fVSS: A New Secure and Cost-Efficient Scheme for Cloud Data Warehouses

Varunya Attasena  
Université de Lyon (ERIC)  
5 av. Pierre Mendès-France  
69676 Bron Cedex – France  
vattasena@eric.univ-lyon2.fr

Nouria Harbi  
Université de Lyon (ERIC)  
69676 Bron Cedex – France  
nouria.harbi@univ-lyon2.fr

Jérôme Darmont  
Université de Lyon (ERIC)  
5 av. Pierre Mendès-France  
69676 Bron Cedex – France  
jerome.darmont@univ-lyon2.fr

ABSTRACT

Cloud business intelligence is an increasingly popular choice to deliver decision support capabilities via elastic, pay-per-use resources. However, data security issues are one of the top concerns when dealing with sensitive data. In this paper, we propose a novel approach for securing cloud data warehouses by flexible verifiable secret sharing, fVSS. Secret sharing encrypts and distributes data over several cloud service providers, thus enforcing data privacy and availability. fVSS addresses four shortcomings in existing secret sharing-based approaches. First, it allows refreshing the data warehouse when some service providers fail. Second, it allows on-line analysis processing. Third, it enforces data integrity with the help of both inner and outer signatures. Fourth, it helps users control the cost of cloud warehousing by balancing the load among service providers with respect to their pricing policies. To illustrate fVSS’ efficiency, we thoroughly compare it with existing secret sharing-based approaches with respect to security features, querying power and data storage and computing costs.

Categories and Subject Descriptors

H. Information Systems [H.2. Data Management]: H.2.7. Database administration—Data Warehouse and Repository; Security, Integrity and Protection

Keywords

Data warehouses; OLAP; Cloud computing; Secret sharing; Data privacy; Data availability; Data integrity

1. INTRODUCTION

Cloud business intelligence (BI) is becoming increasingly popular, providing the benefits of both classical BI (efficient decision-support) and cloud computing (elasticity of resources and costs). However, one of the top concerns of cloud users and would-be users remains security. Some security issues are inherited from classical distributed architectures, e.g., authentication, network attacks and vulnerability exploitation, but some directly relate to the new framework of the cloud, e.g., cloud service provider (CSP) or subcontractor espionage, cost-effective defense of availability and uncontrolled mashups [5]. In this paper, we focus on data security, which is of critical importance in a BI context where sensitive data are processed. Data security is usually managed by CSPs, but with the multiplication of CSPs and subcontractors in many different countries, intricate legal issues arise and trust in CSPs may be jeopardized.

In contrast, end-to-end security gives the control of both data security levels and costs back to users, through classical techniques such as data encryption, anonymization, replication and verification. However, updating and querying secured data in the cloud may turn inefficient and expensive. Thus, specific end-to-end security approaches were designed for distributed/cloud databases (DBs) and data warehouses (DWs) [15, 18]. The MONOMI system [20] notably encrypts both SQL queries and data with multiple cryptographic schemes. However, although MONOMI enforces data privacy, it addresses neither data availability nor integrity issues. In contrast, secret sharing [16] simultaneously enforces data privacy, availability and integrity. Secret sharing transforms sensitive data into individually meaningless data pieces (called shares) that are distributed to n CSPs. Computations can then be performed onto shares, but yield meaningless individual results. The global result can only be reconstructed (decrypted) by the user.

All secret sharing-based approaches allow accessing shares from t ≤ n CSPs, i.e., shares are still available when up to n − t CSPs fail, e.g., go bankruptcy, due to a technical problem or even by malice. However, no approach allows updating shares when even one single CSP fails, thus hindering cloud DWs’ refreshment capabilities. Moreover, although these approaches feature all basic DB querying operators, only one handles on-line analysis processing (OLAP) and it does not support lazy updates on cloud-stored cubes. In addition, few approaches actually enforce data integrity through verification of inner code (to verify whether CSPs are malicious) or outer code (to detect incorrect data before decryption), and only one features both inner and outer signatures for this sake. Finally, although some approaches bring in solutions to reduce overall storage volume so that it falls well under n times that of original data, and thus de-
crease monetary cost in the pay-as-you-go paradigm, there is still room for improvement.

To address all these issues, we propose a novel approach that relies on a flexible verifiable secret sharing scheme (IVSS). To the best of our knowledge, IVSS is the first approach allowing DW refreshment when one or several CSPs fail. Moreover, IVSS allows running OLAP operators on shared DWs or cubes without reconstructing all data first. IVSS also features both inner and outer signatures for data verification. Finally, IVSS allows users adjusting the volume of shared data at each CSP, which helps optimize cost with respect to various CFP pricing policies.

The remainder of this paper is organized as follows. Section 2 discusses previous research related to IVSS. Section 3 details our secret sharing and reconstruction mechanisms, the new outer signature we propose and how our approach applies to data warehouses and OLAP. Section 4 provides a comparative study of IVSS with state-of-the-art existing approaches. Finally, Section 5 concludes this paper and hints at future research perspectives.

2. RELATED WORKS

Among encryption techniques, only secret sharing [3] handles both data privacy and availability, which is why we focus on this family of approaches. The principle of secret sharing [16] is based on the fact that \( t \) points define a polynomial \( y = f(x) \) of degree \( t - 1 \). The secret is the polynomial’s constant term and the remaining terms are usually randomly selected. Each data piece is transformed into \( n \) shares \( f(x_i) \) corresponding to points of the polynomial. Reconstruction of the secret is achieved through Lagrange interpolation [6]: there is only one polynomial \( p(x) \) such that degree \( p(x) < t \) and \( p(x_i) = f(x_i) \). Then the secret is \( p(0) \). Moreover, modern secret sharing schemes, such as multi-secret sharing [2, 14, 22], verifiable secret sharing [17], and verifiable multi-secret sharing [4, 10], also help reduce shared data volume, verify the honesty of CSPs, and both, respectively. We classify secret sharing-based approaches for securing DBs and DWs into two families.

