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

Varunya Attasena, Nouria Harbi, Jérôme Darmont

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

Varunya Attasena
Université de Lyon (ERIC)
Université Lumière Lyon 2
5 av. Pierre Mendès-France
69676 Bron Cedex – France
vattasena@eric.univ-lyon2.fr

Nouria Harbi
Université de Lyon (ERIC)
Université Lumière Lyon 2
5 av. Pierre Mendès-France
69676 Bron Cedex – France
nouria.harbi@univ-lyon2.fr

Jérôme Darmont
Université de Lyon (ERIC)
Université Lumière Lyon 2
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 $t \leq n$ 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-

DOLAP'14, November 7, 2014, Shanghai, China.

ACM 978-1-4503-0999-8/14/11 ...$15.00.
http://dx.doi.org/10.1145/2666158.2666173.

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 (fVSS). To the best of our knowledge, fVSS is the first approach allowing DW refreshment when one or several CSPs fail. Moreover, fVSS allows running OLAP operators on shared DWs or cubes without reconstructing all data first. fVSS also features both inner and outer signatures for data verification. Finally, fVSS 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 fVSS. 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 fVSS 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 $\{CSP_i\}_{i>n}$, 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

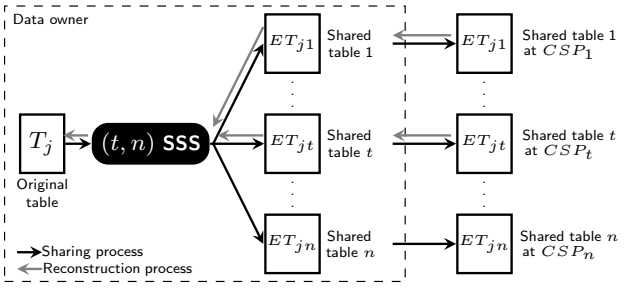


Figure 1: First strategy for sharing a database

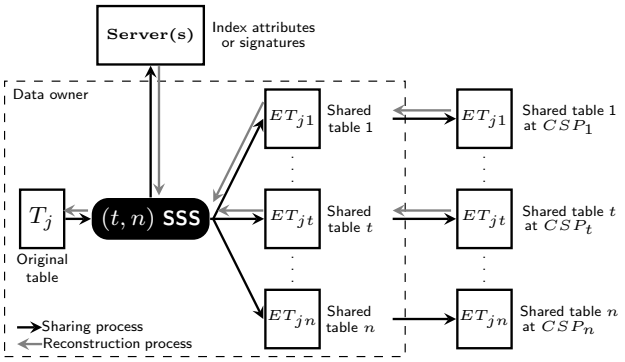


Figure 2: Second strategy for sharing a database

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 fVSS' 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 fVSS' 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 d_{jkl}, e.g., the category ID of product #124 in Figure 3(a). We also need to share its inner signature s_in_{jkl} 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: fVSS parameters

Parameters	Definitions
p	A big prime number
n	Number of CSPs
t	Number of shares necessary for reconstructing original data
CSP_i	CSP number i
ID_i	Identifier number of CSP_i such that $p > ID_i > 1$
m	Number of tables
T_j	Table number j such that $T_j = \{R_{j,k}\}_{k=1...r_j}$
ET_{ij}	Shared table of T_j stored at CSP_i
q_j	Number of attributes of T_j and ET_{ij} (not including primary key)
r_j	Number of records of T_j
er_{ij}	Number of records of ET_{ij} such that $r_j \geq er_{ij}$ and $\sum_{i=1}^{n} er_{ij} = (n-t+2)\, r_j$
A_{jl}	Attribute number l of T_j and ET_{ij}
R_{jk}	Record $\#k$ of T_j such that $R_{jk} = \{pk_{jk}, d_{jk1} \dots d_{jkq_j}\}$
ER_{ijg}	Record $\#g$ of ET_{ij} such that $ER_{ijg} = \{pk_{ijg}, e_{ijg1} \dots e_{ijgq_j}\}$ ER_{ijg} is shared record of R_{jk} if $pk_{jk} = pk_{ijg}$
pk_{jk}	Primary key value of R_{jk}
pk_{ijg}	Primary key value of ER_{ijg}. It is not encrypted.
d_{jkl}	Value of A_{jl} of R_{jk} such that $p > d_{jkl} \geq 0$
e_{ijgl}	Share of d_{jkl} stored in A_{jl} of ER_{ijg} such that $p > e_{ijgl} \geq 0$.
SG_{jk}	Group of CSPs that store shares of R_{jk} such that $SG_{jk} \subset \{CSP_i\}_{i=1..n}$ and $\lvert SG_{jk}\rvert = n-t+2$
UG_{jk}	Group of CSPs that do *not* store shares of R_{jk} such that $UG_{jk} \subset \{CSP_i\}_{i=1..n}$ and $UG_{jk} = \{CSP_i\}_{i=1..n} - SG_{jk}$
RG	Group of CSPs selected to reconstruct data such that $RG \subset \{CSP_i\}_{i=1..n}$ and $\lvert RG\rvert = t$
s_in_{jkl}	Inner signature of d_{jkl} such that $p > s\_in_{jkl} \geq 0$
s_rout_{ijuv}	Outer record signature number v, level u of ER_{ijg} in ET_{ij} stored at CSP_i
s_tout_{iuv}	Outer table signature number v, level u of ET_{ij} stored at CSP_i
w_i	Maximum number of child nodes in CSP_i's outer signature tree

