Shamir's Secret Sharing Algorithm Implemented in SQL

Dr. Constantin Florin Sirbu
Al. I. Cuza University of Iaşi
Department of Business Information Systems

ABSTRACT
In this paper, we propose a solution for the problem of secret sharing along different databases that may be used in hybrid cloud data security. In a hybrid cloud data may be spitted over more clouds and may be stored by service providers, so one important issue is the security of sensitive data. Secret sharing is not a new challenge in the IT world; there are many old and good algorithms that may be used. The contribution of this paper is to implement such an algorithm (Shamir's secret sharing algorithm) in SQL, 100% declarative and to describe the architecture that ensures secret sharing and secret recovery in a transparent manner. The challenge is to solve problems from finite field arithmetic with operators from relational algebra. By using this architecture the developers would concentrate only at the business design, being ensured that secrets spread over clouds remain secrets even if the service providers will not play fair.

Keywords
Shamir's secret sharing; invers mod function; distributed secure database service; SQL.

1. INTRODUCTION
Hybrid cloud is a composition of two or more clouds (private, community or public clouds) not necessary of different types. This model became available as a result of standardization ensuring data and application portability [4]. Hybrid cloud solutions offer bigger benefits as the traditional cloud solutions in terms of disclosure control and minimizing cloud resources [8] given that most organizations already have an infrastructure they can use.

There are three challenges that drive the design of Relational Cloud [3]: efficient multi-tenancy to minimize the hardware footprint required for a given (or predicted) workload, elastic scale-out to handle growing workloads, and database privacy. IDC’s findings show that security concerns are the number one issue facing cloud computing [10].

In the database-service provider model there are two main privacy issues [5]. First the owner of the data needs to be assured that the data is protected from against outside attackers. Second, some data has to be protected even from the service providers, if the providers themselves cannot be trusted or there are legal constraints concerning this. By using the Shamir's secret sharing algorithm, both security issues are solved in hybrid clouds that use more service providers because: the secret may be recovered also by using some of the secret parts, not all of them; each service provider saves only a part of the secret.

2. SHAMIR’S SECRET SHARING
The general architecture of a distributed secure database service consists of a trusted client as well as two or more servers that provide database services [1]. These providers ensure reliable content storage, scaling, performance tuning, and data backup but are not trusted to preserve content privacy.

Shamir's secret sharing is an algorithm [9] written in finite field arithmetic used to divide a secret into parts, giving each participant (each database-service provider in our case) its own unique part, where some of the parts or all of them are needed in order to reconstruct the secret. We demonstrate in this article that Shamir’s secret sharing algorithm may be used to ensure privacy of the data stored in relational clouds.

We will consider:
- Z a big prime number used to construct the ring of integers modulo Z; all the next operations will be performed in this ring, that means all the additions, subtractions and multiplications will start with the usual operation on integers, followed by reduction modulo Z; all the next numbers will be smaller than Z.
- S the secret
- n the number of participants
- k the number of parts that are enough to reconstruct the secret
- I(x) a polynomial function of degree k-1 in which the coefficients are randomly chose, except the free one that is equal with the secret S.

According to Shamir's algorithm the parts of the secret will be I(i) for participant i.

The secret recovery starts from \{li | i \in A\} with |A|=k and using the Lagrange's interpolation [2]:

\[ S = \sum_{i \in A} (I_i \cdot \prod_{j \in A \setminus \{i\}} \frac{x_j}{x_j - x_i}). \]

For the values of Xi Shamir proposes Xi = i, and so, the secret S is computed with the next formula [2]:

\[ S = \sum_{i \in A} (I_i \cdot \prod_{j \in A \setminus \{i\}} \frac{j}{j - i}). \]
3. THE SQL IMPLEMENTATION
The Shamir’s algorithm may be easily found implemented in a procedural way. The contribution of this paper is to demonstrate that even such an algorithm in finite field arithmetic may be implemented with the operators from relational algebra. We will use for this the SQL dialect from Oracle 11g.

The parameters of the algorithms will be created as public variables of the package Pck. These variables will be read using specific get functions. For the recovered parts of the secret we create the next data types:

Create or replace type rec as Object (i integer, s integer);
Create or replace type array as table of rec;

For the recovered parts of the secret we create the next values will be used to test the algorithm implementation.

```
Z integer := 17;
n integer := 5;
k integer := 3;
S integer := 15;
```

For the polynomial function \( P(x)=3X^2+5X+S \) the secret parts are displayed in Table 1.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Secret part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>

To recover the secret will be enough the parts of any three participants. For example, we may choose for tests just the parts from the first, the third and the fifth participant:

```
recoveredSi array := array(rec(1,6),
                          rec(3,6),
                          rec(5,13));
```

The formula used to recover the secret would be in this case:

\[
S = 6 \cdot \frac{3}{3-1} \cdot \frac{5}{5-1} + 6 \cdot \frac{1}{1-3} \cdot \frac{5}{5-3} + 13 \cdot \frac{1}{1-5} \cdot \frac{3}{3-5}
\]

