Université Lumière Lyon 2
École Doctorale Informatique et Mathématiques

THÈSE

pour obtenir le grade de

Docteur en Informatique

présentée par

Varunya Attasena

Secret Sharing Approaches for Secure Data Warehousing and On-line Analysis in the Cloud

(Approches de partage de clés secrètes pour la sécurisation des entrepôts de données et de l'analyse en ligne dans le nuage)

préparée au sein du laboratoire

sous la direction de

Mme Nouria Harbi et **M. Jérôme Darmont**

Soutenue publiquement le 22 Septembre 2015 devant le jury:

Mme Salima Benbernou	Rapporteur	(PR, Université Paris Descartes)
M. Patrick Marcel	Rapporteur	(MDF, Université de Tours)
Mme Lynda Mokdad	Examinateur	(PR, Université Paris-Est, Créteil)
Mme Nouria Harbi	Directrice de thèse	(MCF, Université Lumière Lyon 2)
M. Jérôme Darmont	Directeur de thèse	(PR, Université Lumière Lyon 2)

Abstract

Cloud business intelligence is an increasingly popular solution 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. Many security issues are raised by data storage in a public cloud, including data privacy, data availability, data integrity, data backup and recovery, and data transfer safety. Moreover, security risks may come from both cloud service providers and intruders, while cloud data warehouses should be both highly protected and effectively refreshed and analyzed through on-line analysis processing. Hence, users seek secure data warehouses at the lowest possible storage and access costs within the pay-as-you-go paradigm.

In this thesis, we propose two novel approaches for securing cloud data warehouses by base-p verifiable secret sharing (bpVSS) and flexible verifiable secret sharing (fVSS), respectively. Secret sharing encrypts and distributes data over several cloud service providers, thus enforcing data privacy and availability. bpVSS and fVSS address five shortcomings in existing secret sharing-based approaches. First, they allow on-line analysis processing. Second, they enforce data integrity with the help of *both* inner and outer signatures. Third, they help users minimize the cost of cloud warehousing by limiting global share volume. Moreover, fVSS balances the load among service providers with respect to their pricing policies. Fourth, fVSS improves secret sharing security by imposing a new constraint: no cloud service provide *group* can hold enough shares to reconstruct or break the secret. Five, fVSS allows refreshing the data warehouse even when some service providers fail.

To evaluate bpVSS' and fVSS' efficiency, we theoretically study the factors that impact our approaches with respect to security, complexity and monetary cost in the pay-as-you-go paradigm. Moreover, we also validate the relevance of our approaches experimentally with the Star Schema Benchmark and demonstrate its superiority to related, existing methods.

Keywords: Data warehouses, On-line analysis processing (OLAP), Cloud computing, Secret sharing, Data privacy, Data availability, Data integrity

Résumé

Les systèmes d'information décisionnels dans le cloud Computing sont des solutions de plus en plus répandues. En effet, ces dernières offrent des capacités pour l'aide à la décision via l'élasticité des ressources pay-per-use du Cloud. Toutefois, les questions de sécurité des données demeurent une des principales préoccupations notamment lorsqu'il s'agit de traiter des données sensibles de l'entreprise. Beaucoup de questions de sécurité sont soulevées en terme de stockage, de protection, de disponibilité, d'intégrité, de sauvegarde et de récupération des données ainsi que des transferts des données dans un Cloud public. Les risques de sécurité peuvent provenir non seulement des fournisseurs de services de cloud computing mais aussi d'intrus malveillants. Les entrepôts de données dans les nuages devraient contenir des données sécurisées afin de permettre à la fois le traitement d'analyse en ligne hautement protégé et efficacement rafraîchi. Et ceci à plus faibles coûts de stockage et d'accès avec le modèle de paiement à la demande.

Dans cette thèse, nous proposons deux nouvelles approches pour la sécurisation des entrepôts de données dans les nuages basées respectivement sur le partage vérifiable de clé secrète (bpVSS) et le partage vérifiable et flexible de clé secrète (fVSS). L'objectif du partage de clé cryptée et la distribution des données auprès de plusieurs fournisseurs du cloud permet de garantir la confidentialité et la disponibilité des données. bpVSS et fVSS abordent cinq lacunes des approches existantes traitant de partage de clés secrètes. Tout d'abord, ils permettent le traitement de l'analyse en ligne. Deuxièmement, ils garantissent l'intégrité des données à l'aide de deux signatures interne et externe. Troisièmement, ils aident les utilisateurs à minimiser le coût de l'entreposage du cloud en limitant le volume global de données cryptées. Sachant que fVSS fait la répartition des volumes des données cryptées en fonction des tarifs des fournisseurs. Quatrièmement, fVSS améliore la sécurité basée sur le partage de clé secrète en imposant une nouvelle contrainte: aucun groupe de fournisseurs de service ne peut contenir suffisamment de volume de données cryptées pour reconstruire ou casser le secret. Et cinquièmement, fVSS permet l'actualisation de l'entrepôt de données, même si certains fournisseurs de services sont défaillants.

Pour évaluer l'efficacité de bpVSS et fVSS, nous étudions théoriquement les facteurs qui influent sur nos approches en matière de sécurité, de complexité et de coût financier dans le modèle de paiement à la demande. Nous validons également expérimentalement la pertinence de nos approches avec le Benchmark schéma en étoile afin de démontrer son efficacité par rapport aux méthodes existantes.

Mots-clés: Entrepôts de données, Analyse en ligne (OLAP), Infonuagique, Partage de clés secrètes, Confidentialité des données, Disponibilité des données, Intégrité des données

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisors Mr. Jérôme Darmont and Mrs. Nouria Harbi for their wholehearted support and encouragement during my PhD research time. Under their excellent supervision, I successfully overcame many difficulties and obstacles I encountered during the project's life cycle. Their constructive advice greatly inspired me with the way of thinking and developing research ideas. I am tremendously thankful for their precious time, effort, patience and willingness in helping me to fulfill this thesis.

I am extremely grateful to Mrs. Salima Benbernou and Mr. Patrick Marcel for their expertise in reviewing and giving constructive feedback to this thesis. Equally, my great honors and particular thanks go to Mrs. Lynda Mokdad for accepting to be the jury examiners.

I take this opportunity to sincerely acknowledge all members of the SID team, Mr. Omar Boussaid, Mrs. Fadila Bentayeb, Mrs. Sabine Loudcher, Ms. Cecile Favre, Mrs. Nadia Kabachi and Mr. Gérald Gavin, for their valuable suggestions and encouragements in this research work.

I would like to show my profound gratitude to the former secretary, secretary and technician of the ERIC Lab, Mrs. Valerie Gabriele, Mrs. Habiba Osman and Mr. Julien Crevel, for their enormous help and enthusiastic support in administrative procedures and studying facilities. Thanks to all colleagues at the ERIC Lab for supporting and sharing with me memorable moments.

Last but not least, I am deeply indebted to my parents, my brother, my husband's parents and all my relatives for their encouragement and support during the research. I affectionately thank my husband, Thitipong Satiramatekul, for his understanding and encouragement. His constant love has sustained me throughout my life.

Varunya Attasena
Lyon, July 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Business intelligence (BI) has been an ever-growing trend for more than twenty years, but the recent advent of cloud computing now allows deploying data analytics even more easily. While building a traditional BI system typically necessitates an important initial investment, with the cloud pay-as-you-go model, users can punctually devote small amounts of resources in return for a one-time advantage. This trend is currently supported by numerous "BI as a service" offerings, with high economic stakes.

Although cloud computing is currently booming, data security remains one of the top concerns for cloud users and would-be users. 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 or subcontractor espionage, cost-effective defense of availability and uncontrolled mashups [2–4]. In the particular context of cloud BI, privacy is of critical importance. Up to now, security issues have been handled by cloud service providers (CSPs). But with the multiplication of CSPs and subcontractors in many different countries, intricate legal issues arise, as well as another fundamental issue: *trust*. Telling whether trust should be placed in CSPs eventually falls back onto end-users, with the implied costs.

Critical security concerns in (especially public) cloud storage are depicted in Figure 1.1. User data might be deleted, lost or damaged because of several probable reasons. First, some CSPs have the policy of taking the highest profit. Therefore, unmodified or unaccessed data may be deleted to serve other customers. Second, data loss may also be caused by accidental, e.g., electrical or network failure, or intentional plans,

e.g., maintenance or system backup. Moreover, virtual cloud architectures might not be sufficiently safeguarded from inside attacks. Finally, all CSPs cannot guarantee 100% data availability, although some cloud businesses must run on a 7/24 basis. Thus, data privacy, availability and integrity are the major issues in cloud data security.

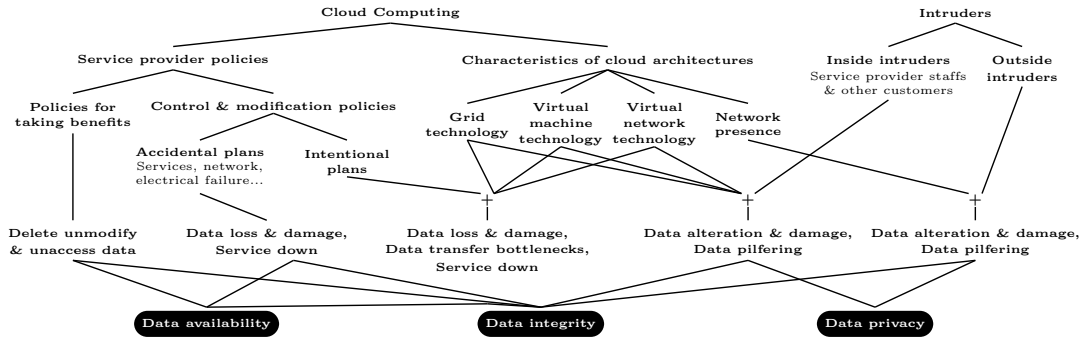


FIGURE 1.1: Cloud data security issues

In the context of cloud BI, cloud data warehouses (DWs) must not only be highly protected, but also effectively refreshed and analyzed through on-line analysis processing (OLAP). Thence, while CSPs must optimize service quality and profit, users seek to reduce storage and access costs within the pay-as-you-go paradigm. Thus, in cloud DWs, the tradeoff between data security and large-scale OLAP analysis poses a great challenge [4, 5].

1.2 Cloud Computing and Security

Security is the top concern when sensitive data are shared and processed in the cloud. Existing research notably addresses three security issues (data privacy, availability and integrity) through encryption, anonymization, replication and verification (Figure 1.2).

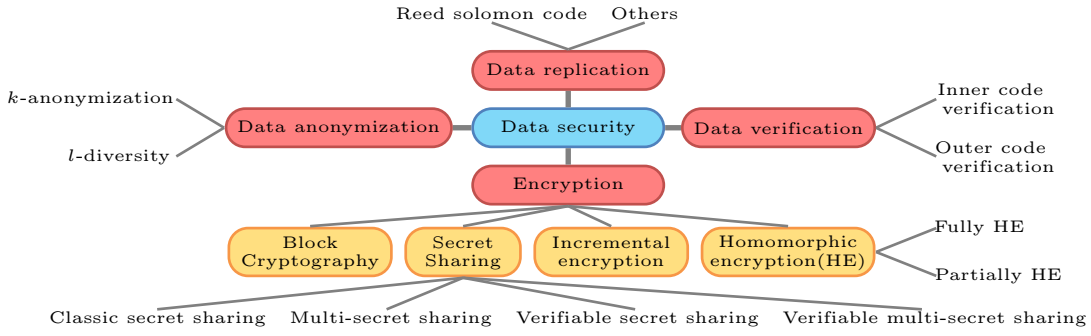


FIGURE 1.2: Existing data security solutions

This section aims at providing a general overview of these data security methods, as background information. Some of them are also exploited in our own proposals.

1.2.1 Encryption

Encryption is a process that turn data into an unreadable cipher-text (encrypted data) [6]. Thus, it mainly addresses data privacy. Moreover, secret sharing, which is a well-known encryption scheme, also handles data availability by design. Similarly, classical (e.g., block cryptography) and modern encryption (e.g., homomorphic encryption, incremental encryption) provide data privacy while minimizing the cost of data access and/or update.

1.2.1.1 Block Cryptography

Block cryptography is a type of symmetric key cryptography [6]. It encrypts a data block, which is a data set or a binary string, with a symmetric key. Sizes of both data blocks and encrypted data blocks are fixed. Block cryptography always encrypts a data block the same way, given the same key is used. Hence, matching functions can be run on encrypted data blocks. However, security drops if block size is too small and block values are not variable enough, because patterns may be extracted to break encryption.

Several research approaches apply block cryptography for searching words in large encrypted documents [7–10]. Only a few secure database (DB) systems exploit block cryptography to enforce privacy [11–13]. However, they do not handle availability nor integrity issues, which are also very important in DB context. Block cryptography allows exact match queries on encrypted DBs. Then, only query results are transferred back to the user. However, results may include false positives because encryption functions are non-injective. Minimizing false negative rate is still a challenge for block cryptography.

1.2.1.2 Homomorphic Encryption

Homomorphic encryption allows mathematical operations to be performed on encrypted data without decrypting data first [6, 14, 15]. There are two types of homomorphic encryption: partially and fully homomorphic encryption. Partially homomorphic encryption allows only one operation, e.g., addition or multiplication, whereas fully homomorphic encryption supports several operations (e.g., $A + B$, $B \times C$), but still does not allow more complex operations (e.g., $A + B \times C$).

Several secure DB approaches exploit homomorphic encryption [11–13, 16, 17]. Thanks to the homomorphic property, aggregation (e.g., SUM, AVG) queries can be run on encrypted DBs, outputting encrypted aggregates to the user, without the necessity to decrypt the data. This helps optimize global query response time and reduce bandwidth.

Moreover, computation cost at the user's side is also minimized. However, only a few homomorphic encryption approaches can support data integrity, and none supports data availability.

1.2.1.3 Incremental Encryption

Incremental encryption supports efficient updates directly on encrypted data [18, 19]. For example, let data piece d be encrypted with function $e = f(d, k)$, where e and k are an encrypted data piece and a key, respectively. When d is modified into $d + \delta$, its new encryption can be computed with function $e_{new} = g(e, \delta, k)$ without decrypting d. Some secure storage and DB approaches exploit incremental encryption [20, 21]. However, they do not support data availability nor integrity.

1.2.1.4 Secret Sharing

Secret sharing distributes individually meaningless shares of data to n participants to enforce privacy [1, 22–29]. A subset of $t \leq n$ participants is required to reconstruct the secret. Moreover, up to $n - t$ participants may disappear without compromising data availability. The drawback of this solution is the multiplication of the initial data volume by the number of participants. Modern secret sharing schemes, such as multi secret sharing [29–34], verifiable secret sharing [35–37], and verifiable multi secret sharing [38–53], help reduce the volume of shares, verify the honesty of each participant, and both, respectively.

Several secure distributed DB approaches exploit secret sharing for data security and availability [21, 54–60]. Moreover, three approaches also support data integrity [21, 58, 60]. Although almost approaches can perform queries on encrypted DBs thanks to the homomorphic property, large data volumes remain a challenge and costs for analyzing, transferring and storing encrypted DBs are expensive.

1.2.2 Data Anonymization

Data anonymization is also used to enforce data privacy [61–68]. In a DB, only keys or sensitive information are protected, whereas the schema and other information are not [68]. Thus, data anonymization straightforwardly allows data querying. There are several models, e.g., k-anonymity, l-diversity, and techniques, e.g., hashing, hiding, permutation, shift (Table 1.1), to protect keys and sensitive information, respectively. For example, k-anonymity transforms k distinguishable records into k indistinguishable

TABLE 1.1: Anonymization techniques

Technique	**Example**			
	Name	**Phone Number**	**Salary**	**Description**
	Bob	01 23 45 67 89	2500 €	**Sample data**
	Alice	04 78 55 12 34	3000 €	
Suppression	Bob	01 23 45 67 89	**0 €**	The salary field is hidden or salary is replaced with a constant value.
	Alice	04 78 55 12 34	**0 €**	
Generalization	Bob	01 23 45 67 89	**2*** €**	Salary is coarsened into sets.
	Alice	04 78 55 12 34	**3*** €**	
Perturbation	Bob	01 23 45 67 89	**25050 €**	Noise is added to salary.
	Alice	04 78 55 12 34	**30030 €**	
Hashing	**683452353981**	01 23 45 67 89	2500 €	Name is hashed into a single fixed-length number
	234951034293	04 78 55 12 34	3000 €	
Permutation	**Jones**	01 23 45 67 89	2500 €	Name is mapped to new values.
	Smith	04 78 55 12 34	3000 €	
Shift	Bob	01 23 45 67 89	**3000 €**	500 € is added to salary values.
	Alice	04 78 55 12 34	**3500 €**	
Enumeration	Bob	01 23 45 67 89	**2700 €**	Salary is changed, but the relative order of salaries is preserved.
	Alice	04 78 55 12 34	**3500 €**	
Truncation	Bob	**01**	2500 €	Phone number is truncated after code area, hiding actual location.
	Alice	**04**	3000 €	
Prefix-preserving	Bob	**01 23 45 23 56**	2500 €	Phone number is changed but some prefix bits are preserved.
	Alice	**04 78 55 45 32**	3000 €	

records [63–66]. Hashing or permutation techniques help hide values in attributes, e.g., NAME in Table 1.1. While cheap when accessing data, anonymization is not strong enough to protect against attacks such as homogeneity and background knowledge attacks [68], and is not designed to address data availability and integrity issues.

1.2.3 Data Replication

Data replication is the process of copying some or all data from one location to one or several others [20, 69–73]. Its main purposes are to improve availability, fault-tolerance and/or accessibility. A well-known data replication scheme is Reed Solomon (RS) code [70, 71], which is quite similar to secret sharing. RS code indeed distributes data amongst a group of participants and can reconstruct data even if some participants disappear, thus enforcing availability. RS code and secret sharing mostly differ in their driving goals, i.e., availability and privacy, respectively.

1.2.4 Data Verification

Data verification is the process of checking data integrity, by verifying data corruption caused by either accidents or intruder attacks, with the help of signatures, e.g., digital signature, message authentication or fingerprint [74–80]. However, since signature sizes are small with respect to data size and signature creation typically involves

random or hash functions, they cannot guarantee 100% data correctness. Moreover, so-called outer code verifying methods allow checking encrypted data without decrypting them first [75–80]. Only few secure DB approaches exploit inner code verification to guarantee data integrity [21, 58, 60].

1.3 Thesis Motivation and Contributions

Table 1.2 summarizes the features of the security approaches introduced in Section 1.2, with respect to data privacy, availability, integrity and access. No existing approach simultaneously satisfies all criteria. Only verifiable (single/multi) secret sharing simultaneously enforces all security issues (data privacy, availability and integrity). However, storing/updating/accessing data with verifiable secret sharing turns to be difficult and expensive because these approaches replicate data n times and do not allow accessing encrypted data.

TABLE 1.2: Comparison of data security solutions

Approaches	**Data privacy**	**Data availability**	**Data integrity**	**Data access**
Encryption				
- Block cryptography	√			Exact match query on encrypted data
- Homomorphic cryptography	√			Aggregate query on encrypted data
- Incremental cryptography	√			Incremental update encrypted data
- Classic secret sharing	√	√		Aggregate query on encrypted data
- Multi-secret sharing	√	√		
- Verifiable secret sharing	√	√	√	
- Verifiable multi-secret sharing	√	√	√	
Data anonymization	√			On non-anonymized data
Data replication(RS code)	√	√		
Data verification				
- Inner code verification			√	
- Outer code verification			√	

All secret sharing-based approaches guarantee encrypted data cannot be decrypted by a single CSP, nor any intruder who would hack a CSP. However, a coalition or the compromission of at least t CSPs breaks the secret. For data availability, all 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 DBs' refreshment capabilities. Moreover, although secret sharing approaches feature all basic DB querying operators, none handles OLAP. In addition, few approaches actually enforce data integrity, through verification of only inner code (to verify whether CSPs are malicious). Finally, only one approach brings in solutions to reduce overall storage volume so that it falls well under n times that of original data, and thus decrease monetary storage cost in the pay-as-you-go paradigm. However, its data-access cost is still high because queried data must be wholly reconstructed beforehand.

To address all these issues, we propose two novel approaches that rely on a base-p verifiable secret sharing (bpVSS) and a flexible verifiable secret sharing scheme (fVSS). To the best of our knowledge, bpVSS and fVSS are first secret sharing-based approaches that allow running OLAP operators on shared DWs or cubes without reconstructing all data first, while minimizing global share volume to less than n times that of original data. They also feature both inner (to verify whether CSPs are malicious) and outer (to detect incorrect data before decryption) signatures for data verification. Moreover, fVSS is the first approach that guarantees no CSP *group* can hold enough shares to reconstruct the secret. fVSS also allows DW refreshment when one or several CSPs fail, and allows users adjusting the volume of shared data at each CSP's, which helps optimize cost with respect to various CSP pricing policies.

1.4 Dissertation Organization

The remainder of this thesis is structured as follows. Chapter 2 provides a survey of secret sharing techniques, which are the security approaches most related to ours. We notably discuss security and data-access features, as well as complexity.

Chapters 3 and 4 present our secure DB approaches (bpVSS and fVSS, respectively). These chapters first introduce the principle of our approaches. Then, they detail our secret sharing and verification mechanisms, the new outer signatures we propose and how our approaches apply to data warehouses and OLAP.

Chapter 5 describes a mainly theoretical study of our approaches along three axes: security, performance and monetary cost in the pay-as-you-go paradigm. We particularly discuss the tradeoffs between security and performance in our approaches.

Chapter 6 aims at experimentally validate our secret sharing schemes by comparing them to state-of-the-art approaches with respect to shared data volume, query response time and transferred data volume.

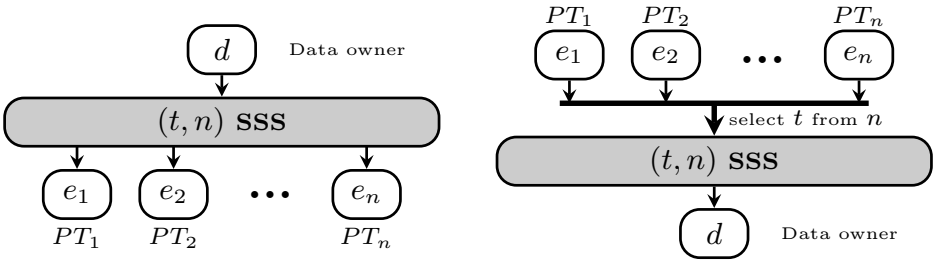
Finally, Chapter 7 concludes this dissertation, by discussing our contributions and highlighting future research perspectives.