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	Shirt	Red	1	75
125	Shoe	*NULL*	2	80
126	Ring	*NULL*	1	80

(a) Original data

ProNo	Share location
124	10101
125	01110
126	11010

(b) Indices on index server

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	{6,5,3,11,7}	{10,5,8}	1	6
126	{10,3,6,12}	*NULL*	2	45

(c) Shares at CSP_1

ProNo	ProName	ProDescr	CategoryID	UnitPrice
125	{6,5,4,5}	*NULL*	2	5
126	{2,6,11,10}	*NULL*	6	8

(d) Shares at CSP_2

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	{6,6,5,7,9}	{12,8,1}	4	7
125	{6,5,8,3}	*NULL*	9	11

(e) Shares at CSP_3

ProNo	ProName	ProDescr	CategoryID	UnitPrice
125	{9,15,13,8}	*NULL*	12	7
126	{2,7,6,9}	*NULL*	12	1

(f) Shares at CSP_4

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	{5,9,11,1,5}	{10,6,7}	8	13

(g) Shares at CSP_5

Figure 3: Sample original and shared data

schemes use random polynomials. In contrast, we construct a polynomial by Lagrange interpolation using d_{jkl}, s_in_{jkl} and the $t-2$ pseudo shares. Then, we can share d_{jkl} and s_in_{jkl} at $n-t+2$ CSPs. Figure 4 plots an example where $t = 4$. Shares e_{1jgl}, e_{2jgl} and e_{3jgl} are created from polynomial $f_{jkl}(x)$ of degree $t-1=3$.

To reconstruct d_{jkl}, let us assume we select the following set of t CSPs: $RG = \{CSP_1, CSP_2, CSP_4, CSP_5\}$. Then, if e_{ijgl} is stored at CSP_i, it is used for reconstruction. Otherwise, the corresponding pseudo share is used instead. To ease this operation, bitmaps representing where shares are stored are maintained in the index server(s). For example, the bitmap corresponding to product #124 in Figure 3 is 10101, with a 1 value at position $\#i$ representing share storage at CSP_i.

The remainder of this section details the sharing and reconstruction processes (Section 3.2), our novel outer signatures (Section 3.3) and the way we share data warehouses

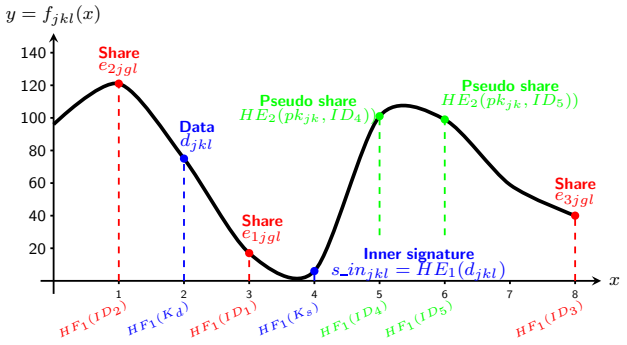


Figure 4: Sample sharing process

(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 fVSS, 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_{jkl} of attribute A_{jl} in record R_{jk} from table T_j is encrypted into $n-t+2$ shares generated from a polynomial $f_{jkl}(x)$ of degree $t-1$ created by Lagrange interpolation from d_{jkl}, its inner signature s_in_{jkl}, the identifier numbers ID_i of the CSPs selected to store the shares (this set of CSPs is denoted SG_{jk}), and two generated keys K_d and K_s for data and signatures, respectively. The inner signature we use is very similar to that of [2]. It is created from d_{jkl} by homomorphic encryption [7]. s_in_{jkl} matches with d_{jkl} 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 $HE_2(a,b)$, a and b are reals and $HE_2(a_1,b) + HE_2(a_2,b) = HE_2(a_1+a_2,b)$.
5. Set values of CSP identifiers $ID_{i=1..n}$, K_d and K_s such that their values range in $]0,p[$. All $HF_1(ID_i)$ must be unique and different from $HF_1(K_d)$ and $HF_1(K_s)$.

3.2.2 *Data Sharing Process*

Any record R_{jk} is encrypted independently as follows.

1. Determine the group of CSPs SG_{jk} that will store R_{jk}'s $n-t+2$ shares. Let UG_{jk} be the group of CSPs that do not store R_{jk}'s shares, i.e., $UG_{jk} = \{CSP_i\}_{i=1..n} - SG_{jk}$.

2. For each attribute A_{jl}:

   (a) Compute d_{jkl}'s inner signature: $s\_in_{jkl} = HE_1(d_{jkl})$.

