Vertical Fragmentation Security Study in Distributed Deductive Databases

DOREL SĂVULEA AND NICOLAE CONSTANTINESCU

ABSTRACT. Aiming to extend relational databases while preserving their declarative programming style, deductive databases support a rule-based language capable of expressing complete applications. Distributed deductive databases have been intensively studied in the past decades mainly because they provide a high level for protecting voluminous data with low costs. In such systems one of the most important processes is the fragmentation of data and rules since it represents a basis for the allocation process. It also needs to be secured all the remote database fragments and the infrastructure. This paper studies the vertical fragmentation for such a system proposing a security method which provides authentication.

2010 Mathematics Subject Classification. Primary 14G50; Secondary 18A15.
Key words and phrases. encryption system, database security, ruled-based language, distributed deductive databases.

1. Introduction

Deductive databases have resulted from relational databases by adding rules that includes deductive capabilities. A deductive database system is a combination of a conventional database containing facts, a knowledge base containing rules, and an inference engine which allows the derivation of information implied by the facts and rules. Commonly, the knowledge base is expressed in a subset of first-order logic and either a SLDNF or Datalog inference engine is used. Deductive databases provide a declarative, logic-based language for expressing queries, reasoning, and complex applications on databases [16].

The most important advantages of deductive databases languages are [17]:

• The goals execution order does not depend on their order in the rules writing; the execution order is controlled by the system and not by the programmer.
• The selection between forward-chaining and backward-chaining execution is automatic; it is also controlled by the system and not by the programmer.

These advantages not only enhance data independency since the resulting code can be reused even if physical changes have been made to the database, but they also ease the programmer tasks.

A distributed database is a database physically stored in two or more computer systems. Although geographically dispersed, a distributed database system manages and controls the entire database as a single collection of data. If redundant data are stored in separate databases due to performance requirements, updates to one set of data will automatically update the additional sets in a timely manner [13]. The most popular distributed database system is maybe the Internet’s domain name system (DNS).

Received June 21, 2011. Revision received August 18, 2011.
A deductive database can be definite or disjunctive. The main difference between these two types is that the latter can capture indefinite information. Indefinite information is information that is possibly true and not unconditionally true. A disjunctive system allows disjunction of predicates to appear in the head of any rule from the database [4].

All the operations upon a database system are named transactions. A transaction is a transformation of state which has the properties of atomicity, durability, and consistency [6]. These four properties are named A.C.I.D and are followed also by the distributed deductive database systems’ transactions:

- **Automaticity**: when an update occurs to a database, either all or none of the update becomes available to anyone beyond the user or application performing the update, which means that only a fragment of the update cannot be placed into the database, should a problem occur with either the hardware or the software involved.
- **Consistency**: if a transaction which violates the database’s consistency rules is executed, the entire transaction will be rolled back and the database will be restored to a state consistent with those rules; however, if a transaction ends successfully, it will take the database from one state that is consistent with the rules to another state that is also consistent with the rules.
- **Isolation**: multiple transactions occurring at the same time not impact each others’ execution; the following degrees of isolation were originally described as degrees of consistency by Jim Gray [6]:
  - degree 0 - a transaction does not overwrite data updated by another user or process of other transactions;
  - degree 1 - degree 0 plus a transaction does not commit any writes until it completes all its writes (until the end of transaction);
  - degree 2 - degree 1 plus a transaction does not read data updated by another user or process of other transactions;
  - degree 3 - degree 2 plus other transactions do not read data updated by another user or process of a transaction before the transaction commits;
- **Durability**: ensures that any transaction committed to the database will not be lost.

Note that the isolation property does not ensure the execution order of the transactions, merely that they will not interfere with each other. Durability is ensured through database backups and transaction logs that facilitate the restoration of committed transactions even if any subsequent software or hardware fails.

To work with an optimum distributed deductive database system we have to be sure that the fragmentation process end successfully causing the success of the allocation process. We propose a security method based on authentication and key exchange for providing a safe vertical fragmentation.

2. State of Art

For working with deductive databases there must be used a declarative language for defining relations, rules and user queries. Such a language, also one of the most popular, is Datalog. Rules can be compared with the relational views [18, 15] since they specify derived relations that are not stored in the database but that can be formed from facts through inference mechanisms based on the specifications of the
rule. The main difference between these two concepts is that rules involve recursion and relational views do not.

Two of the main design activities in a distributed deductive database system are fragmentation and allocation of data and rules. The performance and the efficiency of a distributed deductive database system depend on the fragmentation process which allows parallel execution of a single query and increases the level of concurrency.