In the first family of approaches [1, 2, 8, 9, 11], each table is encrypted into \( n \) shared tables, each of which is stored at one given CSP (Figure 1). Recall that only \( t \) of \( n \) shared tables are sufficient to reconstruct the original table. Most of these approaches assume that CSPs are not malicious and that connections between CSPs and users are secure. Only one [2] includes a data verification process that exploits hash-generated signatures: an inner signature (incorporated to the shares) to verify whether CSPs are malicious, and an outer signature that helps detect incorrect or erroneous (lost, damaged, alternative...) data before decryption and prevents useless data transfers. Both signatures are stored at CSPs.

In the second family of approaches [12, 13, 19, 21], one or more additional index servers, located at \( \{CSP_1\}_{n=1} \), store B++ tree indices and signatures (Figure 2). The index servers require higher security and computing power than that of other nodes, and a secure connection to the user’s. The index servers support data verification by various means, i.e., homomorphic encryption [13], a hash function [19] and checksums and a hash function [21]. However, data verification cannot take place if the index server fails.

Regarding accessing shares, classical secret sharing [16] and most of its above-cited extensions natively support some exact match and aggregation operators, i.e., equality and inequality, sum, average and count. Each approach handles operators needing sorted data, e.g., range operators, maximum and minimum, with various techniques: preaggregation before sharing [8], a B++ tree index [12, 13] or the rank coefficient used in classical secret sharing [1, 9, 11]. However, extra storage space is needed to store these data structures. Finally, almost all approaches allow updates on shares, since each piece of data is encrypted independently.

The features of all above-cited approaches are summarized in Table 2. We compare them with IVSS in Section 4.

3. FLEXIBLE VERIFIABLE SECRET SHARING

3.1 fVSS Principle

fVSS is a \((t, n)\) flexible verifiable secret sharing scheme belonging to the second family of approaches identified in Section 2. As all similar approaches, fVSS shares data over \( n \) CSPs, \( t \) of which are necessary to reconstruct original data. Table 1 lists IVSS’ parameters, which will be introduced throughout this section.

The main novelty in fVSS is that, to optimize shared data volume and thus cost, we share a piece of data fewer than \( n \) times. For example, in Figure 3 where \( n = 5 \), record \#124 of table PRODUCT is only shared at CSP\(_1\), CSP\(_3\) and CSP\(_5\), which presumably feature the lowest storage costs. To achieve this, we proceed as follows.

Suppose we want to share a piece of data \( a_{jkl} \), e.g., the category ID of product \#124 in Figure 3(a). We also need to share its inner signature \( s_{xjkl} \) to enforce data integrity. To generate a polynomial of degree \( t \), we need \( t - 2 \) more values that we call pseudo shares. Usually, secret sharing...
Table 1: IVSS parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>A big prime number</td>
</tr>
<tr>
<td>( n )</td>
<td>Number of CSPs</td>
</tr>
<tr>
<td>( t )</td>
<td>Number of shares necessary for reconstructing original data</td>
</tr>
<tr>
<td>( CSP_i )</td>
<td>CSP number ( i )</td>
</tr>
<tr>
<td>( ID_i )</td>
<td>Identifier number of CSP ( i ) such that ( p &gt; ID_i &gt; 1 )</td>
</tr>
<tr>
<td>( m )</td>
<td>Number of tables</td>
</tr>
<tr>
<td>( T_j )</td>
<td>Table number ( j ) such that ( T_j = (R_{k1})<em>{j1} \ldots (R</em>{kt})_{jt} )</td>
</tr>
<tr>
<td>( ET_{ij} )</td>
<td>Shared table of ( T_j ) stored at CSP ( i )</td>
</tr>
<tr>
<td>( \eta_i )</td>
<td>Number of attributes of ( T_j ) and ( ET_{ij} ) (not including primary key)</td>
</tr>
<tr>
<td>( r_{ij} )</td>
<td>Number of records of ( T_j )</td>
</tr>
<tr>
<td>( e_{ij} )</td>
<td>Number of records of ( ET_{ij} ) such that ( r_{ij} \geq e_{ij} ) and ( \sum_{j=1}^{t} e_{ij} = (n - t + 2) r_{ij} )</td>
</tr>
<tr>
<td>( A_j )</td>
<td>Attribute number ( j ) of ( T_j ) and ( ET_{ij} )</td>
</tr>
<tr>
<td>( R_{kj} )</td>
<td>Record ( k ) of such that ( R_{kj} = { pk_k, ek_k \ldots dk_k } )</td>
</tr>
<tr>
<td>( ER_{ij} )</td>
<td>Record ( k ) of ( ET_{ij} ) such that ( ER_{ij} = { pk_k, ek_k \ldots dk_k } )</td>
</tr>
<tr>
<td>( p_k )</td>
<td>Primary key value of ( R_{kj} )</td>
</tr>
<tr>
<td>( pk_{kij} )</td>
<td>Primary key value of ( ER_{ij} ). It is not encrypted.</td>
</tr>
<tr>
<td>( d_{kij} )</td>
<td>Value of ( A_j ) of ( R_{kj} ) such that ( p &gt; d_{kij} \geq 0 ).</td>
</tr>
<tr>
<td>( SV_{jk} )</td>
<td>Group of CSPs that store shares of ( R_{kj} ) such that ( S_{Gk} \subset { CSP}_{1\ldots n} ) and (</td>
</tr>
<tr>
<td>( UG_{jk} )</td>
<td>Group of CSPs that do not store shares of ( R_{kj} ) such that ( UG_{jk} \subset { CSP}<em>{1\ldots n} ) and ( UG</em>{jk} \subset { CSP}_{1\ldots n} ).</td>
</tr>
<tr>
<td>( RG )</td>
<td>Group of CSPs selected to reconstruct data such that ( RG \subset { CSP}_{1\ldots n} ) and (</td>
</tr>
<tr>
<td>( s_{\Sigma n_{kij}} )</td>
<td>Inner signature of ( d_{kij} ) such that ( p &gt; s_{\Sigma n_{kij}} \geq 0 )</td>
</tr>
<tr>
<td>( s_{\Sigma out_{n_{kij}}} )</td>
<td>Outer record signature number ( v ), level ( u ) of ( ER_{ij} ) in ( ET_{ij} ) stored at CSP ( i )</td>
</tr>
<tr>
<td>( w_i )</td>
<td>Maximum number of child nodes in CSP ( i )'s outer signature tree</td>
</tr>
</tbody>
</table>