Chapter 2

Secret Sharing for Databases

In this chapter, we extensively survey secret sharing approaches, as well as SSS-based approaches that allow sharing DBs.

SSSs are primarily aimed at enforcing privacy. Individual pieces of secret data d are each encrypted into n so-called shares $\{e_i\}_{i=0,\cdots,n}$, each share e_i being stored by a different participant (PT) PT_i (Figure 2.1(a)). Each share e_i is meaningless to PT_i. A subset of $t \leq n$ PTs is required to reconstruct the secret (Figure 2.1(b)). Thence, a convenient side effect of SSSs is data availability, since up to $n - t$ PTs may disappear without preventing secret reconstruction. Classical SSSs [1, 23–29, 81] mainly differ by their encryption methods, which bear different security properties with different data storage and CPU requirements.

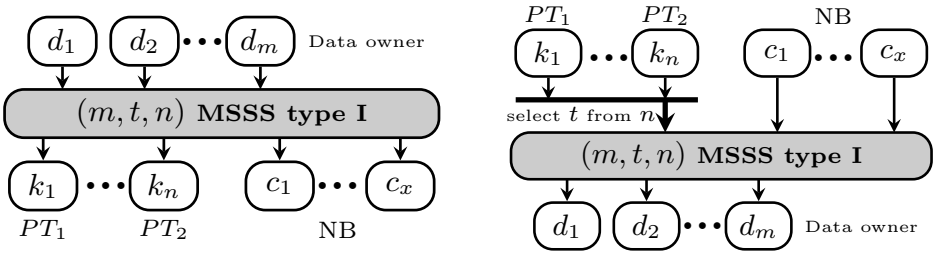


(a) Sharing process (b) Reconstruction process

FIGURE 2.1: Classical secret sharing

A major drawback of initial SSSs is the multiplication of the initial data volume by the number of PTs. Multi secret sharing schemes (MSSSs) thus aim to reduce computation, storage and data transfer costs by sharing and reconstructing more than one secret at once. Some MSSSs achieve an overall shared data volume (i.e., at all PTs') that is close to that of original secret data. We categorize MSSSs into two types.

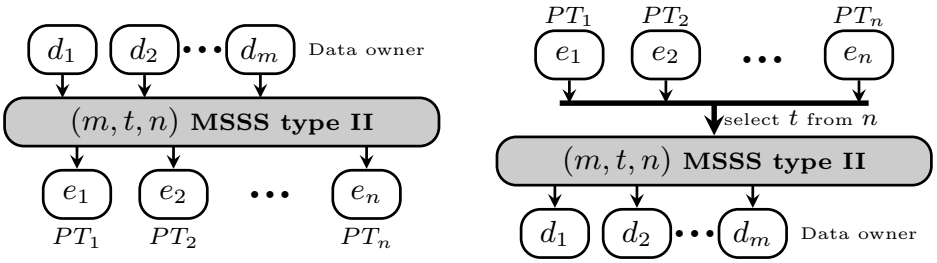
In MSSSs type I [30, 32], data are encrypted with the help of keys. m secrets $\{d_j\}_{j=1,\cdots,m}$ and n keys $\{k_i\}_{i=1,\cdots,n}$ are used to construct x encrypted pieces of data $\{c_h\}_{h=1,\cdots,x}$, where $m \leq x$. Encrypted data are stored in a news bulletin board (NB), whereas each key k_i is stored at PT_i (Figure 2.2(a)). To reconstruct the m secrets, all or some (depending on the MSSS) shares and t keys are used (Figure 2.2(b)).



(a) Sharing process (b) Reconstruction process

FIGURE 2.2: Multiple secret sharing type I

In MSSSs type II [29, 31, 33, 34], m secrets $\{d_j\}_{j=1,\cdots,m}$ are encrypted into n shares $\{e_i\}_{i=1,\cdots,n}$, where $m \leq t \leq n$. In case $m > t$, data are first organized into blocks that are fewer than t. Then, each data block is encrypted at once. Finally each share e_i is stored by PT_i (Figure 2.3(a)). As in SSSs, reconstructing the secrets requires t PTs (Figure 2.3(b)).



(a) Sharing process (b) Reconstruction process

FIGURE 2.3: Multiple secret sharing type II

SSSs and MSSSs assume that all players, i.e., PTs and NB, are honest and always provide valid information (data and keys). However, in reality, they might not, intentionally or not. Thus, verifiable secret sharing schemes (VSSSs) [35–37, 82, 83] and verifiable multi secret sharing schemes (VMSSSs) [38–53] verify the correctness of data and/or keys before or after reconstruction. Therefore, VSSSs and VMSSSs enforce data integrity in addition to privacy and availability.

Eventually, some SSSs aim at specific goals. The proactive secret sharing scheme (PSSS) can update shares and add in a new PT without impacting existing shares [84]. The weighted secret sharing scheme (WSSS) introduces a priority among PTs by

assigning to each PT a weight, i.e., the number of shares it stores [85]. Thus, for instance, only two high priority PTs could reconstruct the secret if the sum of their weights is greater or equal to some threshold related to t, whereas two lower priority PTs could not. The social secret sharing scheme (SSSS) extend from WSSS by allowing weights to be adjusted depending on the situation, e.g., if some PTs are found insincere [86–89]. Even though PSSS, WSSS and SSSSs bring in a more flexible PT management, they induce a higher share volume, i.e., at least n times the original data volume *vs.* at most n times for previous SSSs, supposing that individual shares use up the same volume as the corresponding secrets. Thus, we do not survey them further.

Section 2.1 classifies SSSs into ten groups, whose properties and the costs they imply in the pay-as-you-go paradigm are thoroughly detailed. Moreover, we compare all SSSs with respect to data security, encrypted data querying, and storage and computing costs. Section 2.2 describes secure sharing-based approaches for sharing DBs and DWs. Moreover, their security, DB features and cost are discussed (Section 2.2.3).

2.1 Secret Sharing Schemes

We categorize other SSSs into ten groups (Table 2.1) with respect to their basic type, i.e., SSSs and MSSSs types I and II, as well as eventual data or key verifications. We survey all groups in the following subsections. Moreover, we introduce the parameters and notations used throughout this section in Table 2.2.

TABLE 2.1: Classification of secret sharing schemes

		SSSs	**MSSSs type I**	**MSSSs type II**	
Verification	**None**	**Group 1** [1, 23–28, 81], [29] SSS	**Group 2** [30, 32]	**Group 3** [31, 33, 34], [29] MSSS	**SSSs/MSSSs**
	Data	**Group 4** [35, 82, 83]	**Group 7** [38]		**VSSSs/VMSSSs**
	Keys	**Group 5** [36]	**Group 8** [39–42, 44, 50, 51],[43]-I&II		
	Data & keys	**Group 6** [37]	**Group 9** [45–49, 52]	**Group 10** [53]	

2.1.1 Group 1: Classical Secret Sharing Schemes

In the very first (t, n) SSS [23], a random polynomial (Equation 2.1) is generated over $\mathbb{Z}_p[i]$ such that coefficient c_0 is the secret and other coefficients $c_{u=1,\cdots,t-1}$ are random integers. Then, each share e_i is created by Equation 2.2 and stored at PT_i. A number $t \leq n$ of PTs can reconstruct the original polynomial by Lagrange interpolation

TABLE 2.2: Secret sharing schemes' parameters

Parameter	**Definition**
m	Number of data pieces
D	Secret data such that $D = \{d_1, \cdots, d_m\}$ and $D = \{b_1, \cdots, b_o\}$
d	Piece of secret data in integer format
$\|\|d\|\|$	Storage size of d
d_j	j^{th} element of D in integer format
n	Number of PTs
t	Number of shares necessary for reconstructing the secret
γ	Number of PTs of the first group in [34]
PT_i	PT number i
ID_i	Identifier of PT_i
g	Number of groups of PTs
G_r	r^{th} groups of PTs such that $G_r \subseteq \{PT_i\}_{i=1,\cdots,n}$ and $G_r = \{PT_{r,1}, \cdots, PT_{r,g}\}$
$PT_{r,v}$	PT number v of G_r
$ID_{r,v}$	Identifier of $PT_{r,v}$ of G_r
o	Number of data blocks
b_l	l^{th} block of D such that $b_l = \{d_{l,1}, \cdots, d_{l,t}\}$ with fixed-sized blocks and $b_l = \{d_{l,1}, \cdots, d_{l,t_l}\}$ with variable-sized blocks
t_l	Number of shares necessary for reconstructing the secret in b_l (in case of variable-sized blocks)
$d_{l,q}$	q^{th} element of b_l in integer format
e_i	Share stored at PT_i
$e_{j,i}$	j^{th} share stored at PT_i
$e_{l,i}$	share of b_l stored at PT_i
c_h	h^{th} encrypted data stored at the NB
$c_{j,h}$	h^{th} encrypted data of d_j stored at the NB
$c_{l,h}$	h^{th} encrypted data of b_l stored at the NB
$c_{l,q,h}$	h^{th} encrypted data of $d_{l,q}$ in b_l stored at the NB
$c_{r,l,h}$	h^{th} encrypted data of b_l from G_r stored at the NB
k_i	Key stored at PT_i
$k_{i,q}$	Key number q stored at PT_i
$k_{r,i}$	Key stored at PT_i of G_r
$\|\|k\|\|$	Storage size keys
s_d_i	Signature stored at PT_i
s_d_l	Signature of d_l
$s\_d_{l,i}$	Signature of b_l stored at PT_i
$s\_d_{l,q}$	Signature number q of b_l
s_k_i	Signature of PT_i's key
$s\_k_{r,v}$	Signature of $PT_{r,v}$'s key of G_r
$\|\|s\|\|$	Storage size of signatures
$p, p_1, p_2 \ldots$	Big prime numbers
$A, A_1, A_2 \ldots$	Matrices
$f, f_1, f_2 \ldots$	Functions
$H, H_1, H_2 \ldots$	Hash functions

over a finite field. A sample application of this scheme is given in Figure 2.4, where $t = 4$ and $n = 6$. The random polynomial of degree $t - 1 = 3$ is $e_i = f(i) = i^3 - 5i^2 + 2i + 4$, where 4 is the secret. The six shares $\{(i, e_i)\}_{i=1,\cdots,6}$ (plotted in blue) are (1,2), (2,-4), (3,-8), (4,-4), (5,14) and (6,52)

$$f(i) = \sum_{u=0}^{t-1} c_u \times i^u \quad (2.1)$$

$$e_i = f(i) \quad (2.2)$$

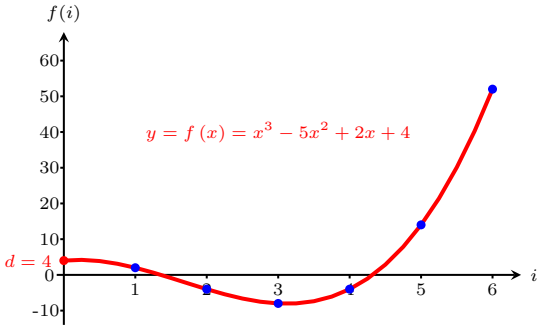
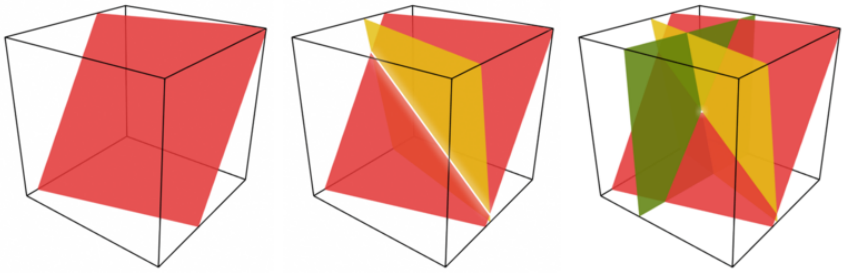


FIGURE 2.4: Secret sharing through polynomial interpolation

In [24], each PT is associated with an hyperplane in a t-dimensional space over a finite field. Hyperplanes, i.e., shares, intersect in a point that is the secret, which can be reconstructed by solving the hyperplanes' equation system. A sample application of this scheme is given in Figure 2.5, where $t = 2$ and $n = 3$ (there are thus three hyperplans).



from http://en.wikipedia.org/wiki/Secret_sharing

FIGURE 2.5: Secret sharing through hyperplan intersection

[25] exploits the Chinese remainder theorem. First, $n+1$ uniquely relatively primes[1] $\{p_i\}_{i=0,\cdots,n}$ are determined such that $p_0 < p_1 < \cdots < p_n$ and $\prod_{i=1}^{t} p_i > p_0 \prod_{i=1}^{t-1} p_{n-i+1}$. Then, n shares $\{e_l\}_{l=1,\cdots,n}$ are created by Equations 2.3 and 2.4, where u is a random positive integer. Finally, secret d is reconstructed from t shares by Equations 2.5 and 2.6.

[1] Uniquely relatively primes are random prime numbers that are related to each other by some conditions.

$$e_i = y \bmod p_i \tag{2.3}$$
$$y = d + u \times p_0 \tag{2.4}$$
$$d = y \bmod p_0 \tag{2.5}$$
$$y \equiv e_i \bmod m_i \tag{2.6}$$

All subsequent SSSs extend the three foundation schemes above. [26] extends from [25] to reduce the size of shares. Moreover, this SSS can reconstruct data from t or more shares, whereas previous schemes exploit exactly t shares. In the sharing process, the secret is split in t. Share creation from the t splits and secret reconstruction proceed as in [25]. All other SSSs seek to improve polynomial interpolation.

[81] extends from [23] to guarantee the t-consistency of shares. A random polynomial function $f(x)$ is created as in [23] ($f(0) = d$). However, $k_{i,1}$ and $k_{i,2}$ are random keys stored at PT_i and $k_{i,2}$ is do not need to be distinct from each other. Next, n shares $\{c_i\}_{i=1,\cdots,n}$ are created by Equation 2.7 and stored on the NB. Data piece d can be reconstructed by Lagrange interpolation from t pairs $\{k_{i,1}, c_i + k_{i,2}\}$.

$$c_i = f(k_{i,1}) - k_{i,2} \tag{2.7}$$

[1] proceeds in two steps. First, secret d is encrypted into t intermediate shares $\{u_v\}_{v=1,\cdots,t}$ by mapping d to the x-axis of a random polynomial. Second, these t shares are encrypted again into n actual shares $\{e_i\}_{i=1,\cdots,t}$ by Equation 2.8, where A_1 is an $n \times t$ random matrix. Secret d is reconstructed from a polynomial of degree t created by Equation 2.9. $\{u_v\}_{v=1,\cdots,t}$ are reconstructed by Equation 2.10, where A_2 is a $t \times t$ inverse matrix seeded from t rows of matrix A_1.

$$[e_1, \cdots, e_n]^T = A_1 \times [u_1, \cdots, u_t]^T \tag{2.8}$$
$$\prod_{a=1}^{t}(x - u_a) \equiv 0 \bmod p \tag{2.9}$$
$$[u_1, \cdots, u_t]^T = A_2 \times [e_1, \cdots, e_t]^T \tag{2.10}$$

The second step enforces availability and is optional. A sample application of the first step is given in Figure 2.6, where $d = 10$ and $t = 3$. The polynomial equation of degree 3 $(x - u_1)(x - u_2)(x - u_3) \equiv x^3 - 21x^2 + x - 10 \equiv 0 \bmod 31$ is created with the

help of prime $p = 31$ and random positive integers $u_1 = 19$, $u_2 = 22$ and $u_3 = 11$, where u_1, u_2, u_3 match with condition $u_3 \equiv d \times (u_1 \times u_2) \bmod p$.

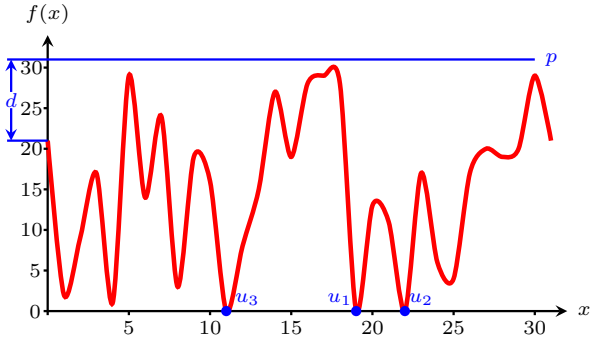


FIGURE 2.6: [1]'s secret mapping step

In [28], a data piece d is split into $t-1$ smaller data units $\{u_v\}_{v=1,\cdots,t-1}$ to reduce shared data volume. Then, a polynomial equation of degree $t-1$ is created by running recursive functions $t-1$ times (Equation 2.11, where y is a random integer) to improve security. Next, n shares $\{e_i\}_{i=1,\cdots,n}$ are created by Equation 2.12. Finally, data are reconstructed through $t-1$ steps by Lagrange interpolation.

$$f_v(x) = \begin{cases} u_v + y \times x & \text{if } v = 1 \\ u_v + \sum_{w=1}^{v} f_{v-1}(w) \times x^w & \text{otherwise} \end{cases} \tag{2.11}$$

$$e_i = f_{t-1}(i) \tag{2.12}$$

Eventually, [27, 29] extend from [82] (Section 2.1.4). However, none verifies the correctness of shares. In addtion, both approaches verify a strong t-consistency property. The verification processes guarantee that any subset of t shares or more (created by summing n random polynomial functions of degree $t-1$ in [82]) always reconstruct the same data, but that any subset of t shares or fewer cannot. Verification time is slower in [27] than in [29].

2.1.2 Group 2: Multi Secret Sharing Schemes Type I

The first (m, t, n) MSSS type I [30] extends from [23] to reduce shared data volume and execution times. All data are shared at once, with share volume being controlled to remain close to that of original data. To this aim, to share m data pieces $\{d_j\}_{j=1,\cdots,m}$ among n PTs, n keys $\{k_i\}_{i=1,\cdots,n}$ are created with a two-variables one-way function. Then, a polynomial (Equation 2.1) is created over $\mathbb{Z}_p[x]$ [23], but the degree of the polynomial is $w = \max(m, t) - 1$. Moreover, coefficients $\{u_j\}_{j=1,\cdots,m}$ are data pieces $\{d_j\}_{j=1,\cdots,m}$ and other coefficients $\{u_j\}_{j=(m+1),\cdots,t}$ are random integers. Next, $m+n-t$

shares $\{c_h\}_{h=1,\cdots,(m+n-t)}$ are generated by Equation 2.13 and are published on a NB. Finally, original data are reconstructed by Lagrange interpolation from t or more keys and w shares.

$$c_h = \begin{cases} f(k_h) & \text{if } 1 \leq h \leq n \\ f(H(h)) & \text{if } n+1 \leq h \leq n+m-t \end{cases} \tag{2.13}$$

[32] extends from [53] (Section 2.1.10) by reducing execution time and dynamically adjusting data block sizes. In the sharing process, data are organized into o unfixed size blocks $\{b_l\}_{l=1,\cdots,o}$. Data block b_l stores t_l data pieces $\{d_{l,q}\}_{q=1,\cdots,t_l}$. Next, keys k_i are randomly selected and matrix $A = [a_{x,y}]_{n\times\max(t_1,\cdots,t_o)}$ is created by Equation 2.14, where $l = 1,\cdots,o$, $u_{l,q}$ is a random integer and $A_l = [a_{x,y}]_{n\times t_l}$ such that A_l is made of the first t_l columns of A. Next, $o \times t_l$ shares $\{c_{l,h}\}_{l=1,\cdots,o;h=1,\cdots,t_l}$ are created by Equation 2.15, where v is a random integer. Finally, each key k_i is shared at PT_i and $\{c_{l,h}\}_{l=1,\cdots,o;h=1,\cdots,t_l}$, A and f_l are published on the NB. In the reconstruction process, $\{u_{l,q}\}_{q=1,\cdots,t_l}$ is created by solving Equation 2.14. Then, data are reconstructed by solving Equation 2.15.

$$[f_l(k_1),\cdots,f_l(k_n)]^T = A_l \times [u_{l,1},\cdots,u_{l,t_l}]^T \tag{2.14}$$

$$c_{l,h} = \sum_{q=1}^{t_l} d_{l,q} \times v^{(q-1)(\sum_{l=1}^{h-1} t_l + h - 1)} + \sum_{q=1}^{t_l} u_{l,q} \times v^{(t_l+q-1)(\sum_{l=1}^{h-1} t_l + h - 1t)} \tag{2.15}$$

2.1.3 Group 3: Multi Secret Sharing Schemes Type II

The first (m, t_m, n) MSSS type II [31] extends [23] to share m data pieces with different threshold access structures. In the sharing process, PT identifiers $\{ID_i\}_{i=1,\cdots,n}$ are randomly chosen from distinct integers. With respect to data piece d_j, t_j and a prime p_j are selected such that $t_1 \leq t_2 \leq \cdots \leq t_m$, $p_1 < p_2 < \cdots < p_m$, $P = \prod_{j=1}^{m} p_j$ and $d_j < p_j$. Next, a polynomial of degree $t_m - 1$ (Equation 2.16) is created with coefficients $\{u_v\}_{v=1,\cdots,t-1}$ being integers chosen by the Chinese remainder theorem and the uniqueness theorem of interpolating polynomial. $\forall_v \in [0, t-1], u_v \equiv u_{j,v} \bmod p_j \forall j = 1,\cdots,m$ where $u_{j,v}$ is a coefficient of a random polynomial function of degree $t_j - 1$ $f_j(x) = \sum_{w=0}^{t_j-1} u_{j,v} \times x^w$ [23]) and $u_{j,0} = d_j$. Shares $\{e_i\}_{i=1,\cdots,n}$ are generated by equation 2.17. Finally, ID_i and e_i are stored at PT_i, whereas $\{t_j\}_{j=1,\cdots,m}$ and $\{p_j\}_{j=1,\cdots,m}$ are retained at the user's. Finally, data piece d_j is reconstructed from p_j and t_j pairs (ID_i, e_i) by equations 2.18 and 2.19.