Concurrency control involves the synchronizations of accesses to the distributed database to maintain the integrity of the database. The most popular consistency control algorithms are locking-based. These algorithms place a lock, which depends on the lock compatibility rules, on some unit of storage whenever a transaction attempts to access it. All these algorithms follow the next theorem:

**Theorem 2.1.** No lock on behalf of a transaction should be set once a lock previously held by the transaction is released.

This process has two phases:
- growing phase implies obtaining locks;
- shrinking phase implies releasing the transactions.

Releasing a lock before ending a transaction may cause serious problems. Therefore, most of the concurrency control algorithms are strict in holding their locks until the transaction ends. A big disadvantage of the concurrency control algorithms based on locking may cause deadlocks whose detection and management in a distributed system is very difficult. However they are more performant and simpler than timestamp-based algorithms.

Fragmentation also improves the locality of access throw applications in such a system since application views are usually formed by multiple relations [14]. The performance increasing is mainly due to the decreasing of the transactions’ response time. The response time decreases because fragmentation reduces the irrelevant data that is transferred and accessed among different sites [8].

In the building process steps of a distributed deductive database systems, one of the first issue is the selection of a fragmentation approach [2, 13]:
1. horizontal fragmentation - a relation is subdivided into groups that have the same attributes as the original one; such fragments are expressed as a selection operation on the global relation;
2. vertical fragmentation - the attributes from a relation schema are subdivided into groups; such fragments are obtained through protecting the global relation over each group.

**Definition 2.1.** A fragment is the result of an expression in a relational algebra which takes global relation as operands.

All the horizontal or vertical fragments can be considered relations, too. Because of that we can apply one or more operations on them, obtaining different fragments.

Different approaches for both vertical and horizontal fragmentation in distributive deductive database systems have been proposed. In [8] the authors present four different methods for fragmentating data and rules which maximize locality of query evaluation and minimize communication cost and execution time during query processing. The four algorithms are: RCA for rule clustering, OVF for computing overlapping vertical fragmentation, DVF for generating disjoint vertical fragmentation, and CAA for allocating rules and corresponding fragments. The vertical fragmentation technique is based on the access frequency of queries in one of the fragmentation
algorithms. An important feature of this technique is due to the fact that the attributes clustered in a vertical fragment are not determined by using an attribute affinity matrix [11, 12] but using the rule to attribute dependency matrices.

Another vertical partitioning algorithm on relations using a graphical technique is described in [12]. In [9] is constructed a theory of fragmentation and is also studied the completeness and update problems of overlapping fragments. [10] treats the rule allocation problem in a distributed database system and proposes a rule partitioning method, whereas [20] develops a hybrid knowledge fragmentation approach.

3. Vertical Fragmentation. Securing Rules Transfer

To exemplify fragmentation process we need some definitions.

Definition 3.1. A rule \( r \) in a deductive database system has the form:
\[
p(X_1, \ldots, X_n) : - q_1(Y_1, \ldots, Y_m), \ldots, q_t(Z_1, \ldots, Z_s)
\]
where \( p \) is the head (predicate) of \( r \) and can be derived or mixed predicate, \( q_1 \ldots q_t \) form the body of \( r \) and can be derived, mixed or base predicates. The argument of a predicate is a variable or a constant.

A predicate \( p \) is mixed if there is a set of ground facts for \( p \), and \( q \) appears as the head predicate of some rules [1] and a base predicate corresponds to a relation in the database.

Definition 3.2. A rule which has an empty body and all \( X_i \) are constants is named a fact.

Definition 3.3. A query is a rule that does not have a head.

Definition 3.4. A rule \( r \) is recursively if at least one of the predicates in its body is the head of \( r \).

A predicate may have multiple definitions since multiple rules can have the same head predicate.

Definition 3.5. Let \( p \) be a predicate in a rule \( r \). Any argument of \( p \) which appears as "_" is an unnamed variable called anonymous variable.

Definition 3.6. Let \( p \) and \( q \) be two predicates. The predicate \( p \) directly depends on the predicate \( q \) if the latter appears in the body of \( p \).