Figure 3: Sample original and shared data

(a) Original data  (b) Indices on index server
(c) Shares at CSP \( 1 \)  (d) Shares at CSP \( 2 \)
(e) Shares at CSP \( 3 \)  (f) Shares at CSP \( 4 \)
(g) Shares at CSP \( 5 \)

Figure 4: Sample sharing process

(Section 3.4), achieve loading, backup and recovery processes (Section 3.5) and perform OLAP operations (Section 3.6).

3.2 Data Sharing and Reconstruction

In IVSS, DB attribute values, except NULL values and primary or foreign keys, are encrypted and shared in relational DBs at CSPs'. Keys help match records in the data reconstruction process and perform join and grouping operations. Any sensitive primary key, such as a social security number, is replaced by an unencrypted sequential integer key.

Each value \( d_{kij} \) of attribute \( A_j \) in record \( R_{kj} \) from Table \( T_j \) is encrypted into \( n - t + 2 \) shares generated from a polynomial \( f_{kij}(x) \) of degree \( t - 1 \) created by Lagrange interpolation from \( d_{kij} \), its inner signature \( s_{\Sigma n_{kij}} \), the identifier numbers \( ID_i \) of the CSPs selected to store the shares (this set of CSPs is denoted \( S_{Gk} \)), and two generated keys \( K_d \) and \( K_e \) for data and signatures, respectively. The inner signature we use is very similar to that of [2]. It is created from \( d_{kij} \) by homomorphic encryption [7]. \( s_{\Sigma n_{kij}} \) matches with \( d_{kij} \) in the reconstruction process only if CSPs in \( RG \) return correct shares.

3.2.1 Initialization Phase

1. Set values of \( p \) (a big prime number), \( n \) and \( t \).
2. Define one-variable hash function \( HF_1(a) \) where \( a \) is an integer and hash values \( HF_1(a) \) must be small integers.
3. Define one-variable homomorphic function \( HE_1(h) \) such that \( HE_1(h) \) and \( h \) are reals and \( HE_1(h_1) \pm HE_1(h_2) = HE_1(h_1 \pm h_2) \).
4. Define two-variable homomorphic function \( HE_2(a, b) \), where \( HE_2(a, b) \) and \( a \) and \( b \) are reals and \( HE_2(a_1, b_1) = HE_2(a_2, b_2) = HE_2(a_1 + a_2, b_1 + b_2) \).
5. Set values of CSP identifiers \( ID_{i=1\ldots n} \), \( K_d \) and \( K_e \) such that their values range in \( [0, p[ \). All \( HF_1(ID_i) \) must be unique and different from \( HF_1(K_d) \) and \( HF_1(K_e) \).

3.2.2 Data Sharing Process

Any record \( R_{kj} \) is encrypted independently as follows.

1. Determine the group of CSPs \( S_{Gk} \) that will store \( R_{kj} \)'s \( n - t + 2 \) shares. Let \( UG_{jk} \) be the group of CSPs that do not store \( R_{kj} \)'s shares, i.e., \( UG_{jk} = \{ CSP_i \}_{1\ldots n} - S_{Gk} \).
2. For each attribute $A_{ij}$:
   (a) Compute $d_{ijk}$'s inner signature:
   $$s_{\text{in}ijk} = HE_1(d_{ijk}).$$
   (b) Create polynomial $f_{jkl}(x)$ of degree $t - 1$ by Lagrange interpolation (Equation 1):
   $$f_{jkl}(x) = \sum_{a=1}^{t} \prod_{1 \leq \beta \leq t, \alpha \neq \beta} \frac{x - x_{\alpha}}{x_{\beta} - x_{\alpha}} \times y_{\alpha}$$
   where $\{(x_1, y_1), \ldots, (x_t, y_t)\} = \{(HF_{1}(K_d), d_{jkl}), (HF_{1}(K_s), s_{\text{in}ijk})_{\text{CSP}} \in SG_{jk}\} \cup \{(HF_{1}(ID_i), HE_2(pk_{jkl}, ID_i))_{\text{CSP}} \in UG_{jk}\}$.
   (c) Compute the set of $d_{ijk}$'s $n + t + 2$ shares $\{e_{ijgl}\}$.
   $$\forall \text{CSP}_i \in SG_{jk}: e_{ijgl} = f_{jkl}(HF_{1}(ID_i)),$$
   with $pk_{jkl} = pk_{ijkl}$.