$$f(x) = \sum_{v=0}^{t_m-1} u_v \times x^v \tag{2.16}$$

$$e_i = f(ID_i) \bmod P \tag{2.17}$$

$$f_j(0) \equiv d_j \bmod p_j \tag{2.18}$$

$$f_j(x) \equiv f(x) \bmod p_j \tag{2.19}$$

[33] shares unfixed size data blocks with linear equation. There are t_l data pieces $\{d_{l,q}\}_{q=1,\cdots,t_l}$ in data block b_l ($t_{l_1} < t_{l_2}$ if $l_1 < l_2$). Then, $o \times n$ shares $\{e_{l,i}\}_{l=1,\cdots,o;i=1,\cdots,n}$ are created by multiplying b_l with random matrix $A = [a_{x,y}]_{n\times\max(t_1,\cdots,t_o)}$ by Equation 2.20, where $A_l = [a_{x,y}]_{n\times t_l}$ and A_l is built from the first t_l columns of A. Next, o shares $\{e_{l,i}\}_{l=1,\cdots,o}$ are stored at PT_i and matrix A is published on the NB. Finally, original data from block b_l are reconstructed from matrix A_l and t_l shares $\{e_{l,i}\}_{i=1,\cdots,t_l}$ by solving linear Equation 2.20.

$$[e_{l,1}, \cdots, e_{l,n}]^T = A_l \times [b_l]^T \tag{2.20}$$

[29]'s MSSS extends from [29]'s SSS (Section 2.1.1) with a new sharing process. At PT_i, shares $\{u_{i,a,j}\}_{a=1,\cdots,n}$ of data pieces d_j are computed and distributed to other PTs [82] (Section 2.1.4). However, PT_i's actual share $e_{i,j}$ is computed by weighting the sum of other PTs' shares (Equation 2.21), where w_a is a random integer (weight).

$$e_{i,j} = \sum_{a=1}^{n} w_a \times u_{i,a,j} \tag{2.21}$$

In [34], PTs are categorized into two groups: $G_1 = \{PT_i\}_{i=1,\cdots,\gamma}$ and $G_2 = \{PT_i\}_{i=\gamma+1,\cdots,n}$, with the objective of reducing shared data volume. PTs of G_1 store only one key and one share. To share m data pieces $\{d_j\}_{j=1,\cdots,m}$, a key k_i and an identifier ID_i are defined for each PT_i. Next, a first polynomial $f_1(x)$ is defined by Equation 2.22, where coefficients $\{u_{1,v}\}_{v=1,\cdots,t-1}$ are random integers. Then, n shares $\{e_{1,i}\}_{i=1,\cdots,n}$ are created by Equation 2.23. Moreover, $(m-1) \times \gamma$ pseudo shares $\{e_{j,i}\}_{j=2,\cdots,m;i=1,\cdots,\gamma}$ are generated with a pseudo-random number generator, keys $\{k_i\}_{i=1,\cdots,\gamma}$ and shares $\{e_{1,i}\}_{i=1,\cdots,\gamma}$. Next, $m-1$ polynomials $f_2(x), \cdots, f_m(x)$ (Equation 2.22) are solved from pseudo shares $\{e_{j,i}\}_{j=2,\cdots,m;i=1,\cdots,\gamma}$ and original data $\{d_j\}_{j=2,\cdots,m}$ to construct the other $(m-1) \times (n-\gamma)$ shares $\{e_{j,i}\}_{j=2,\cdots,m;i=\gamma+1,\cdots,n}$ (Equations 2.22 and 2.23). Eventually, each $PT_i \in G_1$ stores k_i and one share $e_{1,i}$; and each $PT_i \in G_2$ stores shares $\{e_{j,i}\}_{j=1,\cdots,m}$.

$$f_j(x) = d_j + \sum_{v=1}^{t-1} u_{j,v} \times x^v \quad (2.22)$$

$$e_{j,i} = f_j(ID_i) \quad (2.23)$$

To reconstruct original data, t of n PTs in both G_1 and G_2 are selected. If $PT_i \in G_1$, pseudo shares $\{e_{j,i}\}_{j=2,\cdots,m}$ are generated as above. Then, original data are reconstructed by Lagrange interpolation from their shares, $m \times t$ pseudo shares and t IDs.

2.1.4 Group 4: Data-Verifiable Secret Sharing Schemes

There are only three (t, n) VSSSs in this group. [82] helps each PT verify other PTs' shares with the help of an RSA cryptosystem. To share data piece d, at PT_i, a random polynomial function f_i (Equation 2.24) is created such that $d = \sum_{i=1}^{n} w_{i,0}$. Then, signatures $\{s\_d_{i,v}\}_{v=0,\cdots,t-1}$ are created (Equation 2.25, where p is a prime and $d = \log_p \prod_{i=1}^{n} y_i$) and shared on the NB. Then, shares $\{u_{i,a}\}_{a=1,\cdots,n}$ are created by Equation 2.26 and distributed to other PTs. PT_i's actual share e_i is created by summing other PTs' shares (Equation 2.27) if they are correct (Equation 2.28). Data is reconstructed by Lagrange interpolation.

$$f_i(x) = \sum_{v=0}^{t-1} w_{i,v} \times x^v \quad (2.24)$$

$$s\_d_{i,v} = \begin{cases} y_i & \text{if } v = 0 \\ p^{w_{i,v}} & \text{Otherwise} \end{cases} \quad (2.25)$$

$$u_{i,a} = f_i(a) \quad (2.26)$$

$$e_i = \sum_{a=1}^{n} u_{a,i} \quad (2.27)$$

$$p^{u_{a,i}} = \prod_{v=0}^{t-1} (s\_d_{a,v})^{i^v} \quad (2.28)$$

[83] extends from [23] by verifying data correctness. To this aim, in the sharing process, a signature s_d is created for each data piece d (Equation 2.29, where u is a random integer). Then, s_d is published on the NB.

$$s\_d = u^d \bmod p \quad (2.29)$$

In the reconstruction process, data piece d is reconstructed from t shares by secure multi-party computation (SMC) [90] (Equation 2.30). Next, a multi-prover zero-knowledge argument [91] helps verify data correctness. Data piece d is correct if $u^{v_1''+\cdots+v_n''} \times s\_d^{v_0} = \prod_{i=1}^{n} v_i' \bmod p$, where $\{v_i'\}_{i=1,\cdots,n}$ and $\{v_i''\}_{i=1,\cdots,n}$ are generated by Equations 2.31 and 2.32, respectively, and $\{v_i\}_{i=0,\cdots,n}$ and $\{w_i\}_{i=0,\cdots,n}$ are random integers such that $d = \sum_{i=1}^{n} w_i$.

$$d = \sum_{i \in G} \left(e_i \times \prod_{j \in G, j \neq i} j/(j-i) \right) \tag{2.30}$$

$$v_i' = u^{v_i} \bmod p \tag{2.31}$$

$$v_i'' = v_i - v_0 \times w_i \bmod p \tag{2.32}$$

[35] exploits NTRU encryption and a hash function to verify data correctness. First, n pairs of PT_i keys $(k_{i,1}, k_{i,2})_{i=1,\cdots,n}$ are randomly created with NTRU. Then, shares e_i and signatures s_d_i are created by Equations 2.33 and 2.34, respectively, where $\{x_i\}_{i=1,\cdots,n}$ are random integers, w is a random polynomial called blinding value and f is a random polynomial [23]. Keys $(k_{i,1}, k_{i,2})$ and shares e_i are stored at PT_i and $\{x_i\}_{i=1,\cdots,n}$ and signatures $\{s\_d_i\}_{i=1,\cdots,n}$ are published on the NB. Before reconstruction, each share e_i is verified for correctness by Equations 2.35 and 2.36. Finally, t pairs of $(e_i, x_i)_{i=1,\cdots,n}$ help reconstruct data from the polynomial by Lagrange interpolation.

$$e_i \equiv (w \times k_{i,1} + f(x_i)) \bmod p_1 \tag{2.33}$$

$$s\_d_i \equiv (w \times k_{i,1} + H(f(x_i))) \bmod p_1 \tag{2.34}$$

$$y_i \equiv k_{i,2} \times e_i \bmod p_1 \bmod p_2 \tag{2.35}$$

$$y_i \equiv k_{i,2} \times s\_d_i \bmod p_1 \bmod p_2 \tag{2.36}$$

2.1.5 Group 5: Key-Verifiable Secret Sharing Schemes

In [36], the only (t, n) VSSS in this group, PT keys and signatures are independent. Hence, if some PTs come or go, the keys of other PTs do not change. PT keys $\{k_i\}_{i=1,\cdots,n}$ and identifiers $\{ID_i\}_{i=1..n}$ are randomly chosen. On the other hand, key signatures $\{s\_k_i\}_{i=1,\cdots,n}$ are generated with the help of an RSA cryptosystem (Equation 2.37). Then, key k_i is stored at PT_i, while identifiers and key signatures $(ID_i, s\_k_i)_{i=1,\cdots,n}$ are published on the NB. For sharing data piece d, several groups of PTs $\{G_r\}_{r=1,\cdots,g}$ are selected, and then shares $\{e_r\}_{r=1,\cdots,g}$ are created by Equations 2.38, 2.39, 2.40 and 2.41, where u is a random integer, $p_4 > p_3$ and $p_4 > p_2 > p_1$. Next, v, w, $\{G_r\}_{r=1,\cdots,g}$ and $\{e_r\}_{r=1,\cdots,g}$ are published on the NB. Before reconstruction, the key signature of

$PT_i \in G_r$ is verified to check whether $s\_k_i = v^{k_i} \bmod p_2$. If this is true, data are decrypted by Equations 2.42 and 2.43.

$$
\begin{aligned}
s\_k_i &= (p_1)^{k_i} \bmod p_2 && (2.37)\\
v &= (p_1)^{u} \bmod p_2 && (2.38)\\
u \times p_3 &= a \bmod \phi(p_2) && (2.39)\\
w_r &= d \oplus (s\_k_{r,1})^u \bmod p_2 \oplus, \cdots, \oplus (s\_k_{r,g})^u \bmod p_2 && (2.40)\\
e_r &= w_r \times \prod_{x=1}^{t} \tfrac{1-ID_{r,x}}{-ID_{r,x}} \\
&+ \sum_{x=1}^{t} \left(\tfrac{(s\_k_{r,x})^u \bmod p_2}{ID_{r,x}} \times \prod_{y=1,y\neq x}^{t} \tfrac{1-ID_{r,y}}{ID_{r,x}-ID_{r,y}} \right) \bmod p_4 && (2.41)\\
d &= w_r \oplus \left(v^{k_{r,1}} \bmod p_2 \right) \oplus, \cdots, \oplus \left(v^{k_{r,g}} \bmod p_2 \right) && (2.42)\\
w_r &= e_r \times \prod_{x=1}^{t} \tfrac{-ID_{r,x}}{1-ID_{r,x}} \\
&+ \sum_{x=1}^{t} \left(\tfrac{v^{k_{r,x}} \bmod p_2}{ID_{r,x}-1} \times \prod_{y=1,y\neq x}^{t} \tfrac{-ID_{r,y}}{ID_{r,x}-ID_{r,y}} \right) \bmod p_4 && (2.43)
\end{aligned}
$$

2.1.6 Group 6: Key *and* Data-Verifiable Secret Sharing Schemes

Unlike other schemes, [37]'s (t, n) VSSS verifies the correctness of both keys and shares. Moreover, it achieves a smaller share size than that of original data, by splitting data before encryption. In the sharing process, key k_0 and keys $\{k_i\}_{i=1,\cdots,n}$ are randomly selected from a prime and distinct positive integers, respectively. Key signatures $\{s\_k_i\}_{i=0,\cdots,n}$ are constructed by Equation 2.44, where z is a positive integer and $\varphi(p)$ is Euler's totient function. Next, any data piece d is split into t^2 smaller pieces stored in Matrix $D = [d_{x,y}]_{t\times t}$. Then, two types of shares are created (PTs' shares and NB's shares). PTs' shares $\{E_i = \{e_{i,1}, \cdots, e_{i,a}\}\}_{i=1,\cdots,n}$ are randomly selected from positive integers such that:

- a is a random integer,
- all entries in E_i are distinct positive integers,
- $e_{i,0}$ is the sum of all entries in E_i ($e_{i,0} = \sum_{h=1}^{a} e_{i,h}$) and $e_{i,0} < p$.

To construct the NB's shares $\{c_i\}_{i=1,\cdots,n}$, polynomial function $f(x)$ (Equation 2.45) is created from data and PTs' shares by Equations 2.46 and 2.47, where A is a Jordan normal form of D[2]. Finally, NB's shares $\{c_i\}_{i=1,\cdots,n}$ are constructed from Equations 2.48 and 2.49; and share signatures $\{s\_d_{i,j}\}_{i=1,\cdots,n;j=1,\cdots,m}$ are created from Equation 2.50. Keys k_i and shares E_i are stored at PT_i; shares $\{c_i\}_{i=1,\cdots,n}$, share signatures

[2] A is a Jordan normal form of D if $DY = YA$, where Y is a row matrix and A is a square, upper triangular matrix whose entries are all the same integer values on the diagonal, all 1 on the entries immediately above the diagonal, and 0 elsewhere.

$\{s\_d_{i,j}\}_{j=1,\cdots,n;j=1,\cdots,m}$, key k_0, key sinatures $\{s\_k_i\}_{i=0,\cdots,n}$, p and A are published on the NB.

$$s\_k_i = \begin{cases} k_0^{-1} \bmod \varphi(p) & \text{if } i = 0 \\ z^{k_i} \bmod p & \text{if } 1 \le i \le n \end{cases} \tag{2.44}$$

$$f(x) = \sum_{i=1}^{t-1} u_i \times x^{i-1} \tag{2.45}$$

$$u_i = (((z)^{k_0})^{e_{i,0}})^{-1} y_i \bmod p \tag{2.46}$$

$$D \times [y_1, \cdots, y_t]^T = [y_1, \cdots, y_t]^T \times A \tag{2.47}$$

$$c_i = f(v_i) \tag{2.48}$$

$$v_i = ((z)^{k_0})^{k_i} \bmod p \tag{2.49}$$

$$s\_d_{i,j} = z^{e_{i,j}} \bmod p \tag{2.50}$$

In the reconstruction process, key k_i is correct if $((z)^{k_i})^{s\_k_{n+1}} = s\_k_i \bmod p$. PT_i's share $e_{i,j}$ is correct if $(((z)^{k_0})^{e_{i,j}})^{s\_k_{n+1}} = s\_d_{i,j} \bmod p$. Next, polynomial function $f(x)$ is reconstructed from t pairs of key and NB's share $\{k_i, c_i\}$ by Lagrange interpolation and Equation 2.49. Then, $\{y_a\}_{a=1,\cdots,t}$ are created by Equation 2.51. Finally, data piece d is reconstructed by solving Equation 2.47.

$$y_i = u_i \prod_{j=1}^{a} ((z)^{k_0})^{e_{i,j}} \tag{2.51}$$

2.1.7 Group 7: Data-Verifiable Multi Secret Sharing Schemes Type I

The only (m, t, n) VMSSS type I in this group encrypts and decrypts all data at once with the helps of a cellular automaton, to enhance computation performance. Moreover, the correctness of shares is verified before reconstruction [38]. In the sharing process, a set of integers $\{u_1, \cdots, u_{\max(m,t)}, \cdots, u_{w+n}\}$ is created, where w is a random integer such that $w \ge \max(m, t)$, $u_j = d_j$ if $1 \le j \le \min(t, m)$ and u_j is a random integer when $m < j \le t$. Others values of u_j are created with the help of the cellular automaton. Then, shares $\{c_h\}_{h=1,\cdots,m-t}$ are generated by Equation 2.52. Shares $\{e_i\}_{i=1,\cdots,n}$ and their signatures $\{s\_d_i\}_{i=1,\cdots,n}$ are created by Equations 2.53 and 2.54, where v is a random integer. Finally, each share e_i is shared at PT_i and shares $\{c_h\}_{h=1,\cdots,m-t}$ and signatures $\{s\_d_h\}_{h=1,\cdots,n}$ are published on the NB.

$$c_h = d_{t+h} + u_{t+h} \,(\mathrm{mod} 2) \tag{2.52}$$

$$e_i = u_{m+i} \tag{2.53}$$

$$s\_d_i = v^{e_i} \bmod p \tag{2.54}$$

Before reconstruction, share integrity is verified by Equation 2.54. Next, $\{u_1, \cdots, u_{\max(m,t)}, \cdots, u_{w+n}\}$ are reconstructed from t shares with the cellular automaton. Finally, all data are regenerated by Equation 2.55.

$$d_j = \begin{cases} u_j & \text{if } 1 \le j \le \min(t, m) \\ c_{j-t} + u_j \,(\mathrm{mod} p) & \text{otherwise} \end{cases} \tag{2.55}$$

2.1.8 Group 8: Key-Verifiable Multi Secret Sharing Schemes Type I

A fair amount of research has been done on (m, t, n) VMSSSs type I, half of which belong to this group. [39] extends from [30] by verifying whether keys shared between PTs are correct. In the sharing process, each key k_i, its signature s_k_i and public key v are created by Equations 2.56, 2.57 and 2.58, respectively, where prime p_1 is a multiple of prime p_2, $\{u_i\}_{i=0,\cdots,n}$ are random integers and ϕ is the Euler phi-function. Key k_i is stored at PT_i and $\{s\_k_i\}_{i=0,\cdots,n}$ and v are published on the NB. Before reconstruction, keys are verified. Key k_i is correct if $((s\_k_0)^{k_i})^v \equiv u_i' \mod p_1$.

$$k_i = (s\_k_i)^{u_0} \bmod p_1 \tag{2.56}$$

$$s\_k_i = (p_2)^{u_i} \bmod p_1 \tag{2.57}$$

$$v = (u_0)^{-1} \bmod \phi(p_1) \tag{2.58}$$

[40] also extends from [30] with the same goal. Only key and signature generation actually varies. However, the verification process is more efficient. Key k_i is created by Equations 2.59, 2.60 and 2.61, where $u_{i=1,2,3}$ are random integers and f is any two-variable one-way function. Signature s_k_i of key k_i is created by Equation 2.62, where u_4 is a random integer. Key k_i is stored at PT_i, while $u_1, \cdots, u_4$ and $\{s\_k_i\}_{i=1,\cdots,n}$ are published on the NB.

$$k_i = f(u_1, w_i) \tag{2.59}$$
$$w_i = ((v_i)^{u_3})^{u_2} \bmod p \tag{2.60}$$
$$u_2 \times u_3 \equiv 1 \bmod \phi(p) \tag{2.61}$$
$$s\_k_i = (u_4)^{k_i} \bmod p \tag{2.62}$$

[41] in turn extends from [40] by proposing new encryption and decryption processes to reduce computation cost. After keys and signatures are created, shares $\{c_{j,1}\}_{j=1,\cdots,n}$ and $\{c_{j,2}\}_{j=1,\cdots,m}$ are generated by Equations 2.63, 2.64, 2.65 and 2.66, where u_0 is a random integer. Next, $\{c_{j,1}\}_{j=1,\cdots,n}$ and $\{c_{j,2}\}_{j=1,\cdots,m}$ are published on the NB. After key verification, data are reconstructed by Equations 2.67, 2.68 and 2.69.

$$c_{j,1} = d_j - y_{j+n} \tag{2.63}$$
$$c_{j,2} = k_j - y_{j-1} \tag{2.64}$$
$$y_j = \begin{cases} k_{j+1} & \text{if } 0 \le j < t \\ -\sum_{v=1}^{t} u_v \times y_{j-v} \bmod p & \text{Otherwise} \end{cases} \tag{2.65}$$
$$(x - u_0)^t = x^t + u_1 \times x^{t-1} + \cdots + u_t = 0 \tag{2.66}$$
$$d_j = y_{j+n} + c_{j,2} \tag{2.67}$$
$$y_j = \begin{cases} k_{j+1} & \text{if } 0 \le j < t \\ k_{j+1} - c_{j+1,1} & \text{if } t \le j < n \\ f(j) \times (u_0)^j \bmod p & \text{Otherwise} \end{cases} \tag{2.68}$$
$$f(x) = \sum_{v=1}^{t} \frac{y_{v-1}}{(u_0)^{v-1}} \prod_{w=1 \& w \neq v}^{t} \frac{x - w + 1}{v - w} \bmod p \tag{2.69}$$

[50] extends from [48] (Section 2.1.9) to improve the efficiency of encryption and decryption. To this aim, data are split into blocks of size t that are each encrypted and decrypted all at once. Block b_l is encrypted into n shares $\{c_{l,h}\}_{h=1,\cdots,n}$ by Equation 2.70, where $A = [a_{i,w}]_{t \times n}$, $a_{i,w} = H(u_l \times k_i \times v)^{w-1}$, and $\{u_l\}_{l=1,\cdots,o}$ and v are random integers. Key k_i is stored at PT_i, whereas key signatures $\{s\_k_i\}_{i=1,\cdots,n}$, shares $\{c_{l,h}\}_{l=1,\cdots,o;h=1,\cdots,n}$ and $\{x_l = u_l \times v\}_{l=1,\cdots,o}$ are published on the NB. To reconstruct data, shares and keys are verified for correctness with a bilinear map $f(u_l \times k_i \times v, v) = f(x_l, s\_k_i)$. Then, data are reconstructed by solving Equation 2.70.

$$[c_{l,1}, \cdots, c_{l,n}]^T = A \times [b_l]^T \tag{2.70}$$

[51] also extends from [48], pursuing the same goal as [50]. The difference is that data are encrypted into $n + m - t$ shares to reduce the number of shares. Shares $\{c_h\}_{h=1,\cdots,(n+m-t)}$ are computed by Equation 2.71, where $A = [a_{x,y}]_{(m+n)\times(m+n-t)}$, $a_{x,y} = (w)^{x(y-1)}$, $z_i = H(u \times v \times k_i)$, and u, v and w are random integers. Key k_i is stored at PT_i, whereas key signatures $\{s\_k_i\}_{i=1,\cdots,n}$, shares $\{c_h\}_{h=1,\cdots,(n+m-t)}$, data signatures $\{s\_d_j\}_{j=1,\cdots,m}$ and $x = u \times v$ are published on the NB. To reconstruct data, shares and keys are verified for correctness with a bilinear map $f(u \times k_i \times v, v) = f(x, s\_k_i)$. Then, data are reconstructed by solving Equation 2.71.

$$[c_1, \cdots, c_{n+m-t}]^T = A \times [z_1, \cdots, z_n, d_1, \cdots, d_m]^T \tag{2.71}$$

