for couple of data that may be subject to comparison. Fortunately, most block encryption algorithms (e.g., DES [NIS93]) satisfy this assumption.

Stronger assumptions on the encryption method might increase the range of computations that can be delegated to the server. *Privacy homomorphisms* (PH) introduced in [RAD78] allow to perform *some computation* on encrypted data. For instance, the PH proposed in [Dom97] preserves the basic four arithmetic operations, but equality predicates can no longer be checked. Order-preserving PH and more generally PH maintaining range information can also be devised but they drastically reduce the robustness of the encryption method [Sch96, Dom97].

Performance rule: *encryption must be symmetric and client-based.*
As stated in section 4.2, client-based encryption/decryption is the first guarantee of scalability. Moreover, considering the large volume of data to be encrypted/decrypted, we promote the use of symmetric encryption algorithms (e.g., DES) because they are more robust and much more efficient (three orders of magnitude faster) than asymmetric algorithms (e.g., RSA[RSA93]). The secure diffusion of secrete keys is the major problem of symmetric algorithms in traditional architectures. This problem is solved by nature in the C-SDA context, thanks to the smartcard device that provides a secure key hosting. Thus, keys are distributed among users along with smartcards.

Multi-key encryption rule: *encryption must exploit as much different keys as possible.*
Increasing the number of keys in the encryption process has two main advantages. First, it makes statistical attacks more difficult to conduct. Second, it reduces the amount of data that will be disclosed if the aforementioned attack succeeds. Different techniques can be envisioned to use multiple keys while respecting the computation rule. A

first solution is based on vertical fragmentation, that is encrypting with different keys the columns that will never participate to an equi-join (e.g., Person.name and Person.age). A second solution is based on horizontal partitioning, that is encrypting with different keys the attribute values of the same column thanks to a one-way hash function. For instance, *Key(h(a))* can be used as a parameter to encrypt the attribute value *a*, and the value $(h(a), E_{key(h(a))}(a))$ is stored in the database in place of *a*. Note that this solution respects the computation rule. Other techniques may be used but space limitations forbid their presentation in this paper.

5.2. Sensitive data

The persistent storage capacity of the smartcard introduces new alternatives to achieve data privacy and confidentiality. Basically, highly sensitive data may be stored in the smartcard instead of in the server, thereby making it ultimately robust against attacks. For instance, identification information could benefit from this property (e.g., name, social security number, …), so that the database in the server is depersonalized. This technique however introduces three issues: (i) how to integrate this sensitive data in the query evaluation process, (ii) how to guarantee its durability and (iii) how to share it if it is used by multiple users.

To make the integration of sensitive data in the query evaluation process as simple as possible, we propose to group sensitive data in *sensitive domains* (i.e., set of data items without duplicates) and to store indices referencing these domain values in place of the corresponding data in the server. This technique can be formally considered as a particular encryption method $E(data) \rightarrow domain\_index$ that is definitely unbreakable without the smartcard. The integration in the query evaluation process is straightforward since *E* satisfies the computation rule.

The complexity of enforcing sensitive data durability depends on whether a sensitive domain is static or dynamic. Static domain can simply be duplicated on any secure storage device (e.g., a backup smartcard). Dynamic domains are trickier to manage, especially if they are shared among multiple users. The solution is to leave an encrypted copy of the domain on a backup server (preferably distinct from the database server) and to synchronize this encrypted backup with the domain copy residing on a smartcard at each connection. One may wonder about the benefit of this method compared with leaving the data in their original form on the database server. The benefit is actually twofold. First, the database and the sensitive domains are located on two separate servers thereby increasing the complexity of attacks. Second, the backup copy of the domain doesn't need to participate in the query evaluation. Thus, it can be encrypted with stronger methods (e.g., a different key for each domain entry) since it is not affected by the computation rule.

5.3. Access Right Management

As stated in section 3, access right management must be embedded in the smartcard to prevent any DBA tampering. Since access rights are commonly defined on database views, the views have to be managed by the smartcard as well. This raises the problem of access rights and views evolution. If the smartcard is responsible for controlling access rights and views, their definitions have to be securely stored in a server accessible by all smartcards. Modeling the list of access right definitions and the list of database view definitions as two dynamic and shared sensitive domains brings a simple and accurate solution to this problem.

The crucial question regarding access rights is who is responsible for granting/revoking them. The common rule in database systems is that the owner of an object inherits this responsibility. In practice, the unlimited privileges of the DBA contradict this rule. Using C-SDA, the DBA conserves all her privileges, so that she can administer the database server but she has no way to break the data confidentiality, as long as she has no access to the user's smartcard. As a conclusion, a C-SDA user is definitely the unique holder of her data and she decides if she wants to exhibit them and to whom.