   (b) Create polynomial $f_{jkl}(x)$ of degree $t-1$ by Lagrange interpolation (Equation 1):

$$f_{jkl}(x) = \sum_{\alpha=1}^{t} \prod_{1\leq\beta\leq t,\alpha\neq\beta} \frac{x - x_\beta}{x_\alpha - x_\beta} \times y_\alpha \qquad (1)$$

   where $\{(x_1, y_2), \ldots, (x_t, y_t)\} = \{(HF_1(K_d), d_{jkl}), (HF_1(K_s), s\_in_{jkl})_{CSP_i \in SG_{jk}}\} \cup \{(HF_1(ID_i), HE_2(pk_{jk}, ID_i))_{CSP_i \in UG_{jk}}\}$. $(HF_1(ID_i), HE_2(pk_{jk}, ID_i))$ are pseudo shares.

   (c) Compute the set of d_{jkl}'s $n-t+2$ shares $\{e_{ijgl}\}$. $\forall CSP_i \in SG_{jk}$: $e_{ijgl} = f_{jkl}(HF_1(ID_i))$, with $pk_{jk} = pk_{ijg}$.

Following this routine, record R_{jk} is shared into $n-t+2$ records ER_{ijg} at CSPs in SG_{jk}. The relationship between R_{jk} and ER_{ijg} is maintained through primary keys $pk_{jk} = pk_{ijg}$. 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. Data of other types (i.e., reals, characters, strings and binary strings) are first transformed into integers before being shared [2].

3.2.3 Data Reconstruction Process

Any attribute value d_{jkl} is reconstructed as follows.

1. Select t CSPs to form reconstruction group RG.
2. For each $CSP_i \in RG$, if outer data verification (Section 3.3) outputs an error, replace CSP_i by another CSP selected from $\{CSP_i\}_{i=1..n} - RG$.
3. For each $CSP_i \in SG_{jk} \cap RG$, load share e_{ijgl} into y_i where $pk_{jk} = pk_{ijg}$.
4. For each $CSP_i \in UG_{jk} \cap RG$, compute pseudo share $y_i = HE_2(pk_{jk}, ID_i)$.
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 $s\_in_{jkl} = f_{jkl}(HF_1(K_s))$.
8. Verify d_{jkl}'s correctness: if $s\_in_{jkl} \neq HE_1(d_{jkl})$, 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_{jkl} is $SG_{jk} = \{CSP_1, CSP_2, CSP_3\}$. Thus, $UG_{jk} = \{CSP_4, CSP_5\}$, from which pseudo shares $HE2(pk_{jk}, ID_i)_{i\in\{4,5\}}$, where pk_{jk} 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_in_{jkl}, pseudo shares and keys K_d and K_s. Then shares of d_{jkl} are: $e_{ijgl} = f_{jkl}(H1(ID_i))_{i\in\{1,2,3\}}$.

Assuming the set of t CSPs selected for reconstruction is $RG = \{CSP_1, CSP_2, CSP_4, CSP_5\}$, d_{jkl} is reconstructed from shares $e_{ijgl\ i\in\{1,2\}}$ (since $CSP_1, CSP_2 \in SG_{jk}$) and pseudo shares $HE2(pk_{jk}, ID_i)_{i\in\{4,5\}}$ (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_i 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. Similarly, higher level nodes represent the signatures of table groups, until the root, which stores the whole DB's signature.

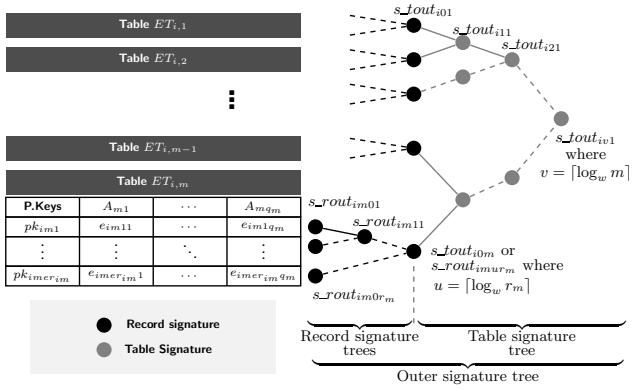


Figure 5: Outer signature tree at CSP_i

Signatures are checked before reconstructing data (Section 3.2.3). Data integrity can also be verified on-demand. Outer record signatures are created with the help of user-defined one-way functions $HF_{*i}(h)$. Shared record ER_{ijg} passes integrity check if $s\_rout_{ij0g} = HF_{*i}(ER_{ijg})$.

Similarly, shared table ET_{ij} passes integrity check if $s\_tout_{i0j} = HE_{*i}(\sum_{g=1}^{er_{ij}} HF_{*i}(ER_{ijg}))$, where $HE_{*i}(h)$ are homomorphic functions and er_{ij} is the number of records in ET_{ij}. Thanks to the homomorphism property [7], all tables are correct if a signature stored in a root node equals the sum of the signatures stored in all its child nodes. This allows directly checking a set of records or tables through one node only, and thus speeds up data integrity verification. Moreover, in case of error, we can also discover where the integrity breach is by testing whether each node respects $s\_out_{iuv} = HE_{*i}(\sum_{a=(v-1)\times w_i}^{v\times w_i} s\_out_{i(u-1)a})$, where s_out_{iuv} may either be a record or a table outer signature, $u > 0$ is a level number and v a node number in level u.