Unlike in other schemes, PTs in [42] can be added or deleted. Moreover, threshold t can be vary. To this aim, keys k_i, key signatures s_k_i and PT identifiers ID_i are randomly selected such that they are different from one PT to the other. Then, data are organized into unfixed-sized blocks, where each data block b_l stores u_l data pieces. All data pieces $\{d_{l,q}\}_{q=1,\cdots,u_l}$ in block b_l are encrypted to $n+u_l-t_l$ shares $\{c_{l,h}\}_{h=1,\cdots,(n+u_l-t_l)}$ by Equations 2.72, 2.73, 2.74, 2.75 and 2.76, where z_l is a random integer. Each key k_i is stored at PT_i and identifiers $\{ID_i\}_{i=1,\cdots,n}$, signatures $\{s\_k_i\}_{i=1,\cdots,n}$ and shares $\{y_l\}_{l=1,\cdots,n}$ and $\{c_{l,h}\}_{l=1,\cdots,o;h=1,\cdots,(n+u_l-t_l)}$ are published on the NB. Before reconstruction, keys are verified for correctness with a discrete logarithm modulo and a one-way hash function. Finally, each data piece $d_{l,q}$ of data block b_l is reconstructed by Lagrange interpolation.

$$c_{l,h} = f_l(n + u_l + h) \tag{2.72}$$

$$f_l(x) = \sum_{v=1}^{u_l} d_{l,v} \times \Delta_1 + \sum_{v=1}^{n} (s\_k_v)^{z_l} \times \Delta_2 \bmod p_1 \tag{2.73}$$

$$\Delta_1 = \prod_{w=1\&w\neq v}^{u_l} \frac{x - (n+w)}{v - w} \times \prod_{i=1}^{n} \frac{x - ID_i}{(n+v) - ID_i} \tag{2.74}$$

$$\Delta_2 = \prod_{i=1\&i\neq v}^{n} \frac{x - ID_i}{ID_v - ID_i} \times \prod_{w=1}^{u_l} \frac{x - (n+w)}{ID_v - (n+w)} \tag{2.75}$$

$$y_l = (p_2)^{z_l} \bmod p_1 \tag{2.76}$$

[44] extends from [28] to reduce computation cost, and verify correctness. Original data are organized into blocks of size $t-1$. Keys $\{k_i\}_{i=1,\cdots,n}$ are randomly selected and their signatures $\{s\_k_i\}_{i=1,\cdots,n}$ are created by Equation 2.77, where H is a hash function. In each data block b_l, the first data piece $d_{l,1}$ is encrypted into two shares $c_{l,1,1}$ and $c_{l,1,2}$ by Equation 2.78, where u is a random integer. Other data pieces in block b_l are shared by Equations 2.79 and 2.80. Key k_i is stored at PT_i and $\{s\_k_i\}_{i=1,\cdots,n}$,

$\{c_{l,q,h}\}_{l=1,\cdots,o;q=1,\cdots,t-2;h=1,\cdots,q+1}$ and $\{c_{l,t-1,h}\}_{l=1,\cdots,o;h=1,\cdots,n}$ are published on the NB. Before reconstruction, each key k_i is verified for validity by Equation 2.81. Then, all data in each block are reconstructed by Lagrange interpolation.

$$s\_k_i = H\left(H^{t-1}(k_i) \oplus k_i\right) \tag{2.77}$$

$$c_{l,1,h} = u \times h + d_{l,1} - (k_q \oplus H(k_i)) \tag{2.78}$$

$$c_{l,q,h} = f_{l,q}(h) - (k_q \oplus H^q(k_q)) \tag{2.79}$$

$$f_{l,q}(x) = \begin{cases} d_{l,q} + u \times x & \text{if } q = 1 \\ d_{l,q} + \sum_{v=1}^{q} x^v \times f_{l,q-1}(x) & \text{Otherwise} \end{cases} \tag{2.80}$$

$$s\_k_i = H\left(H^{t-1}(k_i) \oplus k_i\right) \tag{2.81}$$

Finally, [43] propose two schemes. They create keys and verify their correctness by using a one-way hash function and a LFSR public key cryptosystem [92, 93]. The first scheme encrypts and decrypts data as [30], while the second scheme does as [41], while providing higher security than [30, 41] with keys of same lengths.

2.1.9 Group 9: Key *and* Data-Verifiable Multi Secret Sharing Schemes Type I

The other third of (m, t, n) VMSSSs type I belong to this group. [46] prevents cheating from malicious PTs by verifying both shares and keys. Keys $\{k_i\}_{i=1,\cdots,n}$ and their signatures $\{s\_k_i\}_{i=1,\cdots,n}$ are created by Equations 2.82, 2.83, 2.84, 2.85 and 2.86, where $\{u_v\}_{v=0,\cdots,t-1}$ are random integers and $a_1, \cdots, a_5$ are set as discrete logarithms. Let p_1 and p_2 be big primes. a_1 is a random integer, $a_2 = (2 \times p_1 + 1)(2 \times p_2 + 1)$, $a_3 = p_1 \times p_2$ and $a_3 \times a_2 = \phi(a_5)$, where ϕ is Euler's quotient function. Key k_i is stored at PT_i, while signatures $\{s\_k_i\}_{i=1,\cdots,n}$ and $\{w_v\}_{v=0,\cdots,t-1}$ are published on the NB. Key correctness is checked by Equation 2.87.

$$f(x) = \left(\sum_{v=0}^{t-1} u_v \times x^v\right) \bmod a_3 \tag{2.82}$$

$$w_v = (p_1)^{u_v} \bmod a_2 \tag{2.83}$$

$$y_i = \prod_{\forall PT_v, v \neq i} (ID_i - ID_v) \bmod a_3 \tag{2.84}$$

$$k_i = (f(ID_i)/y_i) \bmod a_3 \tag{2.85}$$

$$s\_k_i = (a_1)^{k_i} \bmod a_2 \tag{2.86}$$

$$((a_1)^{y_i})^{k_i} = \prod_{v=0}^{t-1} (w_v)^{(ID_i)^v} \bmod a_2 \tag{2.87}$$

A 4-tuple of shares $\{c_{j,1}, \cdots, c_{j,4}\}$ is created by Equations 2.88 and 2.89, where $c_{j,1}$ and $c_{j,2}$ are random integers. Shares $\{c_{j,h}\}_{j=1,\cdots,m;h=1,\cdots,4}$ are published on the NB. Before reconstruction, each PT_i must verify share and key correctness by Equation 2.90. If verification is positive, original data are reconstructed by Equations 2.91 and 2.92, where G is any group of t PTs.

$$c_{j,3} = (a_1)^{-a_5+c_{j,1}} \times (c_{j,2})^{2\times a_5+c_{j,1}+1} \bmod a_2 \quad (2.88)$$

$$c_{j,4} = ((c_{j,2})^{u_0} - d_j)(c_{j,3})^{-u_0} \bmod a_2 \quad (2.89)$$

$$((c_{j,3})^{k_i})^{a_4} \equiv (s\_k_i)^{a_4\times c_{j,1}-1} \times ((c_{j,2})^{k_i})^{2+a_4(c_{j,1}+1)} \bmod a_2 \quad (2.90)$$

$$d_j = \left(\prod_{PT_i\in G}((c_{j,2})^{k_i})^{\triangle_i}\right) - \left(c_{j,4}\prod_{PT_i\in G}((c_{j_3})^{k_i})^{\triangle_i}\right) \bmod a_2 \quad (2.91)$$

$$\triangle_i = \prod_{\forall PT_v\in G} -ID_v \times \prod_{\forall PT_v\in G}(ID_i - ID_v) \quad (2.92)$$

[45] extends from [46] to improve the efficiency of encryption and decryption. To this aim, j 3-tuples of shares $\{c_{j,1}, c_{j,3}, c_{j,4}\}_{j=1,\cdots,m}$ are created by Equations 2.93 and 2.94 and published on the NB. Before reconstruction, each PT must verify share and key correctness by Equation 2.95. If verification is positive, original data are reconstructed by Equations 2.96 and 2.92.

$$c_{j,3} = (a_1)^{a_5\times c_{j,1}} \bmod a_2 \quad (2.93)$$

$$c_{j,4} = \left((a_1)^{u_0\times a_5\times c_{j,1}} \bmod a_2\right) \oplus d_j \quad (2.94)$$

$$((c_{j,3})^{k_i})^{a_4} \equiv (s\_k_i)^{c_{j,1}} \bmod a_2 \quad (2.95)$$

$$d_j = c_{j,4} \oplus \prod_{\forall PT_i\in G}((c_{j,3})^{k_i})^{\triangle_i} \bmod a_2 \quad (2.96)$$

[47] extends from [30] by checking whether keys and shares are valid, with the help of a discrete logarithm. Signatures $\{s\_d_j\}_{j=1,\cdots,\max(m,t)}$ are created after data sharing by Equation 2.97, where $\{u_j\}_{j=1,\cdots,m}$ are secret data ($u_j = d_j$) and $\{u_j\}_{j=(m+1),\cdots,t}$ are random integers. They are then published on the NB. Before reconstruction, keys are verified first, and then shares are, both by Equation 2.98. Signature s_d_j is also used to check share integrity.

$$s\_d_j = (p_1)^{u_j} \bmod p_2 \quad (2.97)$$

$$(p_1)^{c_i} = \prod_{h=1}^{\max(t,m)}(c_{h+n+1})^{f(w,k_i)^h} \bmod p_2 \quad (2.98)$$

In [48], each data piece d_j is encrypted independently to vary threshold t_j. Keys $\{k_i\}_{i=1,\cdots,n}$ are randomly selected such that their signatures $\{s\_k_i\}_{i=1,\cdots,n}$ (Equation 2.99, where v is a random integer) are unique. Each data piece d_j is encrypted to n shares $\{c_{j,h}\}_{h=1,\cdots,n}$ by Equations 2.100 and 2.101, where $A_j = [a_{x,y}]_{(n\times t_i)}$, $a_{x,y} = (u)^{x(y-1)}$, $Z_j = [w_j \times v, d_j \times (k_1)^v, \cdots, d_j \times (k_n)^v]$ and u and w_j are random integers. Signature s_d_j of d_j is created by Equation 2.102. Keys k_i are stored at PT_i, whereas key signatures $\{s\_k_i\}_{i=1,\cdots,n}$, shares $\{w_j, c_{j,1}, \cdots, c_{j,n}\}_{j=1,\cdots,m}$, signatures $\{s\_d_j\}_{j=1,\cdots,m}$, u and v are published on the NB. Before reconstruction, shares and keys are verified for correctness with a bilinear map $f((k_i)^{s\_d_j}, v) = f(s\_d_j, (k_j)^v)$. Then, data are reconstructed by solving linear Equations 2.100 and 2.101.

$$s\_k_i = (k_i)^v \tag{2.99}$$

$$[c_{j,1}, \cdots, c_{j,n}]^T = A_j \times [Z_j]^T \tag{2.100}$$

$$d_j = H(w_j \times v) \tag{2.101}$$

$$s\_d_j = d_j \times v \tag{2.102}$$

Unlike other schemes that compute integers over a finite field, [49] exploits binary strings in all processes to improve data sharing/reconstruction efficiency. In the sharing process, two kinds of keys are randomly created in binary string format: PT keys $\{k_i\}_{i=1,\cdots,n}$ and user keys $\{u_{j,v}\}_{j=1,\cdots,m;v=1,\cdots,t_l}$. Then, each share $c_{j,h}$ is created by Equation 2.103, where H is a one-way hash function and $\|$ is the concatenation operator. Finally, shares $c_{j,h}$, $H(d_j)$, $H\left(H(k_i \parallel j \parallel h)\right)$ with $j = 1, \cdots, m$; $h = 1, \cdots, t_l$ and $i = 1, \cdots, n$, are published on the NB.

$$c_{j,h} = d_j \oplus \left\{\oplus_{i:PT_i \in u_{j,v}} H(k_i \parallel j \parallel h)\right\} \tag{2.103}$$

Data are reconstructed by Equation 2.104 if all keys pass the verification process, which is split in two. Before reconstruction, keys $\{k_i\}_{i=1,\cdots,n}$ are checked for correctness by comparison to signatures $H\left(H(k_i \parallel j \parallel h)\right)$. After reconstruction, data $\{d_j\}_{j=1,\cdots,m}$ are checked for correctness by comparison to signatures $H(d_j)$.

$$d_j = c_{l,h} \oplus \left\{\oplus_{i:PT_i \in u_{j,v}} H(k_i \parallel j \parallel h)\right\} \tag{2.104}$$

Finally, [52] extends from [35] by encrypting multiple data, to improve data sharing/reconstruction efficiency and reduce share volume. To this aim, PT_i's identifier ID_i is randomly selected and PT_i's key k_i and signatures $\{s\_k_v\}_{v=0,\cdots,(t-1)}$ are created by Equations 2.105 and 2.106, respectively, where x and y are randomly created with NTRU

and w is NTRU's blinding value. Each data piece d_j is encrypted into a 3-tuple of shares $\{c_{j,1}, c_{j,2}, c_{j,3}\}$ by Equations 2.107 and 2.108, where $c_{j,1}$ is a random integer. Key k_i is stored at PT_i, whereas identifiers $\{ID_i\}_{i=1,\cdots,n}$, signature $\{s\_k_v\}_{v=0,\cdots,(t-1)}$ and shares $\{c_{j,h}\}_{j=1,\cdots,m;h=1,\cdots,3}$ are published on the NB.

$$k_i = \sum_{v=0}^{t-1} u_v \times (ID_i)^v \tag{2.105}$$

$$s\_k_v = w \times x + u_v \bmod p_1 \tag{2.106}$$

$$c_{j,2} = w \times x + c_{j,1} \bmod p_1 \tag{2.107}$$

$$c_{j,3} = d_j \oplus H(u_0 \times c_{j,2}) \tag{2.108}$$

Before reconstruction, keys and shares are verified for correctness by Equations 2.109 and 2.110, respectively. Finally, data are reconstructed by Equation 2.111.

$$k_i = y \sum_{v=0}^{t-1} s\_k_v (ID_i)^v \bmod p_2 \tag{2.109}$$

$$y \times k_i \times c_{j,2} = y \sum_{v=0}^{t-1} (w_v \times (ID_i)^v \times c_{j,1}) \bmod p_2 \tag{2.110}$$

$$d_j = c_{j,3} \oplus H\left(\sum_{i \in G} k_i \times c_{j,2} \times \prod_{v \in G \& v \neq i} \frac{-ID_v}{ID_i - ID_v}\right) \tag{2.111}$$

2.1.10 Group 10: Key *and* Data-Verifiable Multi Secret Sharing Schemes Type II

[53] is the only (m, t, n) VMSSS type II. It exploits elliptic curve cryptography to verify the correctness of both shares and keys. In the sharing process, keys $K = \{k_{i,q}\}_{i=1,\cdots,n,q=1,\cdots,t}$ are randomly chosen from small integers. Then, $l \times t$ data pieces $\{d_{l,q}\}_{l=1,\cdots,o;q=1,\cdots,t}$ are organized into o data blocks $\{b_l\}_{l=1,\cdots,o}$ of size t. Each data block b_l is encrypted into n shares $\{e_{l,i}\}_{i=1,\cdots,n}$ by Equation 2.112. Signature $s\_d_{l,q}$ of $d_{l,q}$ is created by Equation 2.113, where u is an elliptic curve point. Keys $\{k_{i,q}\}_{q=1,\cdots,t}$ and shares $\{e_{l,i}\}_{l=1,\cdots,o}$ are stored at PT_i, whereas data signatures $\{s\_d_{l,q}\}_{l=1,\cdots,o;q=1,\cdots,t}$ are published on the NB. Before reconstruction, each share $e_{l,i}$ and its keys $\{k_{i,q}\}_{q=1,\cdots,t}$ are verified for correctness by Equation 2.114. Finally, each data block is reconstructed by solving t simultaneous linear equations (Equation 2.112).

$$[e_{l,1}, \cdots, e_{l,n}]^T = K \times [b_l]^T \bmod p \quad (2.112)$$

$$s\_d_{l,q} = u \times d_{l,q} \quad (2.113)$$

$$u \times [e_{l,1}, \cdots, e_{l,n}]^T = K \times [s\_d_{l,1}, \cdots, s\_d_{l,t}]^T \quad (2.114)$$

2.1.11 Discussion

In this section, we compare the SSSs presented in previous sections along four axes. First, we provide a global view of the evolution of SSSs since their inception (Section 2.1.11.1). Second, we synthesize and account for the various encryption and verification techniques used in SSSs to enforce data security (Section 2.1.11.2). Third, we compare the features provided by SSSs beyond data privacy and integrity (Section 2.1.11.3). Finally, we study the factors that influence the cost of cloud SSS-based solutions in the pay-as-you-go paradigm (Section 2.1.11.4).

2.1.11.1 Evolution of Secret Sharing Schemes

To clarify the historical relationships between the SSSs reviewed in the chapter and better visualize the improvements brought to Shamir's [23] since 1979, we refer the reader to Figure 2.7. In this flowchart, each scheme is identified by a bibliographical reference (in red), the group (in orange) and type (in yellow) it belongs to, and whether it enforces key (represented by a green K) and/or data (represented by a blue D) verification. Moreover, a brief text describes the novelty brought by each scheme. Finally, an arrow from scheme S_1 to scheme S_2 indicates that S_1 extends from S_2.

Figure 2.7 quite clearly shows that SSSs have been less studied for almost 25 years than since the 2000's, when they attracted new attention in conjunction with the development of new, on-line distributed systems, i.e., clusters, grids and the cloud. Moreover, research about secret sharing seems to have accelerated since 2012, with the wide spread of cloud computing and associated data security concerns.

2.1.11.2 Encryption and Verification Methods

We categorize SSSs into five subprocesses, i.e., data encryption, data decryption, key creation, key verification and data verification. Of course, data encryption and decryption are the main processes for all groups of SSSs (Table 2.1). Key creation is always optional. Finally, data verification is the focus of groups 4, 6, 7, 9 and 10; and

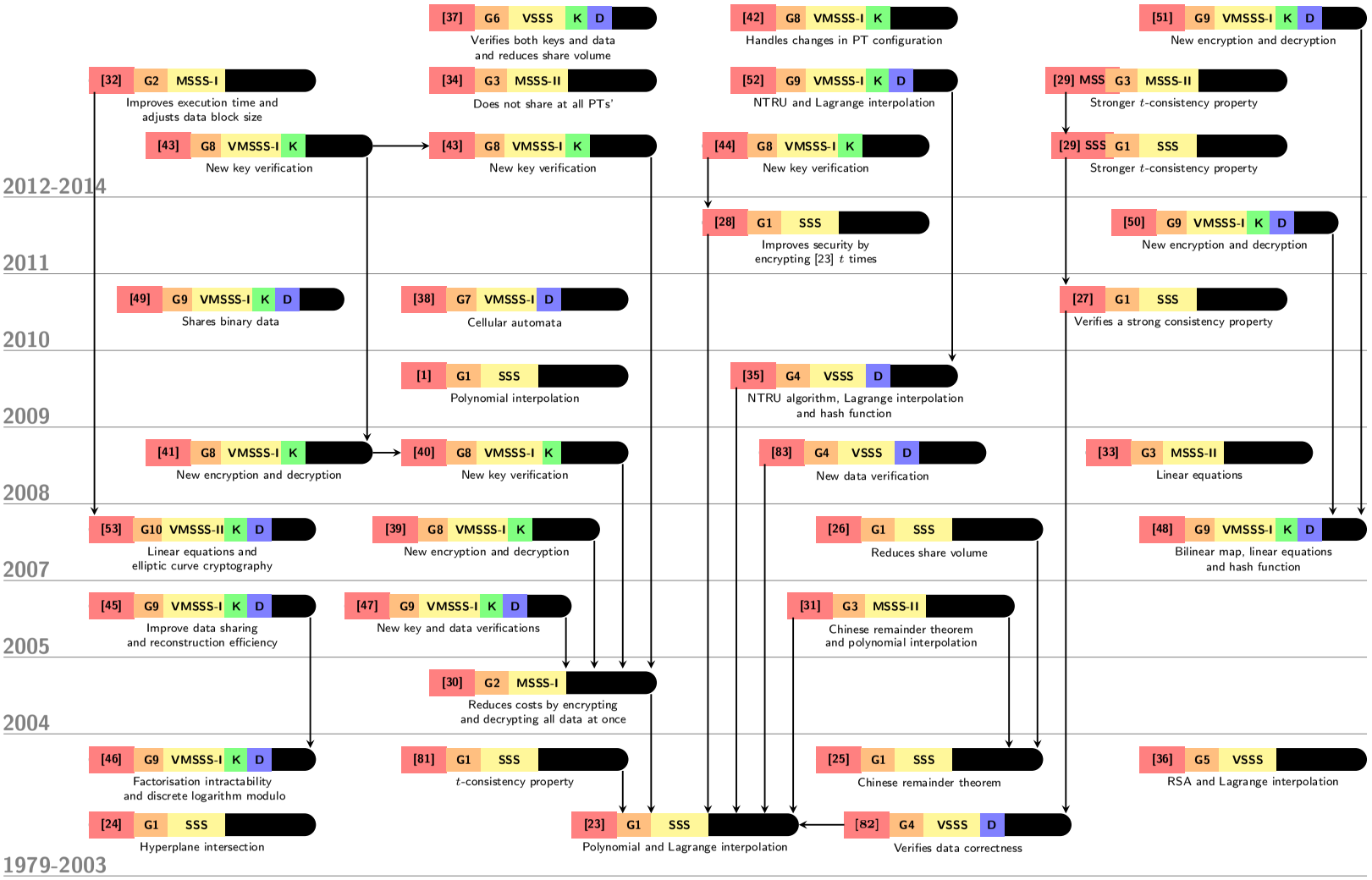


FIGURE 2.7: Evolution of SSSs

key verification the focus of groups 5, 6, 8, 9 and 10. The methods supporting these processes are summarized in Table 2.3.

Approximately half of the surveyed SSSs encrypt data by polynomial interpolation and decrypt data by Lagrange interpolation, as [23]. Yet, other methods, such as homomorphic encryption, NTRU or RSA make encryption stronger. Similarly, approximately half of the schemes necessitating keys generate them randomly, while more elaborate methods such as hash functions, LFSR, NTRU or RSA help better protect keys. Eventually, the same variety of methods is found in the key and data verification processes, although discrete logarithm modulo and hash functions are by far the most popular.

Given such variety, it is difficult to crisply rank the security level of all studied schemes. SSSs have indeed been continually addressing different issues over time, and thus adopted ad-hoc methods suited to their objectives. Moreover, the papers describing SSSs typically do not compare to one another. Thence, we push the comparison of SSSs' features and cost in the following subsections.

2.1.11.3 Features of Secret Sharing Schemes

SSSs mainly aim at enforcing data security (privacy, availability and integrity). However, in the context of cloud computing, efficient data access (update, search and aggregation operations) must also be made possible by SSSs. Thus, some SSSs allow computation directly over shares, i.e., without decrypting data. To provide a global overview, the features of all studied SSSs are synthesized in Table 2.4, where an X means a particular feature is supported by the corresponding SSS(s); NB means that data availability is supported, but only when the NB is accessible; G means that data availability is supported only when shares are replicated; IN and OUT stand for inner and outer code verification, respectively; and B means that updates operate on data blocks instead of individual shares.

Although all SSSs mainly handle privacy, [81] and most (V)MSSSs type-I are vulnerable if the NB is attacked, since they store all shares in the NB. Similarly, although most SSSs guarantee data availability, [33, 37, 81] and most (V)MSSSs type-I loose data access if the NB compromised.

Moreover, all VSSSs and VMSSSs guarantee at least one integrity feature. We categorize them into three classes: verifying keys, shares and data. First, all schemes in groups 5, 6, 8, 9 and 11 verify keys before reconstructing shares. Hence, they can detect PT cheating and prevent transferring any data back to the user when incorrect keys are

TABLE 2.3: Encryption and verification methods in SSSs

Type	Group	Scheme	Method				
			Data encryption	**Data decryption**	**Key creation**	**Key verification**	**Data verification**
SSS	Group 1	[23]	Polynomial interpolation	Lagrange interpolation			
		[24]	Hyperplane intersection				
		[25, 26]	Chinese remainder theorem		Random		
		[81]	Polynomial interpolation	Lagrange interpolation	Random		
		[1]	Polynomial interpolation		Random		
		[27],[29] SSS	Homomorphism and polynomial interpolation	Lagrange interpolation			
		[28]	Polynomial interpolation and recursion				
MSSS	Group 2	[30]	Polynomial interpolation	Lagrange interpolation	Two-variable one-way function		
		[32]	Linear equation and matrix multiplication		Random		
	Group 3	[31]	Chinese remainder theorem and polynomial interpolation		Random		
		[33]	Linear equation and matrix multiplication		Random		
		[29] MSSS	Polynomial interpolation and homomorphism	Lagrange interpolation			
		[34]	Polynomial interpolation and pseudo-random number generation	Lagrange interpolation	Random		
VSSS	Group 4	[82]	Homomorphism and polynomial interpolation	Lagrange interpolation			RSA cryptosystem
		[83]	Polynomial interpolation	SMC			Discrete logarithm modulo
		[35]	NTRU and one-way hash function	Lagrange interpolation	NTRU		NTRU
	Group 5	[36]	RSA cryptosystem and Lagrange interpolation		RSA cryptosystem		
	Group 6	[37]	Jordan matrix and linear equations		Random	Discrete logarithm modulo	Discrete logarithm modulo
VMSSS	Group 7	[38]	One-dimensional cellular automaton				Discrete logarithm modulo
	Group 8	[39]	Polynomial interpolation	Lagrange interpolation	Discrete logarithm modulo		
		[40]	Polynomial interpolation	Lagrange interpolation	Discrete logarithm modulo and two-variables one-way function	Discrete logarithm modulo	
		[41]	Homogeneous linear recursion	Lagrange interpolation	Discrete logarithm modulo and two-variables one-way function	Discrete logarithm modulo	
		[50, 51]	linear equations and hash function		Random	Bilinear map	
		[42]	Discrete logarithm modulo and Lagrange interpolation		Random	Discrete logarithm modulo and one-way hash function	
		[44]	Polynomial interpolation, recursion, XOR and one-way hash function	Lagrange interpolation, recursion, XOR and one-way hash function	XOR and one-way hash function		
		[43]-I	Polynomial interpolation	Lagrange interpolation	One-way hash function and LFSR public key cryptography		
		[43]-II	Homogeneous linear recursion	Lagrange interpolation	One-way hash function and LFSR public key cryptography		
	Group 9	[45, 46]	Factorisation intractability and discrete logarithm modulo		Discrete logarithm modulo	Factorisation intractability and discrete logarithm modulo	
		[47]	Polynomial interpolation	Lagrange interpolation	Two-variable one-way function	Discrete logarithm modulo	
		[48]	linear equations and hash function		Random	Bilinear map	
		[49]	One-way hash function and binary operations		Random binary	One-way hash function	
		[52]	NTRU, XOR and one-way hash function		NTRU	NTRU and one-way hash function	
	Group 10	[53]	Linear equations		Random	Elliptic curve cryptography	

TABLE 2.4: Features of SSSs

Type	**Group**	**Scheme(s)**	**Features**											
								Data access						
								Updates				**Exact**	**Aggregation**	**Other features**
			Data privacy	**Data availability**	**Key availability**	**Data integrity**	**Key integrity**	**Add**	**Delete**	**Append**	**Modify**	**match**		
SSS	Group 1	[23]	X	X				X	X	X			SUM, AVG	
		[24]	X	X				X	X	X		X	SUM, AVG	
		[1, 25, 26, 28]	X	X				X	X	X				
		[81]	X	NB				X	X	X			SUM, AVG	Strong t-consistency property
		[27],[29]SSS	X	X				X	X	X				Strong t-consistency property
MSSS	Group 2	[30]	X	NB	X									
		[32]	X		X			B	B					t and block size may vary
	Group 3	[31]	X	X	X									t may vary
		[33]	X	NB	NB			B	B					t may vary
		[29]MSSS	X	X				X	X	X				Strong t-consistency property
		[34]	X	X				X	X	X				
VSSS	Group 4	[35, 82]	X	X		OUT		X	X	X				
		[83]	X	X		IN		X	X	X				
	Group 5	[36]	X	G	X		OUT	X	X	X				n may vary
	Group 6	[37]	X	NB	X	OUT	OUT	X	X	X				
VMSSS	Group 7	[38]	X	NB		OUT								
	Group 8	[39–41, 43, 51]	X	NB	X		OUT							
		[42]	X	NB	X		OUT	B	B					n and t may vary
		[44, 50]	X	NB	X		OUT	B	B					
	Group 9	[45, 46, 52]	X	NB	X	OUT	OUT	X	X	X				
		[47]	X	NB	X	OUT	OUT							
		[48]	X	NB	X	OUT	OUT	X	X	X				t may vary
		[49]	X			IN	OUT	X	X	X		X		
	Group 10	[53]	X	X	X	OUT	OUT	B	B				SUM, AVG	

detected. Second, most schemes in groups 4, 6, 7, 9, 10 and 11 verify the correctness of shares before reconstruction to reduce computation cost at the user's (no reconstruction from incorrect shares). However, they require extra storage for signatures. Third, [49, 83] verify the correctness of reconstructed data. Their signature volumes are lower than that of the second class of VSSSs, since the number of shares is generally greater than that of data. However, incorrect data are detected only after they are already reconstructed. Finally, although VSSSs and MVSSSs guarantee integrity, they consume more storage to handle signatures and more CPU power to verify keys, shares, and/or data. We push the comparison of their costs in the following subsection.

SSSs manage data at two levels: data piece or data block. First, [34] and most schemes in groups 1, 4, 5, 6, 9 encrypt each data piece independently. Hence, they can directly add/delete/append data. Second, [32, 33, 42, 44, 50, 53] encrypt data blocks, and thus allow updating data blocks without reconstruction. Moreover, they update data faster because several data pieces are updated at once. In contrast, other schemes can only perform updates on unencrypted data, since they encrypt all data at once. Thus, they take longer execution times and use lots of memory to update data.

Some SSSs allow computing exact match on shares. Since [23, 24, 53, 81] use polynomial or linear equations to encrypt data, they allow sum and average operations on shares. Moreover, [24, 49] allow exact match on shares, because they use the same keys to encrypt all data pieces.

Finally, three more features are included in some schemes. First, [27, 29, 81] verify a strong t-consistency property. Thus, they guarantee that any subset of t shares or more always reconstruct the same data, but that any subset of t shares or fewer cannot. Second, [36, 42] allow the user to add and remove PTs to/from the PT pool by updating the value of n. Third, [31–33, 42, 48] allow the user to assign different values of t to different data pieces, to enforce different security levels for each data piece.

2.1.11.4 Costs

In the cloud pay-as-you-go paradigm, the cost of securing data must be balanced with the risk of data loss or pilfering, and thus the level of data security must be balanced with its cost. This is a particularly important issue with secret sharing, which basically multiplies original data volume by n in the worst case (provided individual share volume is not greater than original data volume). We summarize the costs induced by SSSs in Table 2.5.

SSS time complexity and storage volume depend on a few parameters: m, n and t. To determine time complexity and storage volume, we suppose that only m is big.

TABLE 2.5: Costs induced by SSSs

Type	Group	Scheme(s)	Average time complexity				Storage volume						
			Sharing process	Reconstruction process	Key verification	Data verification	Shares		Keys			Signatures	
							PTs	NB	PTs	NB	Client	PTs	NB
SSS	Group 1	[23]	$O(mnt)$	$O(mt^2)$			$mn\|\|d\|\|$						
		[24]	$O(mnt)$	$O(mt^2)$			$mn\|\|d\|\|$				$mt^2\|\|k\|\|$		
		[25]	$O(mn)$	$O(mt\beta)$			$mn\|\|d\|\|$				$(n+1)\|\|k\|\|$		
		[81]	$O(mnt)$	$O(mt^2)$				$mn\|\|d\|\|$	$2mn\|\|k\|\|$				
		[26]	$O(mn)$	$O(mt\beta)$			$\frac{mn\|\|d\|\|}{t}$				$(n+1)\|\|k\|\|$		
		[1]	$O(mn^2t)$	$O(mt^3)$			$mn\|\|d\|\|$		$nt\|\|k\|\|$				
		[27, 29]SSS	$O(mn^2t)$	$O(mt^3)$			$mn\|\|d\|\|$						
		[28]	$O(mn^2t)$	$O(mt^3)$			$mn\|\|d\|\|/t$						
MSSS	Group 2	[30]	$O(nt)$ if $m \leq t$ $O((n+m-t)t)$ otherwise	$O(\max(m^3, mt^2))$				$n\|\|d\|\|$ if $m \leq t$ $(n+m-t)\|\|d\|\|$ otherwise	$n\|\|k\|\|$				
		[32]	$O(\max(nt\beta, mt^2))$	$O(\max(t^3, mt^2))$				$m\|\|d\|\|$	$n\|\|k\|\|$	$nt\|\|k\|\|$			
	Group 3	[31]	$O(mnt)$	$O(\max(mt^2, mt\beta))$			$mn\|\|d\|\|$		$n\|\|d\|\|$		$2m\|\|k\|\|$		
		[33]	$O(mn)$ or $O(bnt)$	$O(mt^2)$ or $O(bnt^3)$			$\geq mn\|\|d\|\|/t$ or $bn\|\|d\|\|$			$\approx nt\|\|k\|\|$			
		[29]MSSS	$O(mnt)$	$O(mt^2)$			$mn\|\|d\|\|$						
		[34]	$O(mnt)$	$O(mt^2)$			$(mn+m\gamma+\gamma)\|\|d\|\|$		$\beta\|\|k\|\|$				
VSSS	Group 4	[82]	$O(mn^2t)$	$O(mt^3)$		$O(mt^2)$	$mn\|\|d\|\|$						$mt\|\|s\|\|$
		[83]	$O(mnt)$	$O(mt^2)$		$O(m)$	$mn\|\|d\|\|$				$4\|\|k\|\|$		$m\|\|s\|\|$
		[35]	$O(mnt)$	$O(mt^2)$		$O(mt)$	$mn\|\|d\|\|$		$2n\|\|k\|\|$	$n\|\|k\|\|$	$4\|\|k\|\|$		$mn\|\|s\|\|$
	Group 5	[36]	$O(mt^2g)$	$O(mt^2)$	$O(t)$			$gm\|\|d\|\|$	$n\|\|k\|\|$	$n\|\|k\|\|$			$n\|\|s\|\|$
	Group 6	[37]	$O(\max(\gamma t, mt^2))$	$O(\max(\gamma t, mt^2))$	$O(t)$	$O(t^2)$	$\gamma m\|\|d\|\|/t^2$	$mn\gamma\|\|d\|\|/t^2$	$n\|\|k\|\|$	$(2\gamma t+2)\|\|k\|\|$			$(mn\gamma+n+1)\|\|s\|\|$
VMSSS	Group 7	[38]	$O(\max(m, t^2))$	$O(\max(m, t^2))$		$O(n)$	$n\|\|d\|\|$	$(m-t)\|\|d\|\|$ if $t > m$					$n\|\|s\|\|$
	Group 8	[39]	$O(nt)$ if $m \leq t$ $O((n+m-t)t)$ otherwise	$O(\max(m^3, mt^2))$	$O(t)$			$n\|\|d\|\|$ if $m \leq t$ $(n+m-t)\|\|d\|\|$ otherwise	$n\|\|k\|\|$	$\|\|k\|\|$	$(n+2)\|\|k\|\|$		$(n+1)\|\|s\|\|$
		[40]	$O(nt)$ if $m \leq t$ $O((n+m-t)t)$ otherwise	$O(\max(m^3, mt^2))$	$O(t)$			$n\|\|d\|\|$ if $m \leq t$ $(n+m-t)\|\|d\|\|$ otherwise	$n\|\|k\|\|$	$4\|\|k\|\|$			$n\|\|s\|\|$
		[41]	$O(m+n)$	$O(\max(m, t^2))$	$O(t)$			$(m+n)\|\|d\|\|$	$n\|\|k\|\|$	$4\|\|k\|\|$			$n\|\|s\|\|$
		[50]	$O(mnt)$	$O(mt^2)$		$O(mt)$		$mn\|\|d\|\|/t$	$n\|\|k\|\|$				$n\|\|s\|\|$
		[51]	$O(\max(m^2, t^2))$	$O(\max(m^2, t^2))$		$O(mt)$		$(n+m+t)\|\|d\|\|$	$n\|\|k\|\|$				$(n+m)\|\|s\|\|$
		[42]	$O(mnt^2)$	$O(mnt^2)$	$O(t)$			$mn\|\|d\|\|/t$	$n\|\|k\|\|$	$2n\|\|k\|\|$			$n\|\|s\|\|$
		[43]-I	$O(nt)$ if $m \leq t$ $O((n+m-t)t)$ otherwise	$O(\max(m^3, mt^2))$	$O(t)$			$n\|\|d\|\|$ if $m \leq t$ $(n+m-t)\|\|d\|\|$ otherwise	$n\|\|k\|\|$				$2n\|\|s\|\|$
		[43]-II	$O(m+n)$	$O(\max(m, t^2))$	$O(t)$			$(m+n)\|\|d\|\|$	$n\|\|k\|\|$	$4\|\|k\|\|$			$2n\|\|s\|\|$
		[44]	$O(mnt)$	$O(mt^2)$	$O(t^2)$			$m(n-1)(t^2-t)\|\|d\|\|/(2t)$	$n\|\|k\|\|$				$n\|\|s\|\|$
	Group 9	[46]	$O(mt)$	$O(mt^2)$	$O(t)$	$O(mt)$		$4m\|\|d\|\|$	$n\|\|k\|\|$	$(t+3)\|\|k\|\|$	$2\|\|k\|\|$		$n\|\|s\|\|$
		[45]	$O(mt)$	$O(mt^2)$	$O(t)$	$O(mt)$		$3m\|\|d\|\|$	$n\|\|k\|\|$	$(t+3)\|\|k\|\|$	$2\|\|k\|\|$		$n\|\|s\|\|$
		[47]	$O(nt)$ if $m \leq t$ $O((n+m-t)t)$ otherwise	$O(\max(m^3, mt^2))$	$O(\max(m^2, mt))$			$n\|\|d\|\|$ if $m \leq t$ $(n+m-t)\|\|d\|\|$ otherwise	$n\|\|k\|\|$				$n\|\|s\|\|$ if $m \leq t$ $(n+m-t)\|\|s\|\|$ otherwise
		[48]	$O(mnt)$	$O(mt^2)$	$O(mt)$			$m(n+1)\|\|d\|\|$	$n\|\|k\|\|$	$2\|\|k\|\|$			$(n+m)\|\|s\|\|$
		[49]	$O(mt^2)$	$O(mt^2)$	$O(mnt)$	$O(m)$		$mt\|\|d\|\|$	$n\|\|k\|\|$	$mt\|\|k\|\|$			$(m+mnt)\|\|s\|\|$
		[52]	$O(\max(nt, m))$	$O(mt^2)$	$O(t^2)$	$O(mt)$		$3m\|\|d\|\|$	$n\|\|k\|\|$	$3\|\|k\|\|$	$\|\|k\|\|$		$t\|\|s\|\|$
	Group 10	[53]	$O(mt)$	$O(mt^2)$	$O(mt)$		$\frac{mn\|\|d\|\|}{t}$ or $bn\|\|d\|\|$		$nt\|\|k\|\|$			$m\|\|s\|\|$	

Other parameters n and t are small, because they relate to the number of PTs, i.e., the number of CSPs, which is limited in practice. Moreover, some SSSs such as [26, 28, 37] cannot assign a big value to parameters n and t because both cannot be greater than the size of a data piece.

Normally, the time complexity of data encryption is a little higher than that of data decryption. In contrast, in most MSSSs type I, the time complexity of data encryption is clearly lower than that of data decryption, because they encrypt several data pieces at once but decrypt each data piece independently. Overall, [38]'s data encryption/decryption complexities are the lowest: $O(\max(m, t^2))$. Moreover, VSSSs and VMSSSs must verify the correctness of keys and/or data. The time complexity of data/key verification is generally lower than that of data encryption/decryption. Moreover, The time complexity of key verification is generally lower than that of data verification. Several schemes share the same lowest key verification complexity $O(t)$, but only [38] achieves the lowest key verification complexity: $O(n)$.

Almost all SSSs require a storage volume about n times that of original data to store shares. Some SSSs propose solutions to minimize share volume. We categorize them into three classes. First, [26, 28, 37] split data before encryption. Hence, share volume is only n/t times that of original data. However, since the size of transformed data decreases when t increases, the value of t cannot be bigger than the size of a data piece. Second, [32, 33, 42, 50, 53] construct n shares per data block. Hence, share volumes of [32] and [33, 42, 50, 53] are only 1 and n/t times that of original data, respectively. Third, [38, 45, 46, 52] encrypt data pieces independently, but they construct fewer than n shares per secret. Hence, share volumes of [38],[46],[45, 52] are only 1, 4, 3 and 3 times that of original data, respectively. Overall, [32, 38] require the lowest storage volume (the same as original data volume) to store shares. However, [32] does not support data availability and [38] supports data availability only when the NB is accessible. Share volumes of [26, 28, 53] are a little higher than that of the lowest-share-volume approaches [32, 38] if n is close to t, but they do support data availability.

Some SSSs require extra storage to store keys. Most of them use only n or nt keys to encrypt all data pieces. Thus, they use only a little storage volume. However, key volumes of [24, 49, 81] are greater than the original data volume (about t^2 [24], $2n$ [81] and t [49] times data volume) because they use different key sets to encrypt a data piece. Hence, their overall storage volume (shares, keys and signatures) are greater than that of other SSSs, and thus incurs a higher storage cost.

Finally, all VSSSs and VMSSSs require extra storage to store signatures. The number of signatures is about the number of keys, data or shares, depending on the verified data type. Thus, overall signature volume is lower than share volume in all

VSSSs and VMSSSs. However, if signatures are too small, verification accuracy becomes weak. Overall, [52] requires the lowest storage volume to store signatures. Hence, its overall storage volume is lower than n times that of original data. In contrast, [35, 82] require the greatest storage volume to store signatures. Hence, their overall storage volume turn to be greater than other SSSs, i.e., the same as [24, 49, 81], which construct a huge key volume.

2.1.11.5 Conclusion

Classic SSSs handle data security and availability with a high sharing/reconstruction time and storage cost. MSSSs share data at once and reduce both costs. In addition, MSSSs type I support data availability by using a NB, but are vulnerable if the NB is attacked. Hence, to share data with MSSSs type I in the cloud, the NB should be to a CSP that guarantees high security and availability.

In addition, VSSSs and VMSSSs can verify the correctness of either or both of data and keys, but these operations induce additional time overhead and require to store signatures in addition to shares. Outer code verification still necessitates to trust PTs, because it is done at PTs'. Moreover, since almost all VMSSSs are also MSSSs type I, their total storage volume (keys, shares and signatures) is still lower than n times that of original data. Only [53] verifies the correctness of both data and keys. Although it is an MSSSs type II, its total storage volume is only about twice that of original data. Moreover, its data sharing complexity is also reasonable, i.e., $O(mt)$ while most SSSs have a cubic sharing complexity.

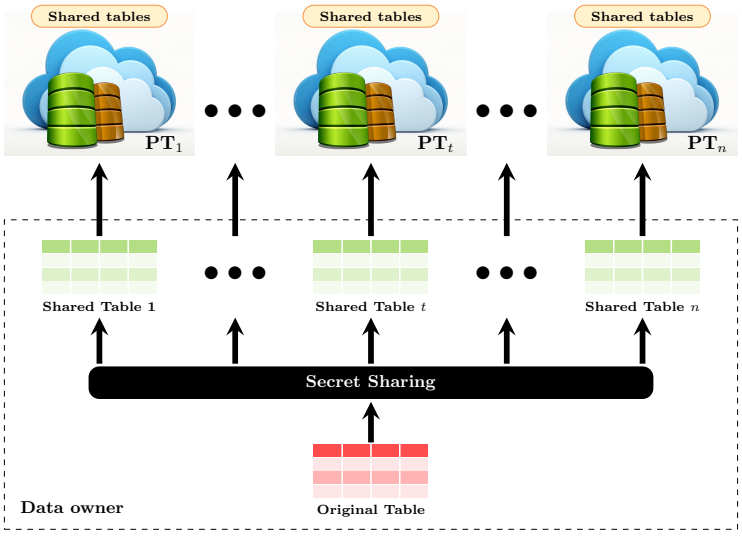
Finally, some SSSs support features such as updates, search operations, aggregation operations, etc. These features help minimize computation cost at the user's side and reduce communication overhead. Only one SSS [24] supports all three operation types, and none can handle composite operations on shares. Among SSSs that support search or aggregation operations, again only [53] optimizes both storage cost and data sharing time.