5.4. Limits of the solution

One may wonder whether this combination of hardware (the smartcard) and software (C-SDA) security components constitutes the ultimate protection against data confidentiality attacks. In this respect, we must state the limits of the solution.

First, an Intruder can infiltrate the user's terminal in order to snoop the query results that are presented in plain text to the user or to alter the query expression sent by the terminal to the smartcard before processing. By this way, the Intruder may try to execute a query selecting more data than expected by the user and snoop them. Anyway, such attack can reveal only data being in the user's access right scope. This threat cannot be avoided by any security architecture, unless the terminal is itself secure. To secure the terminal, both the screen and the keyboard must be part of the SOE, like in today's smartcard payment devices. This solution can be suitable for users willing only to consult their data but is inadequate as soon as computation is required on these data.

Second, an Intruder or an Administrator may try to tamper the database footprint on disk in the hope of decrypting unauthorized data thanks to the smartcard (e.g., by permuting columns encrypted with the same key). This attack can be beat off at the expense of introducing a checksum attribute in each tuple. Anyway, the scope of this attack is limited by the use of multiple encryption keys and by the fact that the attacker must be a cardholder.

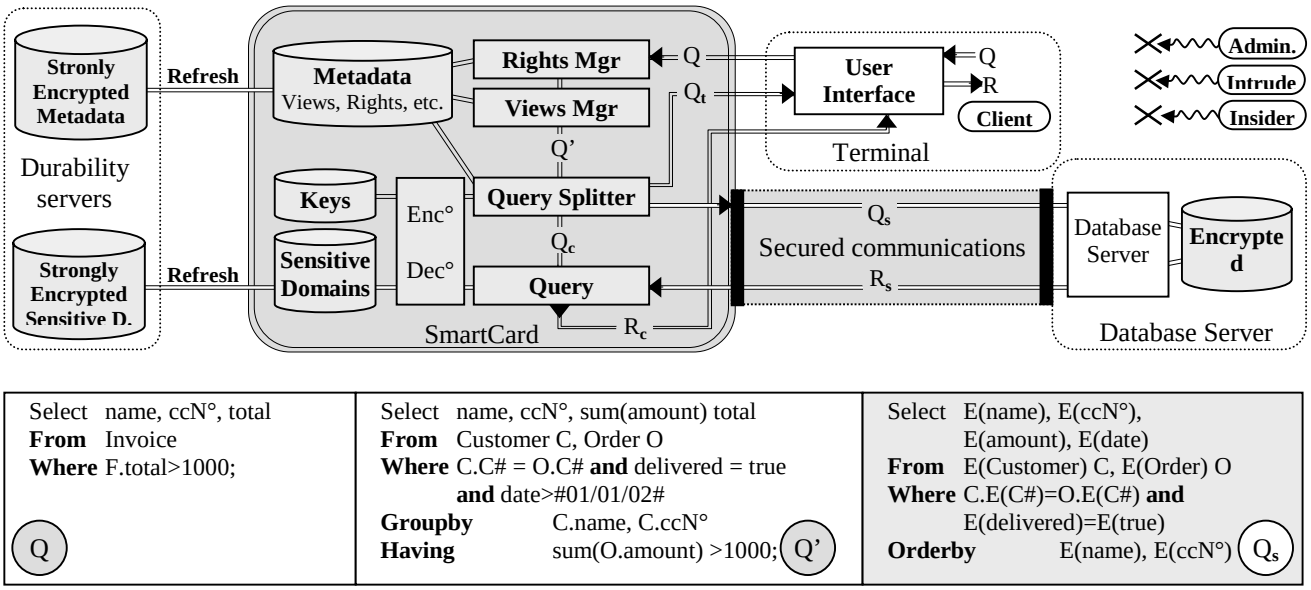

Figure 6: *C-SDA architecture and scenario*

6. C-SDA scenario

This section presents a complete C-SDA scenario illustrating the step by step evaluation of a simple query on a corporate database. Confidentiality and performance issues are discussed along the scenario unfolding.

6.1. Query Execution with C-SDA

Let us consider a business database application where the invoice department is willing to bill invoices having a total amount greater than 1000 US$. The privilege of the invoice department clerk is assumed to be restricted to the *select* operation on the view *Invoice*. This view calculates for each customer, the total amount of delivered orders since January 2002. This view prevents an untrusted clerk to access confidential order-lines. Figure 6 shows a query *Q*, issued by the clerk and expressed on the view Invoice, and the query *Q'*, resulting from the view resolution and expressed on the base relations Customer and Order. The execution of query *Q* comprises the following steps (see Figure 6).