Since all these operations (addition, subtraction, multiplication) are performed in the ring of integers modulo 17, at the beginning of the formula:

\[
\frac{1}{3-1} = \frac{1}{2} = 2^{-1} = 9 \\
\text{because MOD}(2^9, 17) = 1
\]

3.1 The inverse of a modulo in SQL
Since all the operations are in a ring of integers, we will have to use the built in MOD function. For the modular multiplicative inverse of a modulo there are no built in functions in Oracle or in any major DBMS, so we will have to create a function or a query that returns it. We will create both, a query in Oracle SQL and a function in Oracle PL/SQL. In the existing theory there are two solutions to compute the inverse of a modulo: the extended Euclidean algorithm and the Euler’s theorem. We will use none of them (because both of them are procedural and not declarative), and we will propose a new way of computing the inverse of a modulo by using the relational algebra from SQL.

The algorithm should return for each number \( X_i \) from 1 to \( Z-1 \) the number \( Y_i \) that multiplied with \( X_i \) returns 1 in the ring of integers modulo \( Z \):

\[
\text{MOD}(X_i \cdot Y_i, Z) = 1.
\]

To start the algorithm we need a relation with \( Z-1 \) tuples (that will be used as \( X_i \)):

```
SELECT LEVEL as n
FROM DUAL
CONNECT BY LEVEL <= Pck.Get_Z-1
```
and a relation with a lot of numbers that are 1 in the ring of integers modulo Z (that will be used as $X_i \cdot Y_i$).

$$\text{SELECT LEVEL as } n \text{ FROM DUAL WHERE MOD(LEVEL, Pck.Get_Z)=1}$$

CONNECT BY LEVEL <= Pck.Get_Z*Pck.Get_Z

When we make a Cartesian product from the above two relations we have for each $X_i$ a number that is 1 in modulo Z, but it has to be also a multiple of $X_i$:

$$\text{MOD}(X_i \cdot Y_i, X_i) = 0$$

So, the invers of a modulo may be computed in SQL with:

$$\text{Select DISTINCT a.n, MOD(b.n/a.n, Pck.Get_Z) as InversN FROM (SELECT LEVEL as n FROM DUAL CONNECT BY LEVEL < Pck.Get_Z-1) a,}$$

$$(\text{SELECT LEVEL as n FROM DUAL WHERE MOD(LEVEL, Pck.Get_Z)=1 CONNECT BY LEVEL <= Pck.Get_Z*Pck.Get_Z)} b$$

Where MOD(b.n, a.n)=0

With the above query we may create a view (named here VW_INVERSMOD) and/or a function (INVERSMOD), to compute the invers of a modulo for a specific integer:

$$\text{CREATE OR REPLACE Function InversMod (N in integer, Prime in integer) Return Integer as i integer;}$$

$$\text{Begin Select DISTINCT MOD(b.n/a.n,Prime) Into i FROM (SELECT LEVEL-1 as n FROM DUAL CONNECT BY LEVEL <= Prime) a,}$$

$$(\text{SELECT LEVEL as n FROM DUAL WHERE MOD(LEVEL,Prime)=1 CONNECT BY LEVEL <= Prime*Prime)} b$$

Where MOD(b.n, a.n)=0 And a.n = InversMod.N; Return i; End;}$$

3.2 Splitting the secret

The secret has to be split in $k$ parts, so the result of this operation should be a relation of $k$ tuples. We already know from the previous step that the clause CONNECT BY may help us in Oracle to receive this relation. For each tuple we will compute also the result of the polynomial function, in the ring of integers modulo Z:

$$\text{SELECT LEVEL as i, MOD(3*POWER(LEVEL,2) + 5 * LEVEL + Pck.Get_S,Pck.Get_Z) as Si FROM DUAL CONNECT BY LEVEL <= Pck.Get_n;}$$

This query returns the values from table 1.

The same result would be received also with the next PL/SQL function.

$$\text{CREATE OR REPLACE Function SplitSecret (Z in integer, S in integer, n in integer) Return array PIPELINED as i integer; Si integer;}$$

Begin
For i in 1..n Loop
    Si := MOD(3*POWER(i,2) + 5 * i + S,Z);
    Pipe row (REC(i, Si));
End Loop;
End;

Because the function SplitSecret is a pipelined function, its result may be seen as a relation:

Select * From TABLE(SplitSecret( Pck.Get_Z, Pck.Get_S, Pck.Get_n))

3.3. Recover the secret

The procedural version of the algorithm of recovering the secret would be in PL/SQL:

CREATE OR REPLACE Function GetSecret
(Z in integer, Si in array)
Return integer as S integer; c integer;
Begin
    S:= 0;
    For Rec1 in (Select * From TABLE(Si)) loop
        c := 1;
        For Rec2 in (Select * From TABLE(Si)) loop
            If Rec2.i <> Rec1.i Then
                c := c * Rec2.i * INVERSMOD(ABS(Rec2.i - Rec1.i),Z) * SIGN(Rec2.i - Rec1.i);
            End if;
        End loop;
        S:= S + Rec1.S * c;
    End loop;
    Return MOD(S,Z);
End;

To use this function and recover the secret we write:

Select GetSecret(Pck.Get_Z, Pck.Get_recoveredSi) as S
From Dual

The secret recovering algorithm may be written also entirely with the declarative language SQL:

Select MOD(SUM(x),Pck.Get_Z) as S From
    ( Select i, Exp(Sum(ln(ABS(c)))) * Decode(Mod(Count(Decode(Sign(c),-1,1,null)),2),1,-1,1)) * c
    From TABLE(Pck.Get_recoveredSi) rec1,
    TABLE(Pck.Get_recoveredSi) rec2,
    VW_INVERSMOD v
    Where Rec1.i <> Rec2.i
    And v.n = ABS(Rec2.i - Rec1.i)
    Union
    Select rec1.i, rec.S
    From TABLE(Pck.Get_recoveredSi) rec
    )
    Group By i

With this SELECT statement we create a view named VW_SECRET.

In order to understand this declarative solution we should read first the sub-subquery: the relation of the recovered parts of the secret (rec1) makes a Cartesian product with itself (rec2) in order to compute the coefficients that are multiplied with each part. To compute these coefficients we need also the invers modulo of rec2.i – rec1.i, which in the original formula of Shamir was expressed as j-i. That is why we make a join with the view VW_INVERSMOD. Because the VW_INVERSMOD contains only positive integers, this join will be made using the absolute values of rec2.i – rec1.i and the coefficients will receive also SIGN(Rec2.i - Rec1.i).

To the above created relation we add with the reunion operator the secret parts in order to create a product for each participant. “For each participant” is translated into SQL with Group By i. The challenge is that there is no product aggregate function in SQL. For positive
integers we may compute products over aggregates with $\text{Exp}(\text{Sum}(\text{ln}(N)))$, so we use this functions for $\text{ABS}(c)$ and then we count the negative integers to establish the sign of the product.

At the end of the algorithm is made a sum with all the products computed for each participant, sum which is also in the ring of integers modulo Z.

4. THE ADVANTAGE OF THE PROPOSED SOLUTION

In a distributed database environment data may be saved also by service providers. The rise of cloud computing and especially of the hybrid clouds triggers the need to use the local computing capability to hide commercials secrets from the service providers. Not all the data saved in clouds has to be encrypted or split. But, if we identify such sensitive data is important to hide it well and without "reinventing the wheel".

We propose an architecture in which the data owner uses a view that splits and recovers the secrets. The user is able to see the secret with this view, but the secret is stored only in its parts.

In Oracle it is possible to declare INSTEAD OF triggers on views. These triggers capture the DML operations made over the views and replace them with their own action. For example, the above function $\text{Pck.Get}\_\text{recoveredSi}$ may read the secret parts from their storing location and not from a package variable and the view $\text{VW}\_\text{SECRET}$ may have an INSTEAD OF trigger to split the secret and save these parts in different databases/clouds. An example of such trigger.

CREATE OR REPLACE TRIGGER TRG_SPLITSECRET
INSTEAD OF UPDATE ON VW_SECRET
FOR EACH ROW
BEGIN
    For Rec in (SELECT LEVEL as i, MOD(3*POWER(LEVEL,2) + 5 * LEVEL + new.S_Pck.Get_Z) as Si
    FROM DUAL
    CONNECT BY LEVEL <= Pck.Get_n) Loop
        EXECUTE IMMEDIATE '"Update TableX@cloud'||TRIM(Rec.i)||" SET Si = "||Rec.Si||" WHERE PK = ...';
    End loop;
END;

The trigger TRG_SPLITSECRET uses one database link (cloud1, cloud2, ...) for each participant and native dynamic SQL to write just once the structure of the UPDATE operations. This UPDATE structure should be completed at least with the primary key of the involved tables, according to the existing business rules.

5. CONCLUSIONS

We demonstrate in this article that complex algorithms from other mathematics then relational algebra may be solved in a declarative manner in SQL. Once translated into SQL, these algorithms may easily be implemented in the database environment.

Our contribution includes also a new algorithm to compute the modular inverse, which does not use the extended Euclidean algorithm or the Euler's theorem. In the spirit of this paper this new algorithm for computing the modular inverse is written entirely in SQL.

By using the SQL solutions from this paper the Shamir's secret sharing algorithm may be inserted into views and INSTEAD OF triggers improving the data privacy over hybrid clouds.

The algorithms presented in this paper may become a part of a framework that ensures the clients of database-as-a-service that their data is protected from untrusted service providers.

Further researches relate to other parts of such framework (like the dynamic multisecret sharing scheme [6]) considering that a system that uses more algorithms from cryptography is more secure then a system that uses only one algorithm for data privacy.

Also further research may be done in the implementation of traditional procedural algorithms into nonprocedural languages, not only from cryptography, but also from other fields that suppose complex algorithms (like the quantitative research applied in the fund investment management).

6. ACKNOWLEDGMENTS

This work was cofinanced from the European Social Fund through Sectoral Operational Programme Human Resources Development 2007-2013, project number POSDRU89/1.5/S/56287 „Postdoctoral programs at the forefront of the excellence research in the technologies of the informational society and innovative product and process development”, partner University of Oradea.

7. REFERENCES