Following this routine, record $R_{jk}$ is shared into $n + t + 2$ records $ER_{ijkl}$ at CSPs in $SG_{jk}$. The relationship between $R_{jk}$ and $ER_{ijkl}$ is maintained through primary keys $pk_{jkl} = pk_{ijkl}$. Finally, the bitmap corresponding to $R_{jk}$ is stored in the index server(s) at this time, knowing $SG_{jk}$ and $UG_{jk}$.

Finally, since each data piece is shared independently, it is easy to handle the usual data types featured in DBs. Integers, dates and timestamps can be directly shared by fVSS. For each attribute $A_{ij}$, dates and timestamps can be directly shared by fVSS.

### 3.2.3 Data Reconstruction Process

Any attribute value $d_{ijk}$ is reconstructed as follows.

1. Select $t$ CSPs to form reconstruction group $RG$.

2. For each $\text{CSP}_i \in RG$, if outer data verification (Section 3.3) outputs an error, replace $\text{CSP}_i$ by another CSP selected from $\{\text{CSP}_i\}_{i=1..n} - RG$.

3. For each $\text{CSP}_i \in SG_{jk} \cap RG$, load share $e_{ijgl}$ into $y_i$, where $pk_{ijkl} = pk_{ijkl}$.

4. For each $\text{CSP}_i \in UG_{jk} \cap RG$, compute pseudo share $y_i = HE_2(pk_{jkl}, ID_i)$.

5. Create polynomial $f_{jkl}(x)$ of degree $t - 1$ (Equation 1) with $x_i = HF_{1}(ID_i)$.

6. Compute value $d_{jkl} = f_{jkl}(HF_{1}(K_d))$.

7. Compute inner signature $s_{\text{in}ijk} = f_{jkl}(HF_{1}(K_s))$.

8. Verify $d_{jkl}$'s correctness: if $s_{\text{in}ijk} \neq HE_1(d_{jkl})$, then restart reconstruction process at step #1 with a new $RG$.

### 3.2.4 Recapitulative Example

Let us refer back to Figure 4, where $n = 5$ and $t = 4$. The set of CSPs selected for sharing an attribute value $d_{ijk}$ is $SG_{jk} = \{\text{CSP}_1, \text{CSP}_2, \text{CSP}_3\}$. Thus, $UG_{jk} = \{\text{CSP}_4, \text{CSP}_5\}$, from which pseudo shares $HE_2(pk_{jkl}, ID_i)_{i \in \{1, 2, 3\}}$ where $pk_{jkl}$ is the primary key value of record $R_{jk}$, are computed. Polynomial $f_{jkl}(x)$ is created by Lagrange interpolation from $d_{jkl}$, its inner signature $s_{\text{in}ijk}$, pseudo shares and keys $K_d$ and $K_s$. Then shares of $d_{jkl}$ are: $e_{ijgl} = f_{jkl}(HF_{1}(ID_i))_{i \in \{1, 2, 3\}}$.

Assuming the set of $t$ CSPs selected for reconstruction is $RG = \{CSP_1, CSP_2, CSP_3, CSP_4\}$, $d_{jkl}$ is reconstructed from shares $e_{ijgl} \in \{1, 2\}$ (since $CSP_1, CSP_2 \in SG_{jk}$) and pseudo share $HE_2(pk_{jkl}, ID_i)_{i \in \{4, 5\}}$ (since $CSP_4, CSP_5 \in UG_{jk}$).

### 3.3 Outer Signatures

Outer data verification helps determine whether data integrity is compromised by CSPs (willingly or not). For this sake, we propose two new types of outer signatures: record and table signatures (whereas [2] uses an attribute value-level signature).

Moreover, we also propose a tree data structure to efficiently exploit outer signatures (Figure 5). Signature trees are stored at CSPs'. The maximum number of child nodes in the signature tree at CSP, is denoted $w_i$. Each signature tree is constituted of two subtrees: a table signature tree and record signature subtree. Leaf nodes of the record signature tree are record signatures. Higher-level nodes represent the signatures of record clusters, until the root, which is a table signature and thus a leaf of the table signature tree. Similarily, higher level nodes represent the signatures of table groups, until the root, which stores the whole DB's signature.

![Figure 5: Outer signature tree at CSP](image-url)
tree’s leaf level is \( u = u_e - \lfloor \log_{w_i} m \rfloor \) and the record signature tree’s leaf level is \( u = \lfloor \log_{w_i} r_{ij} \rfloor + 1 \).

3.3.1 Setup

For each CSP, the following parameters must be user-defined.

1. Determine \( w_i \).
2. Define one-way function \( HF_{e}(ER_{ij}) \).
3. Define homomorphic function \( HE_{e}(h) \), where
   \[ HE_{e}(h_1) \pm HE_{e}(h_2) = HE_{e}(h_1 \pm h_2) \]

3.3.2 Shared Table Creation

Whenever a new table \( ET_{ij} \) is created at CSP, the table signature tree is updated from leaf to root as follows.