1. *Metadata refreshing:* At connection time (i.e., when the user inserts her smartcard to the card reader), C-SDA contacts the durability server(s) in order to refresh its local copy of relation and view definitions, access right information's and sensitive data.
2. *Access Right checking and view resolution:* The *access right manager* checks that query *Q* involves only authorized relations and views. Then, the *view manager* merges *Q* with the view definition to produce *Q'*.
3. *Query splitting:* The *query splitter* splits query *Q'* into Q_s (step 4), Q_c (step 6) and Q_t (step 7) conforming to the decomposition principle detailed in section 4.2. Q_s is then rewritten in an "encrypted SQL form", that is relation names, attributes and constant are encrypted (encryption is denoted by *E()* in Figure 6). Note that the encrypted form of a well-formed SQL query is a well-formed SQL query.
4. Q_s *transmission and execution:* The encrypted query Q_s is sent to the database server using a secured communication protocol. Secured communication is mandatory to avoid any falsification of Q_s before transmission (which may permit a malicious user to access more data than granted). The database server optimizes and processes Q_s as any traditional query, without being aware of encryption. The query execution plan of Q_s is pictured in Figure 7.
5. Q_s *Result transmission:* The encrypted result R_s is sent back to the smartcard using a secured communication protocol. As explained in section 3, secured communication is mandatory here to avoid plaintext cryptanalysis on the terminal.
6. Q_c *execution:* The encrypted result R_s is processed in a pipelined fashion by C-SDA following the algorithm presented in Section 4.2. Figure 7 presents the query trees of Q_s and Q_c. As shown in the Figure, data decryption is pushed up to the query tree of Q_c as far as possible. This avoids decrypting all attributes of tuples that do not participate in the final result. For instance, the attribute *date* is first decrypted in order to check the predicate *date>01/01/02*. Tuples which survive this selection are further partially decrypted in order to compute the aggregation and to check the having clause. Finally, the qualified tuples are fully decrypted before being sent to the user. Assuming that ccN° (credit card number) is a sensitive domain, decryption of this attribute follows the principle described in section 5.2.
7. R_c *delivering and* Q_t *execution:* Finally, once decrypted, the tuples participating in the final result are sent to the terminal where the *distinct* and/or *sort* clauses potentially present in Q_t are applied.

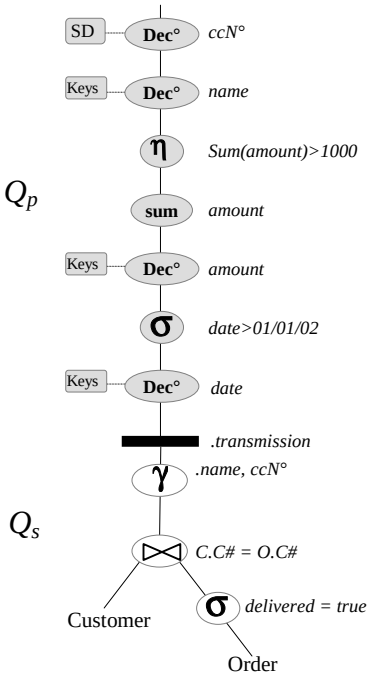

Figure 7: *Q_s and Q_c query tree*

6.2. Optimization Issues

The performance problem pointed out in section 4.2 is exemplified in this scenario. Assume that only 1% of orders satisfies the selection on *date*, 99% of the R_s tuples sent back to the smartcard are irrelevant, generating a bottleneck on the smartcard input-channel. In the following we sketch a solution alleviating this problem. Other optimizations of the C-SDA architecture can undoubtedly be devised but are out of the scope of this paper.

The solution proposed relies on a multi-stage cooperation between the smartcard and the server aiming at minimizing the flow of data traversing the smartcard input-channel. The intuition is to use the smartcard as a secured co-processor which can evaluate inequi-predicates on demand. The evaluation of each inequi-predicate is handled by a pre-processing query that takes as input a collection of encrypted values issued by the server, decrypts them, evaluates the inequi-predicate and sends back the matching values in their encrypted form to the server. On the server side, this result is integrated in the initial query thanks to a semi-join operator.