Finally, note that the signature tree's root level is $u_r = \lceil \log_{w_i} m \rceil + \lceil \log_{w_i} \max(er_{ij})_{j=1\cdots m} \rceil - 1$, the table signature

tree's leaf level is $u_t = u_r - \lceil \log_{w_i} m \rceil$ and the record signature tree's leaf level is $u_t - \lceil \log_{w_i} er_{ij} \rceil + 1$.

3.3.1 Setup

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

1. Determine w_i.
2. Define one-way function $HF_{*i}(ER_{ijg})$.
3. Define homomorphic function $HE_{*i}(h)$, where $HE_{*i}(h_1) \pm HE_{*i}(h_2) = HE_{*i}(h_1 \pm h_2)$.

3.3.2 Shared Table Creation

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

1. Compute ET_{ij}'s table signature $s\_tout_{i0j} = HF_{*i}(0)$ 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 $u = 1 \ldots \lceil \log_{w_i} j \rceil$ and $v = \lceil j/(w_i)^u \rceil$.
   (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_{i0j}.
   (b) If there is no node at level u such that $j = (w_i)^{u-1} + 1$, insert a new root node $(u, 1)$ with value $s\_tout_{iu1} = s\_tout_{i(u-1)1}$.
   (c) Otherwise, stop recursion.

3.3.3 Shared Record Insertion

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

1. Compute record signature $s\_rout_{ij0g} = HF_{*i}(ER_{ijg})$ 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 $u = 1 \ldots \lceil \log_{w_i} g \rceil$ and $v = \lceil g/(w_i)^u \rceil$.
   (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_{ij0g}.
   (b) If there is no node in level u such that $g = (w_i)^{u-1} + 1$, insert a new root node $(u, 1)$ with value $s\_rout_{iu1} = HE_{*i}(s\_rout_{ij(u-1)1} + s\_rout_{ij0g})$. Compute signature change before and after insertion: $\Delta = HE_{*i}(s\_rout_{ij(u-1)1} - s\_rout_{ij0g})$.
   (c) If root node has been reached, compute signature change before and after insertion: $\Delta = HE_{*i}(s\_rout_{ijuv} - s\_rout_{ij0g})$. Then, update the root node's value to $HE_{*i}(s\_rout_{ijuv} + s\_rout_{ij0g})$.
   (d) Otherwise, update the parent node's signature s_rout_{iuv} with $HE_{*i}(s\_rout_{ijuv} + s\_rout_{ij0g})$.
3. Starting from the root of ET_{ij}'s record signature tree, which is also leaf $s\_tout_{i(0)j}$ of the corresponding table signature tree, recursively update parent nodes up to the root of the table signature tree such that $u = 1 \ldots \lceil \log_{w_i} m \rceil$ and $v = \lceil j/(w_i)^u \rceil$. Then, update table signature s_tout_{iuv} to $HE_{*i}(s\_tout_{iuv} + \Delta)$.

An example of outer signature tree update is provided in Figure 6. Record $ER_{i(3)(10)}$ is a new record; all white nodes are updated or inserted.

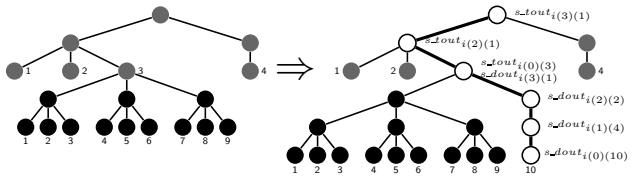


Figure 6: Sample outer signature tree update

3.3.4 Shared Record Update

Whenever a record ER_{ijg} 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: $\Delta = HE_{*i}(s\_rout_{ij0g} - HF_{*i}(ER_{ijg})$.
2. Recursively update nodes (u, v) up to the root of ET_{ij}'s record signature tree such that $u = 0 \ldots \lceil \log_{w_i} er_{ij} \rceil$ and $v = \lceil g/(w_i)^u \rceil$. Update s_rout_{ijuv} to $HE_{*i}(s\_rout_{ijuv} + \Delta)$.
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 $u = 1 \ldots \lceil \log_{w_i} m \rceil$ and $v = \lceil j/(w_i)^u \rceil$. Update table signature s_tout_{iuv} to $HE_{*i}(s\_tout_{iuv} + \Delta)$.

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.,

MAX, MIN, MEDIAN and COUNT) queries, without reconstructing data, with the help of B++ trees. Type II indices are independently stored in the index server(s).