1. Compute \( ET_{ij} \)'s table signature \( s_{tout_{ij}} = HF_{e}(0) \) and store it in a new right-most leaf node of the table signature tree.
2. Recursively create new parent nodes \((u,v)\) up to the root of the table signature tree such that \( u = 1 \ldots \lfloor \log_{w_i} j \rfloor \) and \( v = \lceil j/(w_i)^x \rceil \).
   (a) If the right-most node at level \( u \) bears the maximum number of children, i.e., \( v = (j-1) \mod (w_i)^u \), insert a new right-most parent node \((u,v)\) with value \( s_{tout_{ij}} \).
   (b) If there is no node at level \( u \) such that \( j = (w_i)^u - 1 \), insert a new root node \((u,1)\) with value \( s_{tout_{u1}} = s_{tout_{(u-1)1}} \).
   (c) Otherwise, stop recursion.

3.3.3 Shared Record Insertion

Whenever a new record \( ER_{ij} \) is inserted into shared table \( ET_{ij} \), the outer signature tree is updated from leaf to root as follows.

1. Compute record signature \( s_{rout_{ij}0g} = HF_{e}(ER_{ij}) \) and store it in a new right-most leaf node of \( ET_{ij} \)'s record signature tree.
2. Recursively insert or update parent nodes \((u,v)\) up to the root of \( ET_{ij} \)'s record signature tree such that \( u = 1 \ldots \lfloor \log_{w_i} g \rfloor \) and \( v = g/(w_i)^x \).
   (a) If the right-most node at level \( u \) bears the maximum number of children, i.e., \( v = (g-1) \mod (w_i)^u \), insert a new right-most parent node \((u,v)\) with value \( s_{rout_{ij}0g} \).
   (b) If there is no node in level \( u \) such that \( g = (w_i)^u - 1 \), insert a new root node \((u,1)\) with value \( s_{rout_{u1}} = HE_{e}(s_{rout_{(u-1)1}} + s_{rout_{ij}0g}) \). Compute signature change before and after insertion:
   \[ \Delta = HE_{e}(s_{rout_{ij}uv} - s_{rout_{ij}0g}) \] followed by update of root node’s value to
   \[ HE_{e}(s_{rout_{ij}uv} + \Delta) \] where \( \Delta = HE_{e}(s_{rout_{ij}uv} - s_{rout_{ij}0g}) \). Then, update record signature \( s_{rout_{ijuv}} \) to
   \[ HE_{e}(s_{rout_{ijuv}} + \Delta) \] (d) Otherwise, update the parent node’s signature \( s_{rout_{ijuv}} \) with

3.3.4 Shared Record Update

Whenever a record \( ER_{ij} \) is updated in shared table \( ET_{ij} \), the outer signature tree is updated from leaf to root as follows.

1. Compute signature change before and after insertion:
   \[ \Delta = HE_{e}(s_{rout_{ij}0g} - HF_{e}(ER_{ij})) \]
2. Recursively update nodes \((u,v)\) up to the root of \( ET_{ij} \)'s record signature tree such that \( u = 0 \ldots \lfloor \log_{w_i} h_{ij} \rfloor \) and \( v = \lceil j/(w_i)^x \rceil \). Update \( s_{rout_{ij}uv} \) to \( HE_{e}(s_{rout_{ij}uv} + \Delta) \).
3. Starting from the root of \( ET_{ij} \)'s record signature tree, recursively update parent nodes up to the root of the table signature tree such that \( u = 1 \ldots \lfloor \log_{w_i} m \rfloor \) and \( v = \lceil j/(w_i)^x \rceil \). Then, update table signature \( s_{tout_{ijuv}} \) to \( HE_{e}(s_{tout_{ijuv}} + \Delta) \).

3.4 Sharing Data Warehouses

Since each table of a shared DW is stored in a relational database at a given CSP’s and each attribute value in each record is encrypted independently, our approach straightforwardly helps implement any DW logical model, i.e., star, snowflake and constellation schemas. A shared DW bears the same schema as the original DW’s. However, all encrypted attributes are transformed into reals by the sharing process (Section 3.2).

Finally, to improve query performance, computation and data transfer costs when performing Relational OLAP (ROLAP) operations, we use indices and cloud cubes, which are described below.

3.4.1 Indices

We exploit three types of indices, which are all created at data-sharing time. In addition to so-called Type I indices used in the reconstruction process (Section 3.2), Type II and III indices are specifically aimed at enhancing query performance and thus computation cost.

Type II indices are customarily used by the second family of secret sharing approaches (Section 2 [11, 13, 21]). They allow computing exact match, range and aggregation (e.g.,
shared tables. Let us illustrate why they are needed through variance and standard deviation, as well as multiplications (MAX, MIN, MEDIAN and COUNT) queries, without reconstructing all values of \(X\) (i.e., decrypt) all values of \(X\) or we share \(X^2\) values in a new attribute, i.e., a Type III index, and computation can operate directly on shares. Similarly, computing \(\text{SUM}(X \times Y)\) or \(\text{SUM}(X \div Y)\) requires sharing \(X \times Y\) and \(X \div Y\) as Type III indices, respectively, because homomorphism properties only work for summation and subtraction.

### 3.4.2 Cloud Cubes

As in [2], our approach supports the storage of data cubes that optimize response time and bandwidth when performing ROLAP operations. In addition, in this work, cubes are directly created in the cloud and refreshed through shares and indices only.

Since cloud cubes are built from shares, they are physically stored into tables that must be shared at all n CSPs, because pseudo shares are not available. In addition to customary dimension references and aggregate measures, they can include additional attributes that actually are embedded Type III indices. For example, suppose we need to compute the average of measure \(M\) from a cube. Then \(\text{SUM}(M)\) and \(\text{COUNT}(M)\) must be stored in the cube too, to allow computations on shares without reconstruction.

Figure 7 features a cloud cube named Cube-I that sums total prices and numbers of sales by time period and by product. As customary, NULL values are used to encode superaggregates. All aggregate measures can be queried directly from Cube-I without reconstruction.

### 3.5 Backup, Recovery and Load processes

In our approach, as in all secret sharing approaches, backups are unnecessary because each shared table \(ET_{ij}\) is actually a backup share of all other shared table \(ET_{ia}\), where \(a \in 1, \ldots, j-1, j+1, \ldots, n\). In case shared tables or shared records are detected as erroneous by outer signature verification (Section 3.3), their can be recovered from other shared tables.

Loading new data into an existing shared DW does not require decrypting previous data first, because each attribute value in each record is encrypted independently. For instance, in Figure 8, data from Figure 3 are already shared and the last record (\#127) is new. After each new shared record is loaded, the outer signature tree must be updated (Section 3.3), and indices and possible cloud cubes refreshed.

Cloud cube refreshment requires further attention. When updating cubes, aggregates can be updated from shares and indices. There are three cases.

For aggregations by, e.g., MAX and MIN operators, the primary key of aggregates is discovered from a Type II index. Then, at each CSP’s, the share corresponding to the primary key is updated in the cloud cube. In case no corresponding share is found at that CSP’s, the shared cube is updated from a pseudo share with the help of a Type I index.

For COUNTs, aggregates can be easily found from a Type II

<table>
<thead>
<tr>
<th>YearID</th>
<th>MonthID</th>
<th>DateID</th>
<th>CategoryID</th>
<th>Product</th>
<th>ProName</th>
<th>CategoryID</th>
<th>ProDescr</th>
<th>UnitPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>NULL</td>
<td>NULL</td>
<td>125</td>
<td>95</td>
<td>125</td>
<td>95</td>
<td>125</td>
<td>95</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>125</td>
<td>95</td>
<td>125</td>
<td>95</td>
<td>125</td>
<td>95</td>
</tr>
</tbody>
</table>

Figure 7: Sample cloud cube Cube-I

<table>
<thead>
<tr>
<th>(Shares)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Attributes</td>
</tr>
<tr>
<td>YearID</td>
</tr>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Figure 8: Example of sharing new data

index and reconstructed. Then, the aggregates can be updated and shared back into the cloud cube [16].

For SUMs, thanks to the homomorphism property, aggregates can be updated from cloud cubes by summing shares and pseudo shares. For example, to update \(\text{SUM}(X)\) at \(CSP_i\), the new aggregate is the sum of shares and pseudo shares (Equation 2, where \(\text{SUM}_{CSP_i}(X)\) is trivially computed at \(CSP_i\) and \(\text{SUM}_{index}(PK_i)\) is computed with the help of a Type I index).

\[
\text{SUM}(X) = \text{SUM}_{CSP_i}(X) + \text{HE}_2(\text{SUM}_{index}(PK_i), ID_i)
\]  

(2)

Finally, more complex aggregations require combining the above cases. For example, \(\text{SUM}(X \pm Y)\) is updated from shares and pseudo shares by Equation 3.

\[
\text{SUM}(X \pm Y) = \text{SUM}_{CSP_i}(X \pm Y) + 2 \times \text{HE}_2(\text{SUM}_{index}(PK_i), ID_i)
\]  

(3)
3.6 Querying a Shared Data Warehouse

Simple SELECT/FROM queries directly apply onto shares, as well as summing positive integer attributes. All join operators, when operating on unencrypted keys, also apply directly. When expressing conditions in a WHERE or HAVING clause, Type II indices must be used [11, 13, 21]. Almost all comparison operators (\(=\), \(!=\), EXISTS, IN, \(>\), \(\geq\), \(<\), \(\leq\), BETWEEN...) can be evaluated against such B++ index trees.

Similarly, aggregation functions such as MAX, MIN and COUNT can directly apply on shares with the help of Type II indices [11, 13, 21]. In contrast, a SUM must combine relevant aggregates of shares and pseudo shares (as in Equation 2) with an external program before reconstruction.

Other aggregation functions must be computed by an external program after reconstructing relevant aggregates from shares and pseudo shares. Average, variance and standard deviation are computed as in Equations 4, 5 and 6, respectively, where \(X^2\) is a Type III index.

\[
\text{AVG}(X) = \frac{DC(SUM(X))}{DC(COUNT(X))}
\]

\[
\text{VAR}(X) = \frac{DC(SUM(X^2))}{DC(COUNT(X))} - \left(\frac{DC(SUM(X))}{DC(COUNT(X))}\right)^2
\]

\[
\text{STDEV}(X) = \sqrt{\frac{DC(SUM(X^2))}{DC(COUNT(X))} - \left(\frac{DC(SUM(X))}{DC(COUNT(X))}\right)^2}
\]

When aggregating calculated fields, multiplication and division can be performed directly from Type III indices. However, summation and subtraction between two attributes must combine relevant aggregates of shares and pseudo shares with an external program before reconstruction. Aggregates at CSP, compute as in Equation 3.

Finally, grouping queries using the GROUP BY or GROUP BY CUBE clauses can directly apply on shares if they target unencrypted key attributes. Again, grouping by other attribute(s) requires the use of a Type II index.

Consequently, executing some queries may require either transforming or splitting the query, depending on its clauses and operators, following the above guidelines. Figure 9 shows a sample query and the way it runs at the user’s, and operators, following the above guidelines. Figure 9 with an external program before reconstruction.

Other aggregation functions must be computed by an external program. Aggregates at CSP, compute as in Equation 3.