Let us illustrate the concept of pre-processing query on our scenario. The objective is to evaluate the inequi-predicate *date>01/01/02* on a data set smaller than R_s. Ideally, this predicate should be evaluated on the subset of Order tuples satisfying the selection *delivered = true*. This situation would be optimal in two respects. First, it would minimize the data flow traversing the smartcard input-channel. Second, it would minimize the cost of evaluating query Q_s on the server side by pushing up selections before joins in the regular way. Pre-processing is the way to achieve this goal. A pre-processing query PQ_s is first generated by the smartcard query splitter to get from the server the tuples resulting from π_{date} ($\sigma_{delivered=true}$ (Order)). Then, the smartcard query processor computes T = E(($\sigma_{date>01/01/02}$ (D(PQ_s)))), the content of which is stored in a temporary relation on the server side. Finally, the smartcard query splitter adds the semi-join predicate T.E(date) = Order.E(date) to the initial query Q_s and sends it to the server for computation (see Figure 8). This strategy applies as well to inequi-join predicates, and can be exploited iteratively for all inequi-predicates involved in the same query.

7. Conclusion and future prospects

The tremendous development of Internet applications prompts citizens and companies to put always more data accessible through the Web. Preserving data confidentiality in this context is becoming one of the most challenging issues for the database community.

This paper addresses this issue and makes the following contributions. First, it gives an in-depth analysis of the security solutions proposed in the database field and capitalizes on strengths and weaknesses of these approaches to clearly state the dimensions of the *data confidentiality problem*. Second, it proposes the *Chip-*

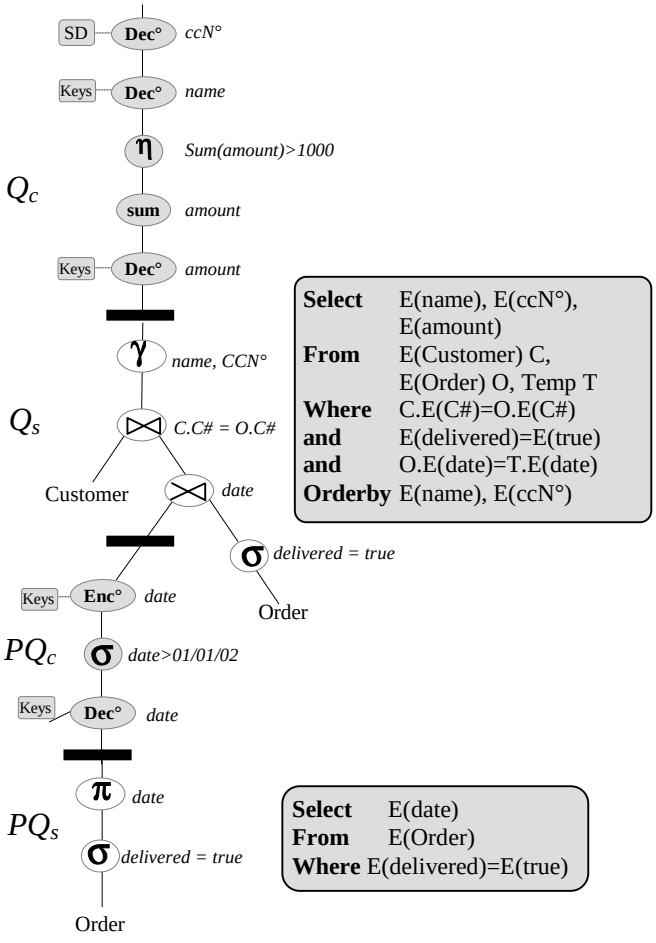

Figure 8: *Optimized query tree*

Secured Data Access (C-SDA) principle as a solution to this problem. The main idea underlying C-SDA is to insulate data encryption, query evaluation and access right management in a Secured Operating Environment (SOE). Third, query evaluation and optimization techniques are proposed to tackle the strong hardware constraints introduced by the most popular representative of SOE, the smartcard.

C-SDA is being validated in the context of a B2B project founded by the French ANVAR agency. This project, started in january 2002, aims at sharing an EDI database between business partners. Depending on the business model, this database can be hosted by a DSP or by one of the partner, but the data confidentiality requirements remain the same.

C-SDA has been devised in the context of smartcards because of its wide acceptance and its well-established technology. However, the C-SDA architecture can be adapted to other secured computing devices. For instance, the Dallas i-button [iBu02] provides a security level comparable to smartcards but benefits from a higher bandwidth with the terminal. Such technology could be exploited to alleviate the performance problem induced by inequi-predicates. The apparition of high-end secure coprocessor [Swe99] may, in the future, render viable tamper-resistant server-based solutions that are technically unfeasible today for performance and scalability reasons. In all situations, the interactions between the C-SDA software hosted by the secured device and the encrypted data store will remain the same, but with different technical tradeoffs.

Other important open issues concern the extension of C-SDA to more complex data models, query languages and client/server interactions. More generally, we believe that tamper-resistant devices will have an increasing influence on the way security solutions for information systems will be devised.