Type III indices help compute aggregate functions such as variance and standard deviation, as well as multiplications and divisions between two attributes, without reconstructing data. Type III indices are stored as extra attributes in shared tables. Let us illustrate why they are needed through examples. To compute the variance and standard deviation over an attribute X, X^2 values are needed. Then, either we reconstruct (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 SUM($X \times Y$) or 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 SUM(M) and 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 is 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 t 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 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 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 $SUM_{index}(PK_i)$ is computed with the help of a Type I index).

$$SUM(X) = SUM_{CSP_i}(X) + HE_2(SUM_{index}(PK_i), ID_i) \quad (2)$$

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

$$SUM(X \pm Y) = SUM_{CSP_i}(X + Y) + 2 \times HE_2(SUM_{index}(PK_i), ID_i) \quad (3)$$

Foreign Keys | Time Attributes | Product Attributes | (Shares) Aggregation Attributes

YearID	MonthID	DateID	CategoryID	ProdNo	TotalPrice	Number
NULL	*NULL*	*NULL*	*NULL*	*NULL*	83231	58244
NULL	*NULL*	*NULL*	1	*NULL*	26701	18254
NULL	*NULL*	*NULL*	1	1	8958	7113
NULL	*NULL*	*NULL*	1	⋮	⋮	⋮
NULL	*NULL*	*NULL*	1	2	4348	1844
NULL	*NULL*	*NULL*	⋮	⋮	⋮	⋮
1	*NULL*	*NULL*	*NULL*	*NULL*	44574	54542
1	*NULL*	*NULL*	1	*NULL*	21158	8954
1	*NULL*	*NULL*	1	1	9754	4544
1	*NULL*	*NULL*	1	⋮	⋮	⋮
1	*NULL*	*NULL*	2	1	18444	5747
1	*NULL*	*NULL*	⋮	⋮	⋮	⋮
1	1	*NULL*	*NULL*	*NULL*	8312	5812
1	1	*NULL*	1	*NULL*	2312	1822
1	1	*NULL*	1	1	988	586
1	1	*NULL*	1	⋮	⋮	⋮
1	1	*NULL*	2	1	756	458
1	1	*NULL*	⋮	⋮	⋮	⋮
1	2	*NULL*	*NULL*	*NULL*	9758	6254
1	⋮	*NULL*	⋮	⋮	⋮	⋮
1	1	1	*NULL*	*NULL*	2578	1587
1	1	1	1	*NULL*	548	425
1	1	1	1	1	56	24
1	1	1	1	⋮	⋮	⋮
1	1	1	1	2	95	67
1	1	1	1	⋮	⋮	⋮
1	1	1	2	*NULL*	689	357
1	1	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 7: Sample cloud cube Cube-I

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	Shirt	Red	1	75
125	Shoe	*NULL*	2	80
126	Ring	*NULL*	1	80
127	Hat	*NULL*	3	5

(a) Original data

ProNo	Share location
124	10101
125	01110
126	11010
127	00111

(b) Type I indices

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	{6,5,3,11,7}	{10,5,8}	1	6
126	{10,3,6,12}	*NULL*	2	45

(c) Shares at CSP_1

ProNo	ProName	ProDescr	CategoryID	UnitPrice
125	{6,5,4,5}	*NULL*	2	5
126	{2,6,11,10}	*NULL*	6	8

(d) Shares at CSP_2

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	{6,6,5,7,9}	{12,8,1}	4	7
125	{6,5,8,3}	*NULL*	9	11
127	{7,2,9}	*NULL*	5	2

(e) Shares at CSP_3

ProNo	ProName	ProDescr	CategoryID	UnitPrice
125	{9,15,13,8}	*NULL*	12	7
126	{2,7,6,9}	*NULL*	12	1
127	{13,8,3}	*NULL*	4	9

(f) Shares at CSP_4

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	{5,9,11,1,5}	{10,6,7}	8	13
127	{7,17,4}	*NULL*	2	9

(g) Shares at CSP_5

Figure 8: Example of sharing new data

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, >, ≥, <, ≤, 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.

$$AVG(X) = DC(SUM(X))/DC(COUNT(X)) \quad (4)$$

$$VAR(X) = \frac{DC(SUM(X^2))}{DC(COUNT(X))} + \frac{DC(SUM(X))^2}{DC(COUNT(X))^2} \quad (5)$$

$$STDDEV(X) = \sqrt{\frac{DC(SUM(X^2))}{DC(COUNT(X))} + \frac{DC(SUM(X))^2}{DC(COUNT(X))^2}} \quad (6)$$

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_i 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, at one index server and at CSP_1 (query processing at other index servers and CSPs is similar). For clarity, the user, index server and CSP are denoted U, IS and CSP, respectively in Figure 9.

Finally, our approach directly supports all basic OLAP operations by directly querying cloud cubes and reconstructing the global result. For example, the total price and the number of products per year can be queried from Cube-I by query:

"SELECT YearID, YearName, TotalPrice, Number
FROM Cube-I, year
WHERE Cube-I.YearID = year.YearID AND MonthID IS NULL AND DateID IS NULL AND CategoryID IS NULL AND Prod-No IS NULL".