2.2 Secret Sharing-Based Secure Database and Data Warehouse

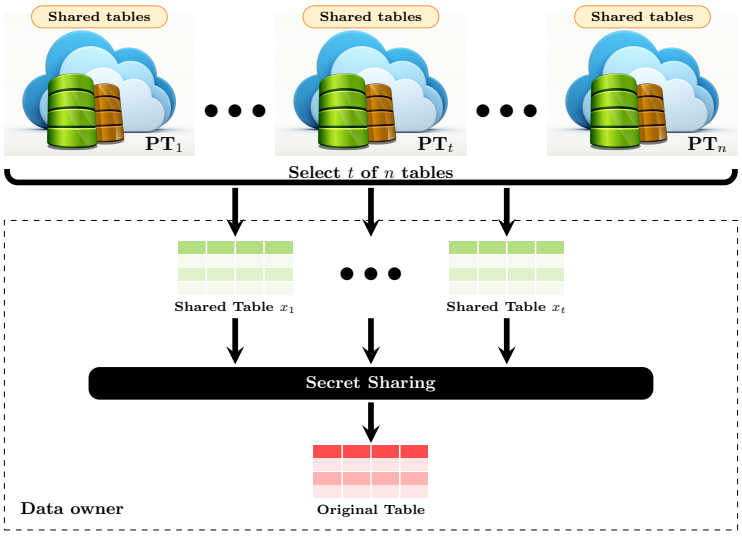
Almost all secret sharing-based approaches [54–60] exploit classical secret sharing for securing DBs and DWs. We classify them into two families: classical DB sharing and index-based DB sharing.

2.2.1 Classical Database Sharing

In this family of approaches, each table is encrypted into n shared tables, each of which is stored at one given PT (Figure 2.8(a)). Recall that only t of n shared tables are sufficient to reconstruct the original table (Figure 2.8(b)).



(a) Sharing process



(b) Reconstruction process

FIGURE 2.8: Classical database sharing processes

[54] exploits Sharmir's SSS [23] to secure DWs. Several keys and attribute values in each records are encrypted at once. First, keys are encrypted with a hash function. Then, attribute values in the same record are encrypted with a hash function. Finally, encrypted records are encrypted again by [23]. This approach can perform aggregate operations (sum and average), as well as join and intersection queries, on shares. Exact

match and range operations are handled through an operator needing sorted data with preaggregation before sharing.

[55] exploits [23] to secure DBs. It encrypts and decrypts data from multiple data-sources with polynomial and Lagrange interpolation, respectively. However, polynomial coefficients are constructed with several distinct hash functions. This approach can perform exact match, range and aggregation operations (equality and inequality, sum, average, count, maximum, minimum and median) on shares. Moreover, it handles intersection and join queries on shares with the help of indexes. Since each data piece is encrypted and decrypted independently, this approach can also update (insert, update and delete) shares directly.

[56] also exploits [23] to secure DBs. In addition to [55], it can share both numerical and nonnumerical data. To share data, the coefficients of [23] are ranked with respect to data values. At each PT's, the rank of share values is the same as that of data values. This approach allows exact match operations, range operations and some aggregation operations (equality and inequality, sum, average, count, maximum, minimum and median) on shares. Moreover, it can also perform join queries on shares if two joined attributes share the same domain.

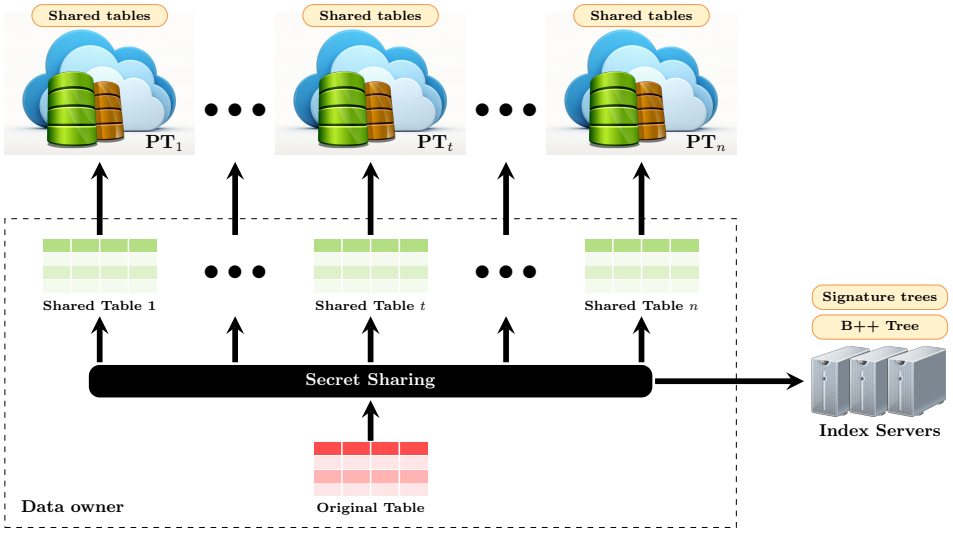
Finally, [57] extends from [56]. It proposes a new solution for ranking the coefficients of [23]. This solution helps enforce higher data sharing efficiency and higher security (strong enough to tolerate statistical attacks). However, it handles only on integers.

2.2.2 Index-based Database Sharing

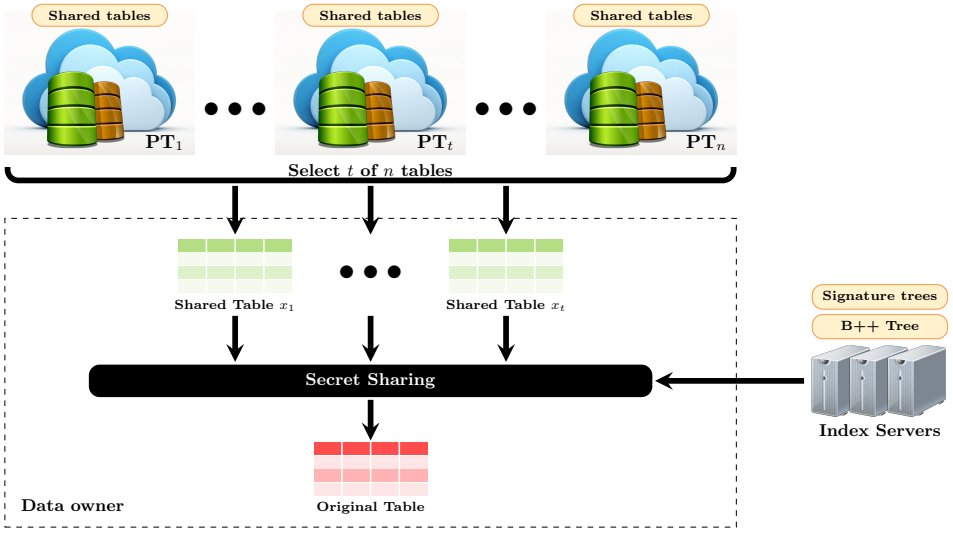
In this family, one or more index servers, located at $\{PT_i\}_{i>n}$, store B+ tree indices and signatures (Figure 2.9). Index servers require higher security and computing power than other nodes, and a secure connection to the user.

[58] exploits [23] to secure DBs and handle data integrity. It proposes a new data verification process. Inner signatures are constructed with a discrete logarithm and are stored in Merkle hash trees at the index server's. Then, data and their inner signatures are independently encrypted with [23]. This approach allows some aggregation operations (sum and average) and updates (insert and delete) on shares.

As [58], [59] encrypts data with [23]. In addition, B+ trees are built from unencrypted data and stored at the index server's. B+ trees help perform exact match queries, range queries and some aggregation queries (maximum, minimum and median) on shares.



(a) Sharing process



(b) Reconstruction process

FIGURE 2.9: Index-based database sharing processes

[60] extends from [59] to handle data integrity. Inner signatures are constructed with a hash function. Then, a pair (data value, inner signature) is encrypted at once with a semi-random polynomial. Thus, inner signatures are hidden in shares.

Finally, unlike other approaches, [21] encrypts data blocks to keep global share volume well down under n times the original data volume. Inner signatures are constructed with Message Authentication Code (MAC). Then, inner signatures and data in several rows are encrypted at once with linear equations, unfortunately implying that no aggregation operation can be handled on shares. However, this approach allows exact match queries and updates on shares through B+ trees.

2.2.3 Discussion

The features of all secret sharing-based DB approaches are summarized in Table 2.6. First, all approaches handle data privacy and availability through secret sharing. However, since existing SSSs allow a coalition or the compromission of at least t PTs to reconstruct the secret, intruders can break the secret if they hold enough shares. Moreover, no approach can share new data if any PT fails, because existing SSSs must share the secret at all n PTs. Only some approaches [21, 58–60] can handle data integrity. They use inner code verification to check whether PTs are malicious. However, none uses outer code verification to detect incorrect or erroneous (lost, damaged, alternative...) data before decryption and prevent useless data transfers. Thus, simultaneously handling data privacy, availability and integrity must be addressed.

TABLE 2.6: Comparison of database sharing approaches

Features and costs	**[56]**	**[54]**	**[55]**	**[57]**	**[59]**	**[60]**	**[58]**	**[21]**
Data privacy	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Data availability	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ability in case PTs fail, to								
- Query shares	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
- Update shares	No	No	No	No	No	No	No	No
Data integrity								
- Inner code verifying	No	No	No	No	No	Yes	Yes	Yes
- Outer code verifying	No	No	No	No	No	No	No	No
Target	DBs	DWs	DBs	DBs	DBs	DBs	DBs	DBs
Data sources	Single	Multi	Multi	Single	Single	Single	Single	Single
Data types	Positive integers, Characters, Strings	Positive integers	Integers	Integers	Positive integers	Positive integers	Positive integers	Positive integers
Shared data access								
- Updates	No	No	Yes	Yes	Yes	Yes	Yes	Yes
- Exact match queries	No	Yes	Yes	Yes	Yes	Yes	No	Yes
- Range queries	No	Yes	Yes	Yes	Yes	Yes	No	Yes
- Aggregation queries on one attribute	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
- Aggregation queries on two attributes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
- Grouping queries	No	No	No	No	No	No	No	No
Complexity								
- Data storage w.r.t. original data volume	$\geq n$	$\geq 2n$	$\geq 2n$	$\geq n$	$\geq n$ +1 (B++ tree)	$\geq n$ +1 (B++ tree)	$\geq 2n$ +1 (hash tree)	$\geq n/t$ $+n/t$ (B++ tree) + signatures
- Sharing time	$O(nt)$	$O(nt)$	$O(nt)$	$O(nt)$	$O(nt)$	$O(nt)$	$O(nt)$	$O(n)$
- Reconstruction time	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t)$

With respect to DB features, all approaches focus on sharing (positive) integers on DBs or DWs. Only one approach [56] can share nonnumerical data such as characters and strings. Moreover, to access shared data, all approaches allow at least one query type (e.g., exact match, range and aggregate queries) on shares. However, none handle OLAP operators and even GROUP BY clauses. Secret sharing-based approaches should definitely help share both numerical and nonnumerical data, and allow both basic DB querying and OLAP analysis.

With respect to costs, most approaches consume an overall storage volume that is at least n times that of original data. The complexity for sharing and reconstructing

data is polynomial. Only one approach reduces storage volume well under n times that of original data [21], and thus decreases monetary storage cost in the pay-as-you-go paradigm. Moreover, its data sharing/reconstruction complexity also falls down to be linear. However, data access costs (computation and data transfer costs) are still high because queried data must be wholly reconstructed before aggregation is computed. Thus, costs should be further minimized without losing the above-mentioned security and DB features.

Finally, most secret sharing-based DB approaches encrypt data with Shamir's SSS [23]. Some SSSs [23, 24, 53, 81] reviewed in Section 2.1 support updates and aggregation operations. Moreover, [24] also allows search operation on shares. [53] requires an overall storage volume that is lower than n times the original data volume, and also enforces data integrity. Thus, these SSSs are good choices for securing DWs. However, they still do not support above-mentioned security and DB features. Thus, SSSs that simultaneously handle all security and DB features must be devised. Moreover, costs should also be mastered.

Chapter 3

bpVSS: Base-p Verifiable Secret Sharing

3.1 bpVSS Principle

bpVSS is a (t, n) base-p verifiable secret sharing scheme belonging to the first family of secure distributed database approaches identified in Section 2.2. As all similar approaches, bpVSS shares data over n CSPs, t of which are necessary to reconstruct original data. Each CSP only stores part of the shares, which are not exploitable, neither by the CSP nor any intruder, because they have been transformed by a mathematical function. Though performing computations on shares is possible, i.e., data need not be decrypted, it yields meaningless results. It is only when results are mathematically transformed back at the user's that they can be reconstructed into global, meaningful information. Individual shares and computed results being encrypted, network transfers to and from CSPs are thus safe. Hence, privacy is achieved at any point outside of the user's (network, providers). Moreover, availability is also enforced because data can still be if $n - t$ CSPs disappear.

In addition, there are three main novelties in bpVSS (Figure 3.1). First is shared data volume and thus storage cost are minimized from that of other approaches. The volume of individual share is controlled to be significant lower than the volume of its data. Hence, total share volume at all n CSPs is lower than n times of data volume. For example, in Figure 3.2, the volume of unit-price's share in record #124 of PRODUCT at each CSP (29, i.e.,5 bits; 19, i.e.,5 bits; 39, i.e.,6 bits; 35, i.e.,6 bits) is lower than its original volume (135, i.e.,8 bits).

FIGURE 3.1: Five main benefits of bpVSS

Second, unlike any approaches, two signature types (inner and outer signatures) are incorporated into bpVSS to verify the honesty of CSPs and the correctness of data and shares. Inner signatures created from an homomorphic function help verify data correctness in case some CSPs are not honest. Moreover, they are hidden in shares (they are part of shares), and thus no extra storage is required to store inner signatures. Outer signatures created from a one-way function help verify incorrect or erroneous shares before reconstructing data. Hence, no erroneous shares are transferred back to the user for reconstruction. This helps minimize both data transfer cost and computing cost on the user's side. Unlike inner signatures, outer signatures are stored in extra attributes in shared tables. For example, in Figure 3.2, attributes Sig-Pname, Sig-PDescr and Sig-UPrice store the outer signatures of the shares of attributes ProName, ProDescr and UnitPrice, respectively.

Third, as some similar approaches, bpVSS allows data analysis on shares. To optimize computing cost at the user's side and data transfer cost when analyzing data, exact match queries and aggregation functions can be directly performed on shares. Then, only matching aggregated results are transferred back to the user for reconstruction. In addition, all basic OLAP operations (roll-up, drill-down, some slice and dice, pivot and drill-across) can also apply directly on shares at the CSPs', with results being reconstructed at the user's.

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	Shirt	Red	1	135
125	Shoe	*NULL*	2	142
126	Ring	*NULL*	1	142

(a) Original data

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{31,10,21,36,16}	{1,0,1,1,1}	{27,26,22}	{2,1,2}	1	29	2
125	{31,10,17,26}	{1,0,2,1}	*NULL*	*NULL*	2	36	0
126	{27,21,13,6}	{2,1,3,1}	*NULL*	*NULL*	1	36	0

(b) Shares at CSP_1

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{40,27,10,28,21}	{0,2,0,3,1}	{33,27,20}	{3,2,0}	1	19	1
125	{40,27,31,27}	{0,2,1,2}	*NULL*	*NULL*	2	23	2
126	{33,10,24,20}	{3,0,4,0}	*NULL*	*NULL*	1	23	2

(c) Shares at CSP_2

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{30,13,20,28,23}	{0,3,0,3,3}	{29,17,16}	{4,2,1}	1	39	0
125	{30,13,19,17}	{0,3,4,2}	*NULL*	*NULL*	2	45	0
126	{29,20,18,12}	{4,0,3,2}	*NULL*	*NULL*	1	45	0

(d) Shares at CSP_3

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{44,14,21,40,22}	{4,4,1,0,2}	{39,27,22}	{4,2,2}	1	39	0
125	{44,14,23,27}	{4,4,3,2}	*NULL*	*NULL*	2	48	0
126	{39,21,18,9}	{4,1,3,4}	*NULL*	*NULL*	1	48	0

(e) Shares at CSP_4

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{34,16,27,42,22}	{4,4,2,2,2}	{30,32,28}	{0,2,3}	1	35	2
125	{34,16,23,32}	{3,1,3,2}	*NULL*	*NULL*	2	42	0
126	{30,27,19,12}	{0,2,4,2}	*NULL*	*NULL*	1	42	0

(f) Shares at CSP_5

FIGURE 3.2: Sample original and shared data

To achieve all benefits, we process each attributed value independently as follows. Suppose we want to share a piece of data $d_{jkl} = 135$, e.g., the unit price of product #124 in Figure 3.2 where $n = 5$, $t = 4$ and $p = 7$. We also need to share its inner signature s_in_{jkl} to enforce data integrity. Usually, secret sharing schemes create shares from polynomials and decimal number. In contrast, we create shares from n distinct random t-variables linear equations $f_i(x_1, \cdots, x_t)_{i=1\cdots n}$, inner signature $s\_in_{jkl} = 2$ and $t-1$ digits of base-p number ($d_{jkl} = 135_{10} = 252_7$). Next, outer signatures are created. Then, shares and their outer signatures are distributed to CSPs. The share at CSP_i is $e_{ijkl} = f_i(d\_s_{jkl1}, \cdots, d\_s_{jkl(t-1)}, s\_in_{jkl})$. To control share volume, all f_i's coefficients and f_i's arguments are controlled to lower than user-defined parameter p. Thus, in normal case, volume of share $e_{ijkl} \in]-tp^2, tp^2[$ is lower than volume of the original data $d_{jkl} \in]-p^t, p^t[$. However, the share volume is greater than the data volume only if $d_{jkl} < p$.

To reconstruct data d_{jkl}, d_{jkl}'s shares are verified first by their outer signatures at CSPs'. If all t shares are correct, inner signature and $t-1$ digits of base-p number are reconstructed at the user's. Then, data is transformed back to decimal number from $t-1$ reconstructed digits of base-p data only if the inner signature matches all digits.

Note that the summing can be reconstructed from the summing shares as normal data because of linear equation property. Moreover, thanks property of block cipher, the matching can be operated on shares to filter requested shares for reconstruction. This help reduce both data transfer cost and computation cost.

The remainder of this chapter is organized as follows. Section 3.2 details bpVSS' sharing and reconstructing processes. Section 3.3 describes the way we share data of various types. Section 3.4 describes the way data warehouses and data cubes can be shared. Section 3.5 presents the loading, backup and recovery processes of data warehouses and data cubes. Sections 3.6 describes how we preform queries and OLAP operations on shared data warehouse. We finally close this chapter by a summary of highlights in Section 3.7.

3.2 Data Sharing and Reconstruction

In bpVSS, DB attribute values, except NULL values and primary or foreign keys, are encrypted and shared in relational databases at CSPs'. Keys help both match records in the data reconstruction process and perform JOIN and GROUP BY queries. For example, in Figure 3.2, key (ProNo and CategoryID) values and NULL values in attribute ProDescr are not encrypted. Moreover, ProNo helps matching records at all CSPs'. To protect any sensitive primary key such as a social security number, an unencrypted sequential integer key is created whereas the sensitive key is encrypted as normal data.

bpVSS classically operates on base-p integers, but real data can be of other types. Therefore, any piece of data is transformed into integer(s) which is/are transformed into a base-p integer before encryption. Such transformations are explained in Section 3.3. Any decimal integer can be transformed into a base-p integer because any decimal integer d_{jkl} in interval $]-p^t, p^t[$ can be written as a base p expansion in polynomial form (Equation 3.1) such that d_s_{jklh} is a digit of base-p integer. For example, 135 and -135 can be expanded in $2\times7^2+5\times7+2$ and $-2\times7^2-5\times7-2$, respectively, where $p=7$. Hence, 2, 5 and 2 are three digits of 135 in base 7; and -2, -5 and -2 are three digits of -135 in base 7.

$$d_{jkl}=\sum_{h=1}^{t-1} d\_s_{jklh}\times p^{(h-1)} \Leftrightarrow d\_s_{jklh}=\begin{cases} \lfloor d_{jkl}/p^{(h-1)}\rfloor \bmod p & \text{if } d_{jkl}\geq 0 \\ -(\lfloor |d_{jkl}|/p^{(h-1)}\rfloor \bmod p) & \text{if } d_{jkl}<0 \end{cases} \tag{3.1}$$

In bpVSS, each value d_{jkl} of attribute A_{jl} from record R_{jk} in table T_j is encrypted and reconstructed independently. Figure 3.3 shows the flowchart of sharing d_{jkl}. First, d_{jkl} is rewritten in base p. Then, inner signature s_in_{jkl} is computed from $t-1$ base-p digits $\{d\_s_{jklh}\}_{h=1\cdots(t-1)}$ by homomorphic encryption [6]. All base-p digits $\{d\_s_{jklh}\}_{h=1\cdots(t-1)}$ and their inner signature s_in_{jkl} are transformed into n shares with n distinct random t-variables linear equations $f_i(x_1, \cdots, x_t)_{i=1\cdots n}$. Outer signatures are created with a user-defined one-way function. Share e_{ijkl} and its outer signature s_out_{ijkl} are distributed into attributes A_{jl} and A_sout_{jl}, respectively, in record ER_{ijk} of table ET_{ij} at CSP_i.

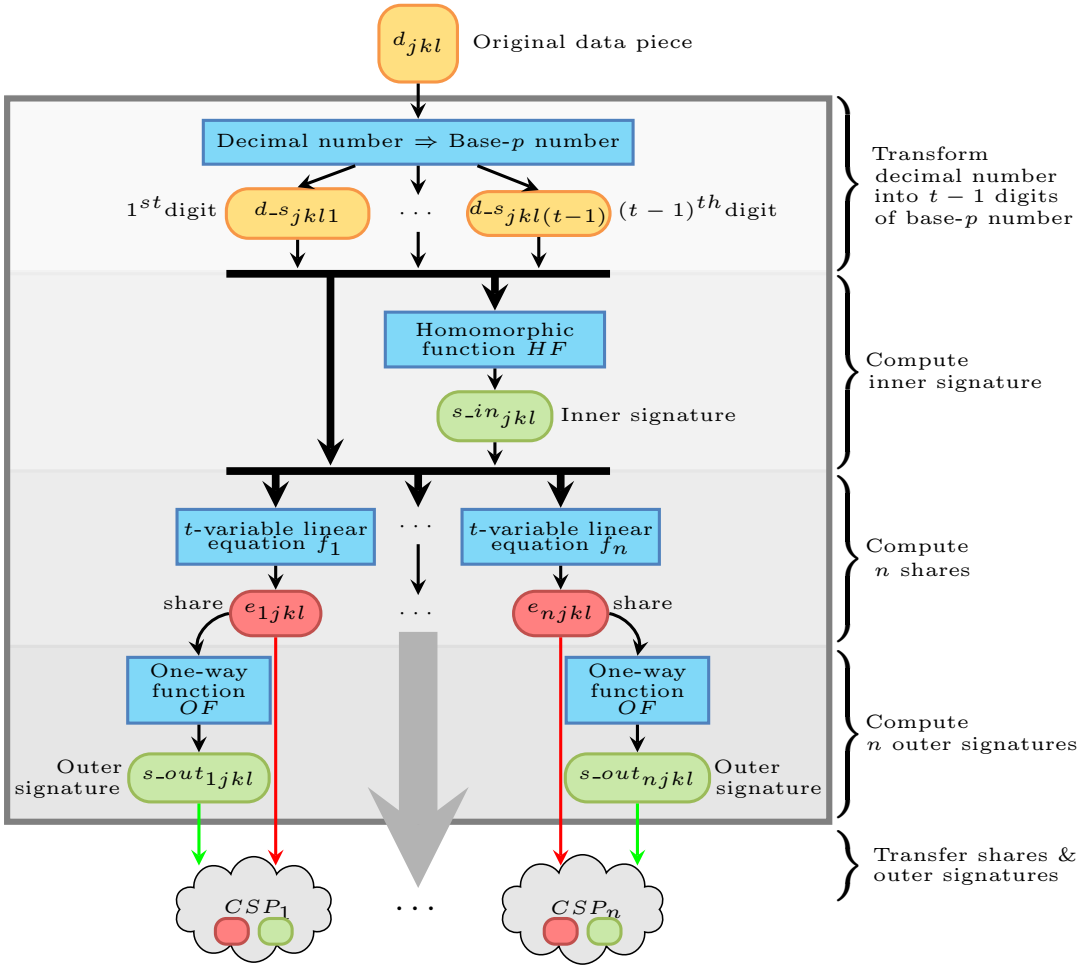


FIGURE 3.3: Data sharing process

Finally, Figure 3.4 shows the flowchart of reconstructing data d_{jkl}. All $t-1$ base-p digits $\{d\_s_{jklh}\}_{h=1\cdots(t-1)}$ and the inner signature are reconstructed from t shares by solving a linear equation system. Then, data d_{jkl} is transformed back into decimal only if all base-p digits are correct. The correctness of shares and base-p digits is verified by outer and inner signatures, respectively. Share e_{ijkl} and outer signature s_out_{ijkl} match if the share is correct. Inner signature s_in_{jkl} matches with all base-p digits

$\{d\_s_{jklh}\}_{h=1\cdots(t-1)}$ only if all t CSPs in RG return correct shares and all base-p digits are correct.

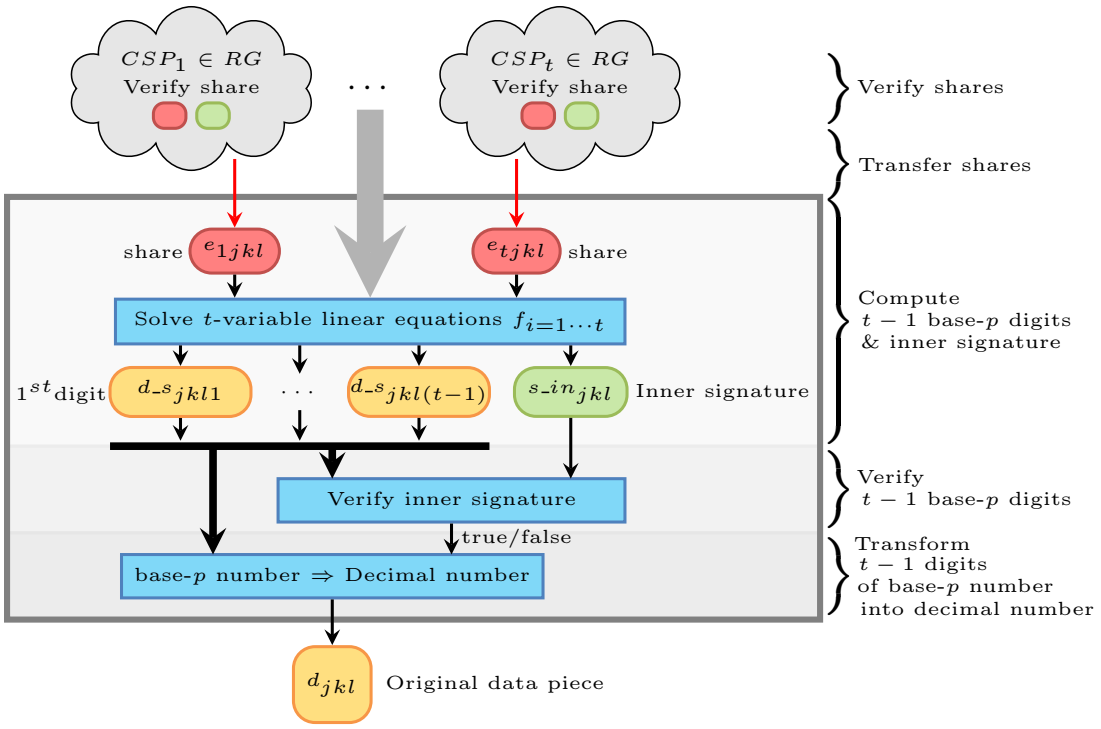


FIGURE 3.4: Data reconstruction process

The remainder of this section is organized as follows. Section 3.2.1 specifies parameter settings and functions related to the data sharing and reconstruction processes. Sections 3.2.2 and 3.2.3 describe each step of the data sharing and reconstruction processes, respectively.

3.2.1 Initialization Phase

1. Set values of n, t and p (a prime number) such that $p^t > d > -p^t$ and $n \geq t$.
2. Set values of CSP identifiers $ID_{i=1..n} \in]0, p[$. Each $ID_{i=1..n}$ must be unique.
3. To create inner signatures, define a $(t-1)$-variables one-way homomorphic function $HF(a_1, \cdots, a_{t-1})$. HF's parameters are base-p digits. HF returns an inner signature. HF must also support an homomorphic property for summation and subtraction $HF(a_1, \cdots, a_{t-1}) \pm HF(b_1, \cdots, b_{t-1}) = HF(a_1 \pm b_1, \cdots, a_{t-1} \pm b_{t-1})$.
4. To create outer signatures, define one-variable one-way function $OF(a)$, where $a \in]-tp^2, tp^2[$. OF inputs a share and outputs its outer signature. Outer signatures are small integers.

5. To create shares, create n distinct random t-variables linear equations $f_i(x_1, \cdots, x_t)_{i=1\cdots n}$ (Equation 3.2),

$$y = f_i(x_1, \cdots, x_t) = \sum_{h=1}^{t} x_h \times b_{ih} \tag{3.2}$$

where b_{ih} is the h^{th} random coefficient seeded at ID_i, $b_{ih} \in [0, p[$ and $f_{i1} \neq f_{i2}$ if and only if $i1 \neq i2$. f_i's parameters are $t-1$ digits of base-p integer and a inner signature. f_i returns a share of base-p integer stored at CSP_i.

Note that all parameters and functions, except one-way function OF, must be secret and highly protected by the user. However, OF is exposed to CSPs in the outer verification process.

3.2.2 Data Sharing Process

Any attribute value d_{jkl} of record R_{jk} is encrypted independently as follows.

1. Compute $t-1$ base-p digits $\{d\_s_{jklh}\}_{h=1\cdots(t-1)}$ such that
   - $d\_s_{jklh} = \lfloor d_{jkl}/p^{(h-1)} \rfloor \bmod p$ if d_{jkl} is a non-negative integer,
   - $d\_s_{jklh} = -(\lfloor |d_{jkl}|/p^{(h-1)} \rfloor \bmod p)$ if d_{jkl} otherwise.
2. Compute d_{jkl}'s inner signature s_in_{jkl} with homomorphic function HF: $s\_in_{jkl} = HF(d\_s_{jkl1}, \cdots, d\_s_{jkl(t-1)})$.
3. Compute the set of d_{jkl}'s shares $\{e_{ijkl}\}_{i=1\cdots n}$ such that $e_{ijkl} = f_i(d\_s_{jkl1}, \cdots, d\_s_{jkl(t-1)}, s\_in_{jkl})$, and distribute each share e_{ijkl} to attribute A_{jl} in record ER_{ijk} at CSP_i.
4. Compute the set of d_{jkl}'s outer signatures $\{s\_out_{ijkl}\}_{i=1\cdots n}$ with one-way function OF such that $s\_out_{ijkl} = OF(e_{ijkl})$, and distribute each signature s_out_{ijkl} to attribute A_sout_{jl} in record ER_{ijk} at CSP_i.

Following this routine, record R_{jk} is shared into n records $ER_{1jk}, \cdots, ER_{njk}$ at n CSPs. The relationship between R_{jk} and ER_{ijk} is maintained through primary keys $pk_{jk} = pk_{ijk}$.

3.2.3 Data Reconstruction Process

Any attribute value d_{jkl} is reconstructed from shares e_{ijkl} and outer signatures s_out_{ijkl} stored at $CSP_i \in RG$ where RG is group of t CSPs selected to reconstruct data. There are two phases to reconstruct original data: the initialization phase and the actual reconstruction phase.

3.2.3.1 Initialization Phase

In this phase, share correctness is verified and a matrix C that is used in the reconstruction phase is created as follows.

1. Verify information at all $CSP_i \in RG$. At each CSP's, only shares to be decrypted are verified for correctness. Share e_{ijkl} is correct if $s\_out_{ijkl} = OF(e_{ijkl})$. In case of error at CSP_i , then another CSP is selected and correctness is verified again.

2. At the user's, matrix B is created from f_i's random coefficients seeded at ID_i of $CSP_i \in RG$. $B = [b_{ih}]_{(t \times t)}$, where b_{ih} is the h^{th} coefficient of f_i. Then, C is computed as $C = B^{-1}$. Let c_{hi} be the entry on the h^{th} row and the i^{th} column of matrix C.

3.2.3.2 Reconstruction Phase

To reconstruct attribute value d_{jkl} and verify its correctness, share e_{ijkl} of $CSP_i \in RG$ is transferred to the user and reconstructed as follows.

1. Compute $t-1$ base-p digits $\{d\_s_{jklh}\}_{h=1\cdots(t-1)}$ (Equation 3.3) and d_{jkl}'s inner signature s_in_{jkl} (Equation 3.4).

$$d\_s_{jklh} = \sum_{\forall i: CSP_i \in RG} c_{hi} \times e_{ijkl} \tag{3.3}$$

$$s\_in_{jkl} = \sum_{\forall i: CSP_i \in RG} c_{ti} \times e_{ijkl} \tag{3.4}$$

2. If $s\_in_{jkl} = HF(d\_s_{jkl1}, \cdots, d\_s_{jkl(t-1)})$, then all base-$p$ digits $\{d\_s_{jklh}\}_{h=1\cdots(t-1)}$ are correct. In case of error, the user can reconstruct data from the shares of a new RG.

3. Compute value of decimal integer d_{jkl} (Equation 3.5).

$$d_{jkl} = \sum_{h=1}^{t-1} d\_s_{jklh} \times p^{(h-1)} \tag{3.5}$$

3.3 Data Type Management

3.3.1 Sharing Various Types of Data

To handle the usual data types featured in databases, we encrypt and handle each attribute value independently. Each attribute value is first transformed into one or more integers depending on data type. Then, the integers are encrypted by bpVSS. Figure 3.5 summarizes the transformation process of data of various types into integers. The following sections detail how original data are transformed.

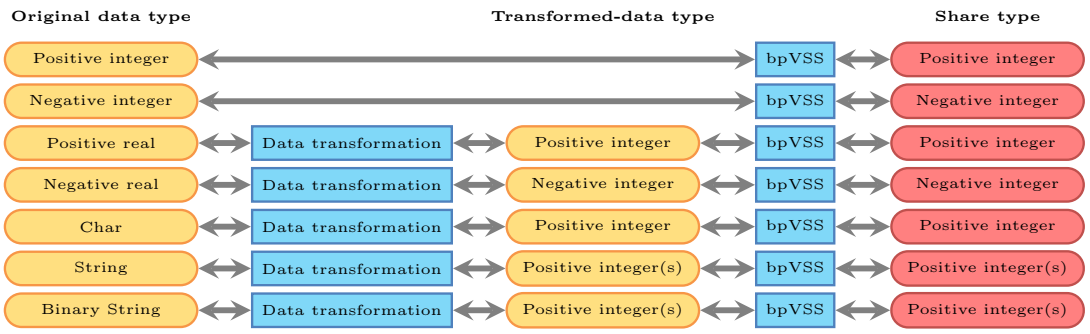


FIGURE 3.5: Data type transformation process

3.3.1.1 Integers, Dates and Timestamps

An integer, date or timestamp I is directly encrypted to n shares with bpVSS. Share type is the same as I. If I is positive (respectively, negative) integer, its share is a positive (respectively, negative) integer.

3.3.1.2 Reals

A real R is transformed into an integer I by multiplication. For example, let R be stored in numeric format (v, s), where v is a precision value and s a scale value. Then, R is transformed into $I = R \times 10^s$. I can then be encrypted as any integer.

3.3.1.3 Characters

A character L is transformed into a positive integer I through its ASCII code. For example, let L be 'A'. L is transformed into $I = 65$. I can then be encrypted as any integer.

3.3.1.4 Character Strings

A string S is transformed into a set of positive integers $\{I_a\}_{a=1\cdots\|S\|}$ where $\|S\|$ is the length of S, using the ASCII code of each character in S. For example, let S be 'ABC'. Then, S is transformed into $\{I_a\}_{a\in\{1,2,3\}} = \{65, 66, 67\}$. After transformation, each character I_a is encrypted independently as any integer.

3.3.1.5 Binary Strings

A binary string B is transformed into a set of positive integers $\{I_a\}_{a=1\ldots\|B\|/(\|tp\|-1)}$ where $\|B\|$ is the length of B and $\|tp\|$ is a number of bits of $(t-1)\times p$. For example, let $B = 1100010$ and $\|tp\| = 6$ bits ($t = 4$, $p = 21$ and $(t-1)\times p = (4-1)\times 21 = 63$ or 111111_2). Then B is split into two smaller binaries: 11 and 00010 sizing $\|tp\| - 1 = 5$, which are then transformed into $\{I_a\}_{a\in\{1,2\}} = \{3, 2\}$. After transformation, each I_a is encrypted independently as any integer.

3.3.2 Example: Sharing and Reconstructing an Integer

Let us refer back to Figures 3.2. Value $d_{jkl} = 135$, e.g., unit price of product #124 is shared and is reconstructed as follows.

3.3.2.1 Initialization Phase

1. Let parameters be assigned as follows: $p = 7$, $n = 5$ and $t = 4$.
2. Let CSP identifiers be assigned as follows: $ID_1 = 1$, $ID_2 = 3$, $ID_3 = 4$, $ID_4 = 7$ and $ID_5 = 2$.
3. To create inner signatures, let us select the following 3-variables homomorphic function: $HF(a_1, \cdots, a_{t-1}) = \sum_{h=1}^{t-1} a_h \bmod 7$.
4. To create outer signatures, let us select the following one-variable one-way function: $OF(a) = a \bmod 3$.

5. To create shares, let us select five distinct random 4-variables linear equations $f_i(x_1, \cdots, x_4)_{i=1\cdots5}$ as follows:

   (a) $y = f_1(x_1, x_2, x_3, x_4) = x_1 \times 0 + x_2 \times 3 + x_3 \times 3 + x_4 \times 4$,

   (b) $y = f_2(x_1, x_2, x_3, x_4) = x_1 \times 4 + x_2 \times 1 + x_3 \times 0 + x_4 \times 3$,

   (c) $y = f_3(x_1, x_2, x_3, x_4) = x_1 \times 0 + x_2 \times 5 + x_3 \times 6 + x_4 \times 1$,

   (d) $y = f_4(x_1, x_2, x_3, x_4) = x_1 \times 1 + x_2 \times 5 + x_3 \times 2 + x_4 \times 4$,

   (e) $y = f_5(x_1, x_2, x_3, x_4) = x_1 \times 0 + x_2 \times 3 + x_3 \times 6 + x_4 \times 4$.

3.3.2.2 Sharing Phase

The unit-price $d_{jkl} = 135$ of record #124 is encrypted as follows.

1. Three base-p digits $\{d\_s_{jklh}\}_{h=1\cdots3}$ are computed as follows:

   (a) $d\_s_{jkl1} = \lfloor 135/7^{(1-1)} \rfloor \bmod 7 = 2$,

   (b) $d\_s_{jkl2} = \lfloor 135/7^{(2-1)} \rfloor \bmod 7 = 5$,

   (c) $d\_s_{jkl3} = \lfloor 135/7^{(3-1)} \rfloor \bmod 7 = 2$.

2. The inner signature is $s\_in_{jkl} = HF(2, 5, 2) = \{2 + 5 + 2\} \bmod 7 = 2$.

3. Five shares $\{e_{ijkl}\}_{i=1\cdots5}$ are computed as follows:

   (a) $e_{1jkl} = f_1(2, 5, 2, 2) = 2 \times 0 + 5 \times 3 + 2 \times 3 + 2 \times 4 = 29$,

   (b) $e_{2jkl} = f_2(2, 5, 2, 2) = 2 \times 4 + 5 \times 1 + 2 \times 0 + 2 \times 3 = 19$,

   (c) $e_{3jkl} = f_3(2, 5, 2, 2) = 2 \times 0 + 5 \times 5 + 2 \times 6 + 2 \times 1 = 39$,

   (d) $e_{4jkl} = f_4(2, 5, 2, 2) = 2 \times 1 + 5 \times 5 + 2 \times 2 + 2 \times 4 = 39$,

   (e) $e_{5jkl} = f_5(2, 5, 2, 2) = 2 \times 0 + 5 \times 3 + 2 \times 6 + 2 \times 4 = 35$.

4. Five outer signatures $\{s\_out_{ijkl}\}_{i=1\cdots5}$ are computed as follows:

   (a) $s\_out_{1jkl} = OF(29) = 29 \bmod 3 = 2$,

   (b) $s\_out_{2jkl} = OF(19) = 19 \bmod 3 = 1$,

   (c) $s\_out_{3jkl} = OF(39) = 39 \bmod 3 = 0$,

   (d) $s\_out_{4jkl} = OF(39) = 39 \bmod 3 = 0$,

   (e) $s\_out_{5jkl} = OF(35) = 35 \bmod 3 = 2$.

5. Each share e_{ijkl} and its outer signature s_out_{ijkl} are distributed to attributes 'Unit-Price' and 'Sig-UnitPrice', respectively, in record #124 of table PRODUCT at CSP_i.

3.3.2.3 Reconstructing Phase

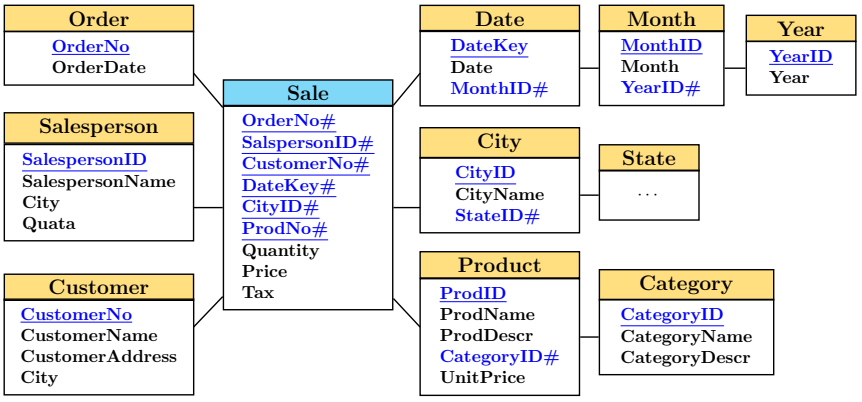
The unit-price $d_{jkl} = 135$ of record #124 is reconstructed as follows.

1. Let us select $RG = \{CSP_i\}_{i\in\{1,2,3,4\}}$.
2. At each $CSP_i \in RG$, share e_{ijkl} is verified as follows.
   (a) Share e_{1jkl} is correct because $s\_out_{1jkl} = OF(29) = 29 \bmod 3 = 2$.
   (b) Share e_{2jkl} is correct because $s\_out_{2jkl} = OF(19) = 19 \bmod 3 = 1$.
   (c) Share e_{3jkl} is correct because $s\_out_{3jkl} = OF(39) = 39 \bmod 3 = 0$.
   (d) Share e_{4jkl} is correct because $s\_out_{4jkl} = OF(39) = 39 \bmod 3 = 0$.
3. Matrix B is created from pseudorandom coefficients of four $f_{i=1\ldots4}$ where the coefficients are seeded at CSP identifiers $\{ID_i\}_{i\in RG}$:
   $B = \begin{bmatrix} 0 & 3 & 3 & 4 \\ 4 & 1 & 0 & 3 \\ 0 & 5 & 6 & 1 \\ 1 & 5 & 2 & 4 \end{bmatrix}$.
4. Matrix C is computed: $C = B^{-1} = \begin{bmatrix} -38 & 59 & 23 & -12 \\ -70 & -21 & 7 & 84 \\ 46 & 17 & 37 & -68 \\ 74 & 3 & -33 & -12 \end{bmatrix} /224$.
5. Three base-p digits $\{d\_s_{jklh}\}_{h\in\{1,2,3\}}$ are computed as follows:
   (a) $d\_s_{jkl1} = (-38 \times 29 + 59 \times 19 + 23 \times 39 - 12 \times 39)\,/224 = 2$,
   (b) $d\_s_{jkl2} = (-70 \times 29 - 21 \times 19 + 7 \times 39 + 84 \times 39)\,/224 = 5$,
   (c) $d\_s_{jkl3} = (46 \times 29 + 17 \times 19 + 37 \times 39 - 68 \times 39)\,/224 = 2$.
6. The inner signature is $s\_in_{jkl} = (74 \times 29 + 3 \times 19 - 33 \times 39 - 12 \times 39)\,/224 = 2$.
7. We verify the original all digits $\{d\_s_{jklh}\}_{h=1\ldots3}$. The result is correct because $s\_in_{jkl} = HF(2, 5, 2) = (2 + 5 + 2) \bmod 7 = 2$.

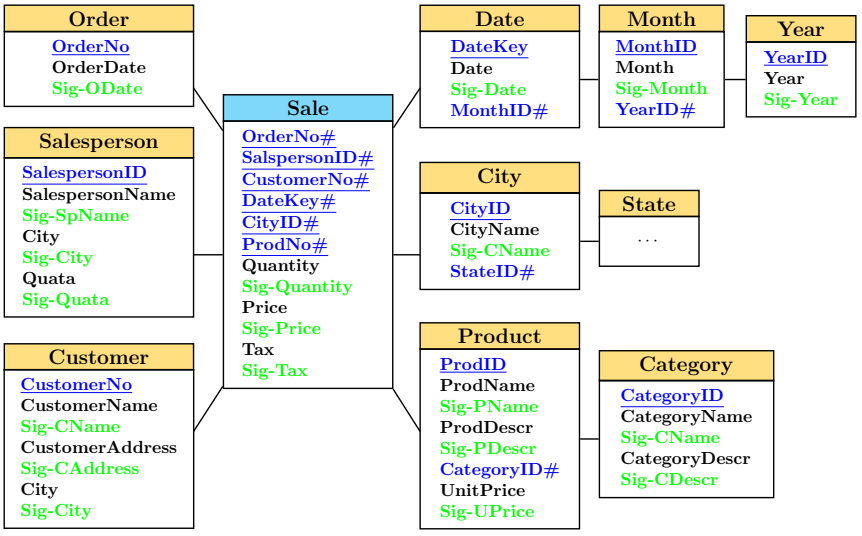
3.4 Sharing Data Warehouses

Since each table of a shared DW is stored in a relational database at a given CSP's and each attribute value is encrypted independently, bpVSS straightforwardly helps implement any DW logical model, i.e., star, snowflake or constellation schema. Figures

3.6(a) and 3.6(b) show an example of snowflake-modeled DW that is shared among three CSPs. Each shared DW bears the same schema as the original DW's, but type and size of each attribute in shared tables differ from the original tables. All attribute types, except Booleans that are not encrypted to save computation and data storage costs, are indeed transformed into integers. Moreover, shared tables have a greater number of attributes than original tables because outer signature attributes (featured in green) are also stored in shared tables. Keys (featured in blue) are not encrypted. Recally, they help match records in different shared tables at each CSP. Moreover, they also help match shared records and query results in the data reconstruction process.



(a) Original relational schema



(b) Transformed relational schema at CSP's

FIGURE 3.6: Example of shared data warehouse

3.4.1 Cloud Cubes

bpVSS supports the storage of data cubes that optimize response time and bandwidth when performing Relational OLAP (ROLAP) operations. Cubes are physically

stored into relational tables that are shared among CSPs, retaining the same structure. For example, Figure 3.7 features a shared cube named Cube-I that totalizes total prices and numbers of sales by time period and by product. The hierarchical aggregation path of time is "Year - Month - Date" whereas that of product is "Category - Product". Shared cubes include outer signatures for shared aggregate measures and customarily use NULL values to encode superaggregates. For instances:

- record #1 stores aggregates for all products for all dates,
- record #2 stores aggregates for all product in category #1 for all dates,
- record #3 stores aggregates for all products in year #1,
- record #4 stores aggregates for all product in category #1 in year #1.

	Foreign Keys					**Shares and Signatures**			
	Time Keys			**Product Keys**		**Aggregate Shares**		**Signatures**	
	YearID	**MonthID**	**DateID**	**CategoryID**	**ProdNo**	**totalPrice**	**number**	**Sig-SPrice**	**Sig-COUNT**
#1	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	83231	58244	864	138
#2	*NULL*	*NULL*	*NULL*	1	*NULL*	26701	18254	839	337
	NULL	*NULL*	*NULL*	1	1	8958	7113	347	483
	NULL	*NULL*	*NULL*	1	⋮	⋮	⋮	⋮	⋮
	NULL	*NULL*	*NULL*	1	2	4348	1844	322	332
	NULL	*NULL*	*NULL*	⋮	⋮	⋮	⋮	⋮	⋮
#3	1	*NULL*	*NULL*	*NULL*	*NULL*	44574	54542	934	344
#4	1	*NULL*	*NULL*	1	*NULL*	21158	8954	342	745
	1	*NULL*	*NULL*	1	1	9754	4544	485	434
	1	*NULL*	*NULL*	1	⋮	⋮	⋮	⋮	⋮
	1	*NULL*	*NULL*	2	1	18444	5747	243	534
	1	*NULL*	*NULL*	⋮	⋮	⋮	⋮	⋮	⋮
	1	1	*NULL*	*NULL*	*NULL*	8312	5812	324	324
	1	1	*NULL*	1	*NULL*	2312	1822	423	343
	1	1	*NULL*	1	1	988	586	434	433
	1	1	*NULL*	1	⋮	⋮	⋮	⋮	⋮
	1	1	*NULL*	2	1	756	458	439	332
	1	1	*NULL*	⋮	⋮	⋮	⋮	⋮	⋮
	1	2	*NULL*	*NULL*	*NULL*	9758	6254	344	232
	1	⋮	*NULL*	⋮	⋮	⋮	⋮	⋮	⋮
	1	1	1	*NULL*	*NULL*	2578	1587	439	320
	1	1	1	1	*NULL*	548	425	978	349
	1	1	1	1	1	56	24	43	32
	1	1	1	1	⋮	⋮	⋮	⋮	⋮
	1	1	1	1	2	95	67	32	23
	1	1	1	1	⋮	⋮	⋮	⋮	⋮
	1	1	1	2	*NULL*	689	357	232	323
	1	1	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

FIGURE 3.7: Sample cloud cube or Cube-I

3.4.2 Indices

Although bpVSS can perform exact match and aggregate queries on shares without the help of indices, indices can be shared to improve query performance. However,

they must be created from the original data before the sharing process takes place. Creating indices is not only a tradeoff between query performance improvement and extra storage cost, but it also incurs higher risks. Indeed, original data may be extracted by unauthorized persons through related indices. For example, data patterns can be learned from the data ordering stored in indices, thus helping breaking encryption.

3.5 Loading, Backup and Recovery Processes

3.5.1 Loading Data

For loading data into a shared DW, each data piece is encrypted and loaded independently. New data can be loaded without decrypting previous data first, because each attribute value in each record is encrypted independently. For instance, in Figure 3.8, data from Figure 3.2 are already shared and the last record (#127) is new. Each share is created as in Section 3.2.2.

3.5.2 Refreshing a Cloud Cube

When updating cubes, some shared aggregates may have to be recomputed. Within bpVSS, we currently cannot apply all aggregation operations on shares (Section 3.6). Thus, such aggregations still require to be computed on the original data. For example, maximum and minimum cannot be computed on shares because original data order is lost in the sharing process. Averaging data must be performed by summing and counting. Hence, to optimize costs, aggregates are first computed on new data, and then aggregated to relevant, existing shares, which are decrypted on-the-fly.

3.5.3 Backup and Recovery

In bpVSS, as in all similar approaches, backups are unnecessary if $n > t$ because each shared table ET_{ij} is actually a backup share of all other shared tables ET_{ia}, where $a \in [1, n], a \neq j$. Moreover, unlike all related approaches, bpVSS' outer signatures help detect incorrect attribute values in any shared table. Erroneous shares can be recovered from t other shared tables.

For example, in Figure 3.8, suppose that the share of unit-price in record #124 of PRODUCT at CSP_5 is modified to 39. The modified value does not match with its outer signature of $2 \neq (39 \bmod 3 = 0)$. The original value can be recovered from other shares of

ProNo	ProName	ProDescr	CategoryID	UnitPrice
124	Shirt	Red	1	135
125	Shoe	*NULL*	2	142
126	Ring	*NULL*	1	142
127	Hat	*NULL*	3	25

(a) Original data

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{31,10,21,36,16}	{1,0,1,1,1}	{27,26,22}	{2,1,2}	1	29	2
125	{31,10,17,26}	{1,0,2,1}	*NULL*	*NULL*	2	36	0
126	{27,21,13,6}	{2,1,3,1}	*NULL*	*NULL*	1	36	0
127	{36,45,16}	{1,0,1}	*NULL*	*NULL*	3	9	0

(b) Shares at CSP_1

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{40,27,10,28,21}	{0,2,0,3,1}	{33,27,20}	{3,2,0}	1	19	1
125	{40,27,31,27}	{0,2,1,2}	*NULL*	*NULL*	2	23	2
126	{33,10,24,20}	{3,0,4,0}	*NULL*	*NULL*	1	23	2
127	{29,48,21}	{4,3,1}	*NULL*	*NULL*	3	19	1

(c) Shares at CSP_2

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{30,13,20,28,23}	{0,3,0,3,3}	{29,17,16}	{4,2,1}	1	39	0
125	{30,13,19,17}	{0,3,4,2}	*NULL*	*NULL*	2	45	0
126	{29,20,18,12}	{4,0,3,2}	*NULL*	*NULL*	1	45	0
127	{27,42,23}	{2,2,3}	*NULL*	*NULL*	3	15	0

(d) Shares at CSP_3

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{44,14,21,40,22}	{4,4,1,0,2}	{39,27,22}	{4,2,2}	1	39	0
125	{44,14,23,27}	{4,4,3,2}	*NULL*	*NULL*	2	48	0
126	{39,21,18,9}	{4,1,3,4}	*NULL*	*NULL*	1	48	0
127	{43,62,22}	{3,2,2}	*NULL*	*NULL*	3	19	1

(e) Shares at CSP_4

ProNo	ProName	Sig-PName	ProDescr	Sig-PDescr	CategoryID	UnitPrice	Sig-UPrice
124	{34,16,27,42,22}	{4,4,2,2,2}	{30,32,28}	{0,2,3}	1	35	2
125	{34,16,23,32}	{3,1,3,2}	*NULL*	*NULL*	2	42	0
126	{30,27,19,12}	{0,2,4,2}	*NULL*	*NULL*	1	42	0
127	{39,48,22,}	{4,3,2}	*NULL*	*NULL*	3	9	0

(f) Shares at CSP_5

FIGURE 3.8: Example of sharing new data

unit-price in record #124 of PRODUCT at other four CSPs $\{CSP_1, CSP_2, CSP_3, CSP_4\}$ as follows:

1. Reconstruct data as in Section 3.3.2.3.
2. Share the correct share at CSP_5 from reconstructed data as in Section 3.3.2.2.

3.6 Data Analysis over Shares

3.6.1 Querying a Shared Data Warehouse

Although some queries apply directly onto shares, others require some or all data to be decrypted. Simple SELECT/FROM queries directly apply onto shares. However, the primary key is required to match resulting records from t CSPs. For example, in

Table 3.1, ProNo is inserted into Query #1 to match the shares of each ProdName from t CSPs. Then, transformed Query #1 is run at each CSP's. All join operators, when operating on unencrypted keys, also apply directly, e.g., Query #2.

TABLE 3.1: Query examples w.r.t. to Figure 3.6's DW and Figure 3.8's data and shares

No	Type	Original query	Transformed query
1	Simple SELECT query	SELECT ProdName FROM Product	SELECT ProdName, ProNo FROM Product
2	JOIN query	SELECT P.ProdName, C.CategoryName FROM Product AS P JOIN Category AS C ON P.CategoryID=C.CategoryID	SELECT P.ProdName, P.ProNo, C.CategoryName, C.CategoryID FROM Product AS P JOIN Category AS C ON P.CategoryID=C.CategoryID
3	Exact match query	SELECT ProdName FROM Product WHERE UnitPrice = 142	SELECT ProdName, ProNo FROM Product WHERE UnitPrice = 36
4	Rank query	SELECT ProdName FROM Product WHERE UnitPrice BETWEEN 133 AND 137	SELECT ProdName, ProNo FROM Product WHERE UnitPrice IN (21,25,29,33,37)
5	GROUP BY query	SELECT ProdName, SUM(UnitPrice) FROM Product GROUP BY ProNo	SELECT ProdName, ProNo, SUM(UnitPrice) FROM Product GROUP BY ProNo

However, when expressing conditions in a WHERE or HAVING clause, the following routine must be followed:

1. encrypt compared values,
2. substitute these shares to compared values in the query,
3. launch the query on t shares,
4. intersect the records of query results from t CSPs,
5. reconstruct the global result.

For example, to run Query #3 at CSP_1, the selected unit price (142) is encrypted into 36 in Step #1. Then, Query #3's comparison clause is transformed in Step #2. Transformed Query #3 is executed at CSP_1. However, at CSP_1, 36 may be the share of other values but 142 (Section 5.2.2.3.2), thus the intersection in Step #4 helps filter shares of other values (false positives) out. Only products with prices equal to 142 are reconstructed in Step #5.

This routine works for most comparison operators (=, ≠, EXISTS, IN, LIKE...) and their conjunction, but when ordering is necessary, as in ORDER BY clauses and comparison operators (>, <, ≥, ≤, BETWEEN...), it cannot directly apply since the original order is broken when sharing data. Thus, all fetched data must be reconstructed at the client's before the result can be computed by an external program. However, some rank queries can be transformed and performed on shares if comparison range is known and the comparison attribute's type is integer, char or string. For example, comparison clause of Query #4 is transformed to run at CSP_1, where 21, 25, 29, 33 and 37 are the shares of 133, 134, 135, 136 and 137, respectively.

Similarly, aggregation functions AVG, COUNT, and SUM can apply on shares, whereas other aggregation functions, such as MAX and MIN, require all original data to be reconstructed prior to computation. Finally, grouping queries can directly apply if and only if they target unencrypted key attributes. Again, grouping by other attribute(s) requires all data to be reconstructed at the user's before aggregation by an external program. For example, Query #5 applies directly at all CSPs.

Consequently, executing a query may require either transforming or splitting it, depending on its clauses and operators, following the above guidelines. Figure 3.10 shows the way a sample query runs at the user's and at four CSPs (CSP_1, CSP_2, CSP_3 and CSP_4). Query #6 (Figure 3.9) inputs data from Figure 3.8:

```
SELECT P.ProdName, AVG(S.Price)
FROM Sale AS S RIGHT JOIN Product AS P
ON S.ProdNo = P.ProdNo
GROUP BY S.ProdNo
WHERE P.UnitPrice = 135
```

FIGURE 3.9: Query 6

Query #6 is rewritten into two queries Query #6.1 and Query #6.2. The comparison clause (featured in red) of Query #6.1 is transformed to match with shares of 135 at each CSP before the query is executed. Unlike Query #6.1, Query #6.2 can directly run on shares at all CSPs because the compared value (ProNo) is an unencrypted key. Note that although some tasks are executed at the user's side, total running time is much faster than that of tasks executed at CSPs', because these tasks must traverse whole shared tables, while at the user's, only query results are processed.

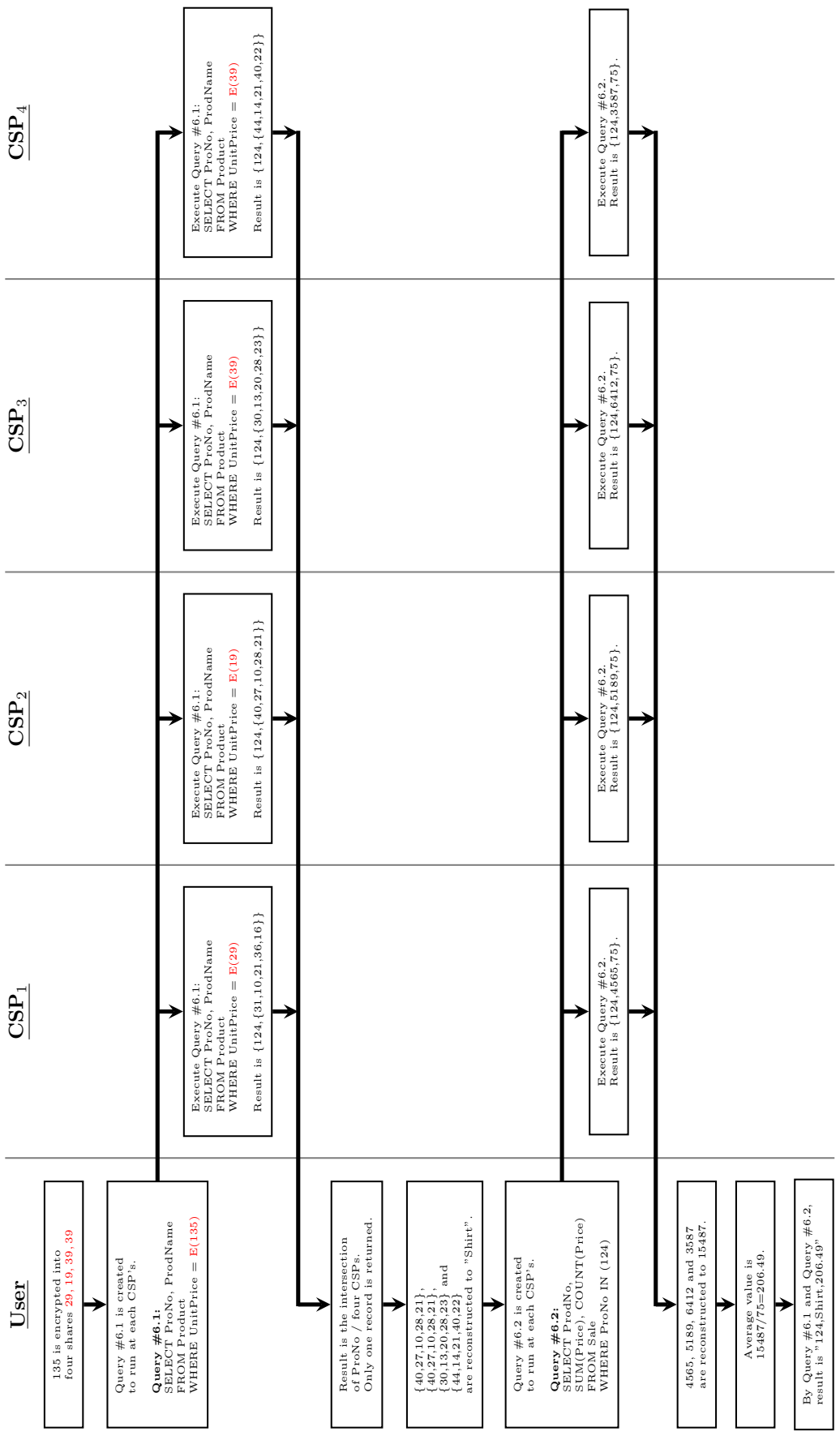


FIGURE 3.10: Example of a query execution over shares

3.6.2 Query a Cloud Cube

bpVSS directly supports all basic OLAP operations by directly querying cloud cubes and reconstructing the global result. Table 3.2 shows example queries that retrieve the total price and the number of products at four levels of the time hierarchy (all - years - months - dates) from Cube-I, drilling down from the first line/query of Table 3.2 to the last.

TABLE 3.2: Example of queries from Cube-I

Time hierarchical level	**Query**
All dates	SELECT SUM(totalPrice), SUM(number) FROM Cube-I WHERE CategoryID IS NULL and ProdNo IS NULL and YearID IS NULL and MonthID IS NULL and DateID IS NULL
All years	SELECT YearID, SUM(totalPrice), SUM(number) FROM Cube-I WHERE CategoryID IS NULL and ProdNo IS NULL and YearID IS NOT NULL and MonthID IS NULL and DateID IS NULL GROUP BY YearID
All months in year #1	SELECT YearID, MonthID, SUM(totalPrice), SUM(number) FROM Cube-I WHERE CategoryID IS NULL and ProdNo IS NULL and YearID=1 and MonthID IS NOT NULL and DateID IS NULL GROUP BY YearID, MonthID
All dates in month #1 and year #1	SELECT YearID, MonthID, DateID, SUM(totalPrice), SUM(number) FROM Cube-I WHERE CategoryID IS NULL and ProdNo IS NULL and YearID=1 and MonthID=1 and DateID IS NOT NULL GROUP BY YearID, MonthID, DateID

3.7 Summary

This chapter describes an original approach named bpVSS to share a DW in the cloud. bpVSS simultaneously supports data privacy, availability, integrity and OLAP querying. bpVSS exploits block cryptography and secret sharing to guarantee data privacy and availability. Moreover, it ensures data correctness by utilizing homomorphic and one-way functions as signatures. This also helps prevent CSP cheating. In addition, bpVSS distributes shares to several CSPs to handle data availability and recover data (in case data error) from other CSPs without requiring any backup.

Since, bpVSS belongs to the first family of secure distributed database approaches (Section 2.2), data analysis are allowed on shares. To analyze data, some traditional SQL queries and OLAP queries can be performed on shares at CSPs' without decryption, and

then only target results are reconstructed at the user's. This helps reduce communication and computation costs at the user's. Moreover, this also helps protect from curious CSPs because data is meaningless at CSPs.

Although secret sharing is very costly in terms of volume, bpVSS controls share volume by computing shares from base-p integers. This helps guaranteeing that share volume is lower than n times that of original data.

‘

Chapter 4

fVSS: Flexible Verifiable Secret Sharing

4.1 fVSS Principle

fVSS is a (t, n) flexible verifiable secret sharing scheme belonging to the second family of approaches identified in Section 2.2. As all similar approaches, fVSS shares data over n CSPs (Figure 4.1), t of which are necessary to reconstruct original data. Hence, both data privacy and data availability are enforced. Moreover, as bpVSS, data integrity is verified with inner and outer signatures. Inner signatures help verify data correctness in case some CSPs are not honest. They are created from an homomorphic function and hidden in shares. Outer signatures help verify share correctness at CSPs', and thus no erroneous shares are transferred back to the user. Outer signatures are stored in a tree data structure to support on-demand verification. Finally, three types of indices are proposed. Type I indices are bitmaps stored in the index server. They store locations of shares and are used in SUM and AVG queries. Type II indices are B+ trees stored at the index server's. They are used in exact match, range and MAX, MIN, MEDIAN, MODE and COUNT queries. Type III indices are encrypted and stored at CSPs'. They are exploited for instance STDDEV and VARIANCE queries.

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 4.2, 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. Moreover, data can be shared even though some CSPs fail, which is impossible in previous SSSs. For example, record #124 of table PRODUCT can be shared although CSP_2 and CSP_4 fail since this record is shared at only CSP_1, CSP_3 and CSP_5. Finally, fVSS can protect data even if all n CSPs are malicious,