Acknowledgments

The authors wish to thank Philippe Bonnet and Ioana Manolescu for their helpful comments on this paper.

References

[ABB01] N. Anciaux, C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez, "PicoDBMS: Validation and Experience", *Int. Conf. on VLDB*, 2001.

[AnK96] R. Anderson, M. Kuhn, “Tamper Resistance – a Cautionary Note”, *USENIX Workshop on Electronic Commerce*, 1996.

[Big98] P. Biget “The Vault, an Architecture for Smartcards to Gain Infinite Memory”, Smart Card Research and Advanced Application Conference (CARDIS'98), 1998.

[Bla95] M. Blaze, “High-Bandwidth Encryption with Low-Bandwidth Smartcards”, AT&T Bell Labs, 1995. (ftp://ftp.research.att.com/dist/mab/ card cipher.ps)

[BPS96] A. Baraani, J. Pieprzyk, R. Safavi-Naini "Security In Databases: A Survey Study", 1996. citeseer.nj.nec.com/baraani-dastjerdi96security.html

[CaB02] The Caspio Bridge DSP. www.caspio.com/bridge.htm

[ChW00] S. Chaudhuri, G. Weikum, “Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System”, Int. Conf. on VLDB, 2000.

[Dom97] J. Domingo-Ferrer, “Multi-application smart cards and encrypted data processing”, Future Generation Computer Systems, (13), 1997.

[eCr02] The eCriteria DSP. www.ecriteria.net

[FBI01] Computer Security Institute, "CSI/FBI Computer Crime and Security Survey". www.gocsi.com/forms/fbi/pdf.html

[HeW01] J. He, M. Wang, “Cryptography and Relational Database Management Systems”, Int. Database and Engineering and Application Symposium, 2001.

[iBu02] The crypto iButton with Java - (http://www.ibutton.com/)

[ISO98] International Standardization Organization (ISO), *Integrated Circuit(s) Cards with Contacts – Part 1: Physical Characteristics*, ISO/IEC 7816-1, 1998.

[ISO99] International Standardization Organization (ISO), *Integrated Circuit(s) Cards with Contacts – Part 7: Interindustry Commands for Structured Card Query Language (SCQL)*, ISO/IEC 7816-7, 1999.

[Mat00] U. Mattsson, *Secure.Data Functional Overview*, Protegity Technical Paper TWP-0011, 2000. (http://www.protegrity.com/White_Papers.html)

[Mic02] The Microsoft.Net Passport. www.passport.com

[NIS93] National Institute of Standards and Technology, *Announcing the Data Encryption Standard (DES)*, FIPS PUB 46-2, 1993.

[NIS94] National Institute of Standards and Technology, Announcement of Weakness in the Secure Hash Standard, 1994.

[Ora99] Oracle Corp., *Database Security in Oracle8i*, 1999. otn.oracle.com/deploy/security/oracle8i

[Ora00] Oracle Corp., *Advanced Security Administrator Guide*, Release 8.1.7, 2000.

[PBV01] P. Pucheral, L. Bouganim, P. Valduriez, C. Bobineau, "PicoDBMS: Scaling down Database Techniques for the Smartcard", *VLDB Journal*, 10(2-3), 2001.

[Qck02] The Quickbase DSP. https://www.quickbase.com/

[RAD78] R. L. Rivest, L. Adleman and M. L. Dertouzos, “On Data Banks and Privacy Homomorphisms”, Foundations of Secure Computation. Academic Press, 1978.

[RSA93] RSA Laboratories, *PKCS #1: RSA Encryption Standard*, RSA Laboratories Technical Note, 1993.

[Rus01] Ryan Russel et al., *Hack Proofing Your Network*, Syngress Publishing, 2001.

[ScS99] B. Schneier, A. Shostack, “Breaking up is hard to do: Modeling Security Threats for Smart Cards”, *USENIX Symposium on Smart Cards*, 1999.

[Sch96] B. Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons, 1996.

[Sky02] SkyDesk : @Backup. www.backup.com/index.htm

[Swe99] S.W. Smith, S.H. Weingart, Building a High-Performance, Programmable, Secure Coprocessor, Computer Networks (31) - 1999

[Sun99] Sun Microsystems, *JavaCard 2.1 Application Programming Interface Specification*, JavaSoft documentation, 1999.

[Tua99] J.-P. Tual, “MASSC: A Generic Architecture for Multiapplication Smart Cards”, *IEEE Micro Journal*, N° 0272-1739/99, 1999.

[Van98] J.J. Vandewalle, P. Biget, “Extended Memory Card”, European Multimedia Microprocessor Systems and Electronic Commerce Conf., 1998.