Suppose we have the rule:
\[
\text{student(Name,University,Desg):-}
\text{studentbase(_,Name,University,Desg,Age),}
\text{Age} > 21.
\]
To horizontally fragmentate this rule we have:
\[
\text{student1(Name,University,Desg):-}
\text{studentbase1(_,Name,'Craiova',Desg,Age),}
\text{Age} > 21.
\]
\[
\text{student2(Name,University,Desg):-}
\text{studentbase2(_,Name,'Oxford',Desg,Age),}
\text{Age} > 21.
\]
So, for the above fragmentation we assumed that the only values for the University attribute are 'Craiova' and 'Oxford'. The initial relation can be obtained from:
\[
\text{student=student1 UN student2}
\]
To exemplify the vertical fragmentation we will use the same rule. Such a fragmentation can be done in two ways. First method obtains the derived relation from a rule fragmented exactly in the same way as a stored base relation:

\[
\text{student}_1\text{Name,University):- studentbase(\_Name,University,\_Age), Age>21.}
\]

\[
\text{student}_2\text{Name,Desg):- studentbase(\_Name,\_Desg,Age), Age>21.}
\]

The initial relation is obtained through:

\[
\text{student=student}_1 \text{ JOIN student}_2
\]

The other way for vertical fragmentation is obtained through distributing literals in the body of a rule. Suppose we have the rule:

\[
R : \neg P, Q, S.
\]

where P, Q and S are relations. The rule can be written as:

\[
R : \neg P_1, P_2, Q, S_1, S_2
\]  \hspace{1cm} (1)

where \( P \) is vertically fragmentated in \( P_1 \) and \( P_2 \) and \( S \) is vertically fragmentated in \( S_1 \) and \( S_2 \). So equation (1) is vertically fragmented in:

\[
R_1 : \neg P, Q, S_1.
\]

and

\[
R_2 : \neg P_2, S_2.
\]

For such a fragmentation to be possible we have to assume that \( P_1, Q \) and \( S_1 \) are defined at one site forming a useful unit of knowledge, while \( P_2 \) and \( S_2 \) are defined over another site also forming a useful of knowledge. We can parallelly execute \( R_1 \) and \( R_2 \). We can reconstruct \( R \) using:

\[
R : \neg R_1 \text{ JOIN } F_2
\]

Our method provides a safe successful ending for the vertical fragmentation process. Before the management system executes one fragmentation rule it first verifies its authenticity. Suppose a user wants to fragmentate the rule (1) in:

\[
R_1 : \neg P, Q, S_1.
\]

and

\[
R_2 : \neg P_2, S_2.
\]

Suppose we have an asymmetric cryptosystem where \((smpu, smpr)\) are the public and the private key for the management system and \((uspu, uspr)\) are the public and the private key for the user. \( En \) and \( De \) denote the encryption and the decryption operations. We suppose that the key pairs are already generated by a trusted part. First, the system sends:

\[
En_{smpr}(X)
\]

Then the user decrypts

\[
De_{smpu}(En_{smpr}(X))
\]

and sends back

\[
En_{uspr}(R_1 : \neg P, Q, S_1, X) \hspace{1cm} En_{uspr}(R_2 : \neg P_2, S_2, X).
\]
The management system computes
\[ D_{\text{uspr}}(En_{\text{uspr}}(R_1 : -P, Q, S_1, X)) \]
\[ D_{\text{uspr}}(En_{\text{uspr}}(R_2 : -P_2, S_2, X)) \]
So the system sends a random value encrypted with his own private key. The user receives it and decrypts it with the system’s public key. Then he encrypts the result along with each fragmentation rule with his own private key and sends them to the system. The system decrypts them with the user public key and verifies if the obtained value is the same one that it randomly chose at the beginning of the protocol. If the values are equal the user is authenticated because the rules and the value \( X \) are decrypted by the system with the public key of the user. Obtaining the same value proves that \( X \) was encrypted with the user’s private key which it is known only by its owner. Using such an authentication protocol, the randomly generated value \( X \) can also be used as a private key for a symmetric cryptosystem.

3.1. Secure the Data through Elliptic Curve Cryptosystem. To fix the keys the communicating parties can use two types of methods: key imposed transmission and key agreement. We will present an example of each method. For key agreement protocol the most used is the Elliptic Curve Diffie-Hellman. Using this protocol the two communicating parties (named \( S_1 \) and \( S_2 \)) agree on a symmetric key. The protocol is described in the algorithm below:

**INPUT:** domain parameters \((F, p, a_E, b_E, G, n, h)\) The two keys \( k_A \) and \( k_B \) are equal

**Algorithm 1 ECDH**

1. \( S_1 \) generates \( a \) and computes \( aG \)
2. \( S_1 \) sends \( aG \) to \( S_2 \)
3. \( S_2 \) generates \( b \) and computes \( bG \)
4. \( S_2 \) sends \( bG \) to \( S_1 \)
5. \( S_1 \) computes \( k_A = abG \)
6. \( S_2 \) computes \( k_B = baG \)

and we note the session key \( K = k_A = k_B \). The only public values are \( aG \) and \( bG \). If Eve (the attacker) intercepts these two values she cannot find \( K = abG \) because finding this key means resolving the ECDLP. This protocol is vulnerable to a man-in-the-middle attack because the information exchange is made in two rounds. In this attack Eve intercepts the messages sent by the two communicating parties and sends others using her own keys. So, Eve will establish a key with \( S_1 \) and one with \( S_2 \). Using these keys Eve can intercept and modify the messages between \( S_1 \) and \( S_2 \). The communicating parties will not even notice believing that the protocol has been successfully ended. We present such an attack in the next algorithm, where the key established with \( S_1 \) is \( acG \) and the one established with \( S_2 \) is \( bdG \):  