To drill down to total price and number of products per month in 2014, the previous query becomes:

"SELECT YearID, YearName, *Month*, TotalPrice, Number
FROM Cube-I, year, month
WHERE Cube-I.YearID = year.YearID *AND Cube-I.MonthID = Month.MonthID AND Cube-I.YearID = 2014* AND DateID IS NULL AND CategoryID IS NULL AND Prod-No IS NULL".

```
SELECT SUM(S.price+S.tax) AS sumprice, P.prodName
FROM Sale AS S JOIN Product AS P ON S.ProdNo=P.ProdNo
JOIN Date AS D ON S.DateKey=D.DateKey
WHERE D.Date BETWEEN '2014-01-01' AND '2014-01-15'
GROUP BY P.prodName
```

(a) Original query

1. (IS) Match DateKey in Type II index with condition D.Date BETWEEN '2014-01-01' AND '2014-01-15'. Let DK be the resulting set.
2. (U) Query-I is created to run at each CSP's:
   SELECT P.prodNo, P.prodName,
   SUM(S.price+S.tax) AS sumprice
   FROM Sale AS S JOIN Product AS P
   ON S.ProdNo=P.ProdNo
   WHERE S.Datekey IN (DK)
   GROUP BY P.ProdNo
3. (CSP) Execute Query-I. W.r.t. Figure 8, result is:
   $\{ \{124, \{6, 5, 3, 11, 7\}, 789\}, \{126, \{10, 3, 6, 12\}, 945\} \}$.
4. (U) Query-II is created to run on Type I index:
   SELECT prodNo, SUM(OrderNo) AS sumPK,
   FROM Sale
   WHERE Datekey IN (DK)
   GROUP BY ProdNo
5. (IS) Execute Query-II. Let us denote the results $sumPK_{pi}$, where $sumPK_{pi}$ is sumPK's value for product p and its CSP_i pseudo share.
6. (U) Reconstruct each record (prodName, sumprice) in the query result. For example, to reconstruct record prodNo=124:
   CSP_1 share of prodName is $\{6, 5, 3, 11, 7\}$ and
   CSP_1 aggregate of sumprice is
   $789 + 2 \times HE_2(sumPK_{(124)(1)}, ID_1)$ (Equation 3).

(b) Execution steps

Figure 9: Sample query rewriting

4. COMPARATIVE STUDY

In this section, we compare fVSS to the related approaches presented in Section 2, with respect to security and cost in the pay-as-you-go paradigm, global cost being customarily divided into storage, computing and data transfer costs. Table 2 synthesizes the features and complexities of all approaches, which we discuss below.

4.1 Data Security Features

By data security, we mean data privacy, availability and integrity. By design, all secret sharing-based approaches enforce privacy by guaranteeing shares cannot be decrypted by a single CSP or an intruder who would hack a CSP. Actually, a coalition or the compromise of at least t CSPs is necessary to break the secret. Privacy is further improved in fVSS, because data is not shared at all CSPs', but only $n - t + 2$. Thence, fVSS imposes a new constraint: no CSP *group* can hold enough shares to reconstruct the original data if $n < 2 \times t - 2$. Indeed, $n < 2 \times t - 2 \Leftrightarrow n - t + 2 < t$, i.e., the number of shares is lower than the number of shares necessary for reconstruction.

With respect to availability, all secret sharing-based approaches, still by design, allow reconstructing the secret, i.e., query shares, when $n - t$ CSPs fail. However, to the best of our knowledge, only fVSS allows updating shares in case

Table 2: Comparison of database sharing approaches

Features and costs	[1]	[2]	[8]	[9]	[11]	[12, 13]	[19]	[21]	fVSS
Data privacy	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Data availability	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ability in case CSPs fail, to									
- Query shares	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
- Update shares	No	No	No	No	No	No	No	No	Yes
Data integrity									
- Inner code verifying	No	Yes	No	No	No	No	Yes	Yes	Yes
- Outer code verifying	No	Yes	No	No	No	No	No	No	Yes
Target	DBs	DWs	DWs	DBs	DBs	DBs	DBs	DBs	DWs
Data sources	Single	Single	Multi	Multi	Single	Single	Single	Single	Single
Data types	Positive integers	Integers, Reals, Characters, Strings, Dates, Booleans	Positive integers	Integers	Integers	Positive integers	Positive integers	Positive integers	Integers, Reals, Characters, Strings, Dates, Booleans
Shared data access									
- Updates	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
- Exact match queries	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
- Range queries	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes
- Aggregation queries on one attribute	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
- Aggregation queries on two attributes	No	No	No	No	No	No	No	No	Yes
- Grouping queries	No	Yes	No	No	No	No	No	No	Yes
Complexity									
- Data storage w.r.t. original data volume	$\geq n$	$\geq n/(t-1)$ + signatures	$\geq 2n$	$\geq 2n$	$\geq n$	$\geq n$ +1 (B++ tree)	$\geq 2n$ +1 (hash tree)	$\geq n/t$ $+n/t$ (B++ tree) + signatures	$\geq n-t+2$ +1 (B++ tree) + signatures
- Sharing time	$O(\sigma nt)$	$O(\sigma nt)$	$O(\sigma nt)$	$O(\sigma nt)$	$O(\sigma nt)$	$O(\sigma nt)$	$O(\sigma nt)$	$O(\max(\sigma \log \sigma, \sigma n))$	$O(\sigma t(n-t))$
- Reconstruction time	$O(\gamma t^2)$	$O(\gamma t^2)$	$O(\gamma t^2)$	$O(\gamma t^2)$	$O(\gamma t^2)$	$O(\gamma t^2)$	$O(\gamma t^2)$	$O(\gamma t)$	$O(\gamma t^2)$

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.