Finally, grouping queries using the GROUP BY or GROUP BY CUBE clauses can directly apply on shares if they target unencrypted key attributes. Again, grouping by other attribute(s) requires the use of a Type II index.

Consequently, executing some queries may require either transforming or splitting the query, depending on its clauses and operators, following the above guidelines. Figure 9 shows a sample query and the way it runs at the user’s, and operators, following the above guidelines. Figure 9 with an external program before reconstruction.

Finally, grouping queries using the GROUP BY or GROUP BY CUBE clauses can directly apply on shares if they target unencrypted key attributes. Again, grouping by other attribute(s) requires the use of a Type II index.

Consequently, executing some queries may require either transforming or splitting the query, depending on its clauses and operators, following the above guidelines. Figure 9 shows a sample query and the way it runs at the user’s, and operators, following the above guidelines. Figure 9 with an external program before reconstruction.

Finally, grouping queries using the GROUP BY or GROUP BY CUBE clauses can directly apply on shares if they target unencrypted key attributes. Again, grouping by other attribute(s) requires the use of a Type II index.

Consequently, executing some queries may require either transforming or splitting the query, depending on its clauses and operators, following the above guidelines. Figure 9 shows a sample query and the way it runs at the user’s, and operators, following the above guidelines. Figure 9 with an external program before reconstruction.

Finally, grouping queries using the GROUP BY or GROUP BY CUBE clauses can directly apply on shares if they target unencrypted key attributes. Again, grouping by other attribute(s) requires the use of a Type II index.

Consequently, executing some queries may require either transforming or splitting the query, depending on its clauses and operators, following the above guidelines. Figure 9 shows a sample query and the way it runs at the user’s, and operators, following the above guidelines. Figure 9 with an external program before reconstruction.

Finally, grouping queries using the GROUP BY or GROUP BY CUBE clauses can directly apply on shares if they target unencrypted key attributes. Again, grouping by other attribute(s) requires the use of a Type II index.

Consequently, executing some queries may require either transforming or splitting the query, depending on its clauses and operators, following the above guidelines. Figure 9 shows a sample query and the way it runs at the user’s, and operators, following the above guidelines. Figure 9 with an external program before reconstruction.
of CSP failure(s), simply by not selecting the failing CSP(s) for sharing new data. This is again possible because data is shared at $n - t + 2$ CSPs instead of $n$.

Finally, to enforce integrity, only [2] and fVSS verify both the correctness of shares and the honesty of CSPs by outer and inner code verification, respectively. Only two other approaches use inner code verification alone. The novelty of fVSS is that outer signatures are computed at the record and table granularity, instead of the attribute value’s, which allows faster verification, e.g., when checking one table signature instead all signatures of one or several attributes in said table.

4.2 Storage Cost

Storage cost directly depends on shared data volume, which in turn depends on parameters $n$ and $t$. Figure 10 plots the volume of shared data for all studied approaches, expressed as a multiple of original data volume $V$, with respect to $n$, with $t = n$ in the upper graph and $t = 3$ in the lower graph. Figure 10 shows that fVSS help control shared data volume better than most existing approach, and is close to the best approaches [2, 21] in this respect.

However, storage cost does not only depend on the global volume of shares. Since fVSS allows selecting the data volume shared at each CSP, it can be differentiated to benefit from different pricing policies. Let us illustrate this through an example. Let $n = 5$, $t = 4$ and $V = 100$ GB. CSP pricing policies for this range of data volume are depicted in Table 3. Prices are real prices from CSPs such as Amazon web services, Windows azure and Google compute engine. Finally, let us assume that each individual share is not bigger than the original data it encrypts, e.g., a shared integer is not bigger than the original unencrypted integer. We also disregard index and signature volume, which depends on user-defined parameters in all approaches, for the sake of simplicity.

| Table 2: Comparison of database sharing approaches |
|----------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Features and costs               | [1]             | [2]             | [8]             | [9]             | [11]            | [12, 13]        | [19]            | [21]            |
| Data privacy                     | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             |
| Data availability                | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             |
| Ability in case CSPs fail to     | - Query shares  | No              | No              | Yes             | Yes             | Yes             | Yes             | Yes             |
| - Update shares                  | No              | No              | No              | No              | No              | No              | No              | No              |
| Data integrity                   | - Inner code verifying | No | No | No | No | Yes | Yes | Yes |
| - Outer code verifying           | No              | Yes             | No              | No              | No              | No              | No              | No              |
| Target                           | DBs             | DWs             | DWs             | DBs             | DBs             | DBs             | DBs             | DBs             |
| Data sources                     | Single          | Single          | Multi           | Multi           | Single          | Single          | Single          | Single          |
| Data types                       | Positive integers | Positive integers | Positive integers | Positive integers | Positive integers | Positive integers | Positive integers | Positive integers |
| Shared data access               | - Updates       | No              | Yes             | No              | Yes             | Yes             | Yes             | Yes             |
| - Exact match queries            | No              | No              | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             |
| - Range queries                  | No              | No              | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             |
| - Aggregation queries on one attribute | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| - Aggregation queries on two attributes | No | No | No | No | No | No | No | Yes |
| - Grouping queries               | No              | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             | Yes             |
| Complexity                       | - Data storage w.r.t. original data volume | $\geq n$ | $\geq \frac{n(t - 1)}{2} + \text{signatures}$ | $\geq 2n$ | $\geq 2n$ | $\geq n$ | $\geq n + 1$ (B++ tree) | $\geq n + 1$ (hash tree) | $\geq n + n/t$ (B++ tree) + signatures | $\geq n + n + 2$ + $t$ + 2 (B++ tree) |
| - Sharing time                   | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ |
| - Reconstruction time            | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ | $O\left(\frac{n^2 t^2}{n^2 + t^2}\right)$ |

| Table 3: CSP pricing policies |
|---------------------|---------------------|---------------------|---------------------|---------------------|
| CSP                | $\text{Storage ($/GB/month)$}$ | $\text{VM CPU time ($/h)$}$ | $\text{VM CPU time ($/h)$}$ |
| CSP1               | 0.030               | 0.040               | 0.033               |
| CSP2               | 0.050               | 0.059               | 0.098               |
| CSP3               | 0.060               | 0.090               | 0.120               |
| CSP4               | 0.070               | 0.120               | 0.140               |
| CSP5               | 0.220               | 0.280               |