**INPUT:** domain parameters \((F, p, a_E, b_E, G, n, h)\)

To avoid this attack the communicating parties must use an authenticated Diffie-Hellman protocol. This means that \( S_1 \) will send along with \( aG \) another information which will prove her identity to \( S_2 \). This information may be a zero knowledge one, an encrypted message known only by \( S_2 \) or a digital signature. The most recommended is using a digital signature scheme. If such a scheme is used \( S_1 \) will send \((aG, (r, s))\) to \( S_2 \), where \((r, s)\) is the digital signature applied to the message \( aG \).

The most used digital signature scheme based on elliptic curves is the ECDSA. Like all the digital signature schemes, the ECDSA has three algorithms: key generation,
Algorithm 2 Man-in-the-Middle Attack for ECDH
1: \( S_1 \) generates \( a \) and computes \( aG \)
2: \( S_1 \) sends \( aG \) to \( S_2 \)
3: Eve intercepts \( aG \), generates \( c \) and computes \( cG \)
4: Eve sends \( cG \) to \( S_1 \)
5: \( S_1 \) computes \( acG \)
6: Eve computes \( caG \)
7: Eve generates \( d \) and computes \( dG \)
8: Eve sends \( dG \) to \( S_2 \)
9: \( S_2 \) generates \( b \) and computes \( bG \)
10: \( S_2 \) computes \( bdG \)
11: \( S_2 \) sends \( bG \) to \( S_1 \) but the message is intercepted by Eve
12: Eve computes \( dbG \)

signature generation, signature verification. The input for ECDSA are the domain parameters defined above. The advantages and disadvantages of the ECDSA can

Algorithm 3 ECDSA Key Generation
1: \( S_1 \) generates \( a \) such that \( a \in [2, n - 2] \)
2: \( S_1 \) computes \( Q = aG \) and sends it to \( S_2 \)
3: The public key is \( Q \) and the private one is \( a \)

Algorithm 4 ECDSA Signature Generation
1: \( S_1 \) generates \( k \in \{1, \ldots, p - 1\} \)
2: \( kG \leftarrow T(x_T, y_T) \)
3: \( r \leftarrow x_T \mod n \)
4: if \( r = 0 \) then
5: goto step 2
6: end if
7: \( s \leftarrow k^{-1}(SHA(m) + ar) \)
8: if \( s = 0 \) then
9: goto step 2
10: end if
11: the signature for the message \( m \) is \( (r, s) \)

Algorithm 5 ECDSA Signature Verification
1: \( S_2 \) receives \( (r, s) \)
2: \( c \leftarrow s^{-1} \mod n \)
3: \( u_1 \leftarrow SHA(m)c \mod n \)
4: \( u_2 \leftarrow rc \mod n \)
5: \( u_1G + u_2Q \leftarrow (x_0, y_0) \)
6: \( v \leftarrow x_0 \mod n \)
7: if \( v = r \) then
8: valid signature
9: end if
be read in [21]. The reader can study a comparison between ECDSA and the classic method, DSA, in [19].

4. Conclusions

Distributed deductive database systems have become a reality in the past decade. They are mostly used in industry, banking and administration. A great interest has appeared in applying logic to databases, particularly in deductive database systems, which not only manage large facts stored in relations in a database and rules in a rulebase but also provide for deduction from given database and rulebase [3, 5]. To provide functionality to a distributed deductive database system the fragmentation process must be efficient and secure. An insecure fragmentation may lead to unmotivated increasing rulebase resulting a system failure.

We propose a simple authentication method based on proving the knowing of a value (throw classic asymmetric encryption and Elliptic Curve asymmetric encryption). The protocol is efficient since there is a small number of operations to compute. The key pairs are already generated and considered valid.

Acknowledgment

One of the authors work was supported by CNCSIS PCCE 08/2010 project.

References


(Dorel Săvulea, Nicolae Constantinescu) Department of Computer Science, University of Craiova, 13 A.I. Cuza Street, Craiova, 200585, Romania

E-mail address: savulea@central.ucv.ro, nikyc@central.ucv.ro