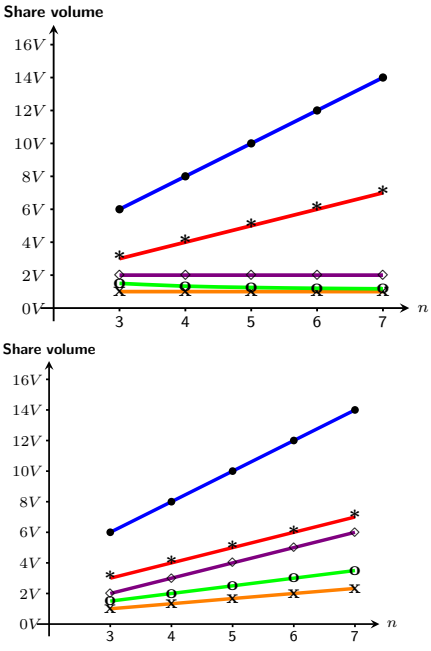
Table 3: CSP pricing policies

	CSP_1	CSP_2	CSP_3	CSP_4	CSP_5
Storage ($/GB/month)	0.030	0.040	0.053	0.120	0.325
sVM CPU time ($/h)	0.013	0.059	0.058	0.060	0.070
mVM CPU time ($/h)	0.026	0.079	0.115	0.120	0.140
lVM CPU time ($/h)	0.053	0.120	0.230	0.240	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_i times CSP_i'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 CSP_1, CSP_2 and CSP_3, 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 that with 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 ($)
	Global	per CSP	
Unencrypted data	100	100	3 to 32.5
[8, 9, 19]	1,000	200	113.60
[1, 11, 12, 13]	500	100	56.80
[2]	167	34	19.31
[21]	125	25	14.77
fVSS-I	300	60	34.08
fVSS-II	300	99.8 + 99.8 + 99.8 + 0.4 + 0.2	**12.39**



• *[8, 9, 19]* ∗ *[1, 11, 12, 13]* o *[2]* x *[21]* ◇ *fVSS*

Figure 10: Storage complexity comparison

4.3 Computing Cost

4.3.1 Sharing Cost

The sharing process time complexity (Table 2) depends on n, t and σ, which is the number of shared data pieces, i.e., individual attribute values. Since σ is normally much bigger than n and t, $\sigma \log \sigma > \sigma nt > \sigma t(n-t) > \sigma n$. Moreover, the number of CSPs where records are shared is $n-t+2$ in fVSS and n in all other approaches. If $t \geq 2$, which is quite probable, $n-t+2 \leq n$. Thus, we expect fVSS to be faster than previous approaches when sharing data.

Let us illustrate this through an example. Let $n = 5$, $t = 4$ and $\sigma = 10^{15}$. CSP pricing policies are depicted in Table 3, where sVM, mVM and lVM stand for small, medium and large virtual machine, respectively. Let us assume that the computing powers of sVMs, mVMs and lVMs are 1×10^{10}, 2×10^{10} and 4×10^{10} records per second, respectively. Virtual machine size is assigned to each CSP with respect to the number of records to share at that CSP.

Table 5 features a sharing cost comparison of all studied approaches but [21]. Sharing time is the number of processed records divided by the virtual machine's power. CPU cost is the sum of sharing times at each CSP_i times CSP_i's computing price. Results show that both fVSS-I and fVSS-II have a cheapest sharing process than all existing approaches, even though fVSS-I bears the longer sharing time. They also outline again the advantage of unbalancing the volume of shares at CSPs, which helps decrease cost by a factor 2.3 with respect to state-of-the-art approaches in this example.

4.3.2 Data Access Cost

Query response time, which is critical in an OLAP context, directly depends on the reconstruction process time complexity, which in turn depends on t and the number of records in the query response, γ (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 fVSS 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 $RG = \{CSP_1, CSP_2, CSP_4, CSP_5\}$. 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_i times CSP_i's computing price. Results show again that, even though response time is comparable for all approaches, fVSS allows much lower costs, especially when unbalancing the volume of shares at CSPs in fVSS-II, which helps decrease cost by a factor 4 with respect to state-of-the-art approaches in this example.