Table 4 features a storage cost comparison of all studied approaches. The first line relates to unencrypted data stored at one CSP, for reference. Global share volume is computed with respect to data storage complexities (Table 2). Storage cost is the sum of data volumes stored at each CSP, times CSP’s storage price. We included two strategies for fVSS. In fVSS-I, data are equitably shared among CSPs. In fVSS-II, data are preferentially shared at CSP1, CSP2 and CSP3, which are the cheapest. Results clearly show that fVSS-II achieves a much lower cost than fVSS-I. Moreover, even though global share volume with fVSS-II is significantly larger than the most efficient previous approaches [2, 21], final cost is comparable, and even a little lower. However, [21] is not applicable in our context since it does not allow aggregation queries. It is thus discarded in the following.

| Table 4: Storage cost comparison |
|-------------------------------|--------------------|--------------------|--------------------|
| Approach                       | Share volume (GB)  | Storage cost ($)   |
| Unencrypted data               | 100                | 100                | 3 to 32.5          |
| [8, 9, 19]                     | 1900               | 200                | 113.00             |
| [1, 11, 12, 13]                | 500                | 100                | 56.80              |
| [2]                            | 167                | 34                 | 19.31              |
| fVSS-I                         | 125                | 25                 | 14.77              |
| fVSS-II                        | 300                | 99.8 + 99.8 + 99.8 | 12.39              |

The price of each attribute is as follows $\text{Price} = \text{price of attribute} \times \text{granularity}$, where the granularity is the cardinality of the set of all possible values of the attribute and $\text{price of attribute}$ is the price of one unit of the attribute, e.g., the price of one integer. The granularity is the integer, Booleans (e.g., 1), Strings (e.g., 10), Dates (e.g., 1), and Others (e.g., 1).
of records in the query response, $\gamma$ (Table 2). All secret sharing approaches bear the same reconstruction complexity but [21]. However, this approach cannot compute aggregations on shares, implying all records involved in aggregations must be reconstructed at the user’s, which is more costly than computing the aggregation on shares and only reconstruct the result. Moreover, we expect IVSS to be more efficient than previous approaches because we can directly perform all query types on shared DWs and cubes, in parallel, whereas other approaches cannot and must reconstruct bigger datasets before processing them at the user’s.

Let us illustrate this through an example. Let $n = 5$, $t = 4$. Let us assume we run a query matched by 10% of records, i.e., $\gamma = 10^{14}$, and $RG = \{CSP_1, CSP_2, CSP_3, CSP_4\}$. CSP pricing policies and virtual machine power are the same as in Section 4.3.1.

Table 6 features a data access cost comparison of all studied approaches but [21]. Response time is the number of processed records divided by the virtual machine’s computing power. CPU cost is the sum of response times at each CSP times CSP’s computing price. Results show again that, even though response time is comparable for all approaches, IVSS allows much lower costs, especially when unbalancing the volume of shares at CSPs in IVSS-II, which helps decrease cost by a factor 4 with respect to state-of-the-art approaches in this example.

4.4 Data Transfer Cost

Data transfer cost directly relates to the size of query results when accessing the shared DW. Since all approaches allow different operations and vary in share volume, it is difficult to compare data transfer cost by proof. However, to reduce data transfer cost, IVSS allows several aggregation operators running on shares. Moreover, by creating shared data cubes, we allow straight computations on shares, and thus only target results are transferred to the user, i.e., with no additional data reconstruction, and thus no stored data transfer.

5. CONCLUSION AND PERSPECTIVES

In this paper, we propose a new approach for securing cloud DWs, which simultaneously supports data privacy,
availability, integrity and OLAP. Our approach builds upon FVSS, which is to the best of our knowledge the first flexible secret sharing that allows users adjusting share volume with respect to CSP pricing polices. Our experiments show that unbalancing share volume at CSPs allows significantly minimizing storage and computing costs in the pay-as-you-go paradigm. Privacy and availability are achieved by design with secret sharing, but FVSS achieves a higher security level and allows DW refreshing even when some CSPs fail. Finally, data integrity is reinforced with both inner and outer signature that help detect errors in query results and shares, respectively.

Future research shall run along three lines. First, we plan to further assess the cost of our solution in the cloud pay-as-you-go paradigm. We especially plan to balance the cost of our solution against the cost of risking data loss or theft. Moreover, since CSP pricing and servicing policies are likely to evolve quickly, we aim at designing a method for adding and removing CSPs to/from the CSP pool, with the lowest possible update costs and while preserving data integrity. Second, we aim at designing a tool that semi-automatically helps users adjust the volume of shares at each CSP’s, with respect to cost, but also quality of service. Finally, we also work on share storage management, to optimize query performance and reduce both response time and computing cost.

6. REFERENCES