Table 5: Sharing cost comparison

Approach	#records at each CSP	VM type	Sharing time (h:mm)	CPU cost ($)
Unencrypted data at 1 CSP	10^{15}	lVM	6:57	0.36 to 1.94
[1, 2, 8, 9, 11, 12, 13, 19]	10^{15}	lVM	6:57	6.40
fVSS-I	6×10^{14}	mVM	8:20	4.40
fVSS-II	9.98×10^{14}	lVM	6:56	**2.80**
	9.98×10^{14}	lVM	6:56	
	9.98×10^{14}	lVM	6:56 ⟸ 6:56	
	4×10^{12}	sVM	0:07	
	2×10^{12}	sVM	0:04	

Table 6: Data access cost comparison

Approach	#records at each CSP	VM type	Response time (h:mm)	CPU cost ($)
Unencrypted data at 1 CSP	10^{14}	lVM	0:42	0.04 to 0.20
[1, 2, 8, 9, 11, 12, 13, 19]	10^{14}	lVM	0:42	0.48
fVSS-I	6×10^{13}	mVM	0:50	0.30
fVSS-II	9.98×10^{13}	lVM	0:42	**0.12**
	9.98×10^{13}	lVM	0:42	
	0	—	0:42 ⟸ 0:00	
	4×10^{11}	sVM	0:01	
	2×10^{11}	sVM	0:01	

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, fVSS 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

[1] D. Agrawal, A. E. Abbadi, F. Emekci, and A. Metwally. Database management as a service: challenges and opportunities. In *25th IEEE International Conference on Data Engineering (ICDE 2009), Shanghai, China*, pages 1709–1716, 2009.

[2] V. Attasena, N. Harbi, and J. Darmont. A novel multi-secret sharing approach for secure data warehousing and on-line analysis processing in the cloud. *International Journal of Data Warehousing and Mining*, 2014. (To appear).

[3] A. Beimel. Secret-Sharing Schemes: A Survey. In *3rd International Conference on Coding and Cryptology (IWCC 2011), Qingdao, China*, pages 11–46, 2011.

[4] S. Bu and R. Yang. Novel and Effective Multi-Secret Sharing Scheme. In *2nd International Conference on Information Engineering and Applications (IEA 2012), Dalian, China*, pages 461–467, 2012.

[5] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control. In *1st ACM Cloud Computing Security Workshop (CCSW 2009), Chicago, USA*, pages 85–90, 2009.

[6] P. J. Davis. *Interpolation and Approximation*. Dover, 1975.

[7] D. E. R. Denning. *Cryptography and data security*. Addison-Wesley, 1982.

[8] F. Emekci, D. Agrawal, and A. E. Abbadi. ABACUS: A Distributed Middleware for Privacy Preserving Data Sharing Across Private Data Warehouses. In *6th International Conference on Middleware (USENIX 2005), Grenoble, France*, pages 21–41, 2005.

[9] F. Emekci, D. Agrawal, A. E. Abbadi, and A. Gulbeden. Privacy preserving query processing using third parties. In *22nd IEEE International Conference on Data Engineering (ICDE 2006), Atlanta, USA*, pages 27–37, 2006.

[10] Z. Eslami and J. Z. Ahmadabadi. A Verifiable Multi-secret Sharing Scheme based on Cellular Automata. *Information Sciences*, 180(15):2889–2894, August 2010.

[11] M. A. Hadavi, E. Damiani, R. Jalili, S. Cimato, and Z. Ganjei. AS5: A secure searchable secret sharing scheme for privacy preserving database outsourcing. In *ESORICS DPM/SETOP 2012 International Workshops, Pisa, Italy*, pages 201–216, 2012.

[12] M. A. Hadavi and R. Jalili. Secure data outsourcing based on threshold secret sharing: towards a more practical solution. In *VLDB 2010 PhD Workshop, Singapore*, pages 54–59, 2010.

[13] M. A. Hadavi, M. Noferesti, R. Jalili, and E. Damiani. Database as a service: towards a unified solution for security requirements. In *36th IEEE Annual Conference on Computer Software and Applications Conference Workshops (COMPSACW 2012), Izmir, Turkey*, pages 415–420, 2012.

[14] Y.-X. Liu, L. Harn, C.-N. Yang, and Y.-Q. Zhang. Efficient (n, t, n) secret sharing schemes. *Journal of Systems and Software*, 85(6):1325–1332, January 2012.

[15] R. R. Ravan, N. B. Idris, and Z. Mehrabani. A Survey on Querying Encrypted Data for Database as a Service. In *5th International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC 2013), Beijing, China*, pages 14–18, 2013.

[16] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[17] B. ShanYue and Z. Hong. A Secret Sharing Scheme Based on NTRU Algorithm. In *5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom 2009), Beijing, China*, pages 1–4, 2009.

[18] R. Sion. *Towards Secure Data Outsourcing*, pages 137–161. Handbook of Database Security. Springer, 2008.

[19] B. Thompson, S. Haber, W. G. Horne, T. Sander, and D. Yao. Privacy-Preserving Computation and Verification of Aggregate Queries on Outsourced Databases. In *9th International Symposium on Privacy Enhancing Technologies (PETS 2009), Seattle, USA*, pages 185–201, 2009.

[20] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, March 2013.

[21] S. Wang, D. Agrawal, and A. E. Abbadi. A comprehensive framework for secure query processing on relational data in the cloud. In *8th VLDB International Conference on Secure Data Management (SDM 2011), Berlin, Germany*, pages 52–69, 2011.

[22] A. Waseda and M. Soshi. Consideration for Multi-threshold Multi-secret Sharing Schemes. In *2012 International Symposium on Information Theory and its Applications (ISITA 2012), Honolulu, USA*, pages 265–269, 2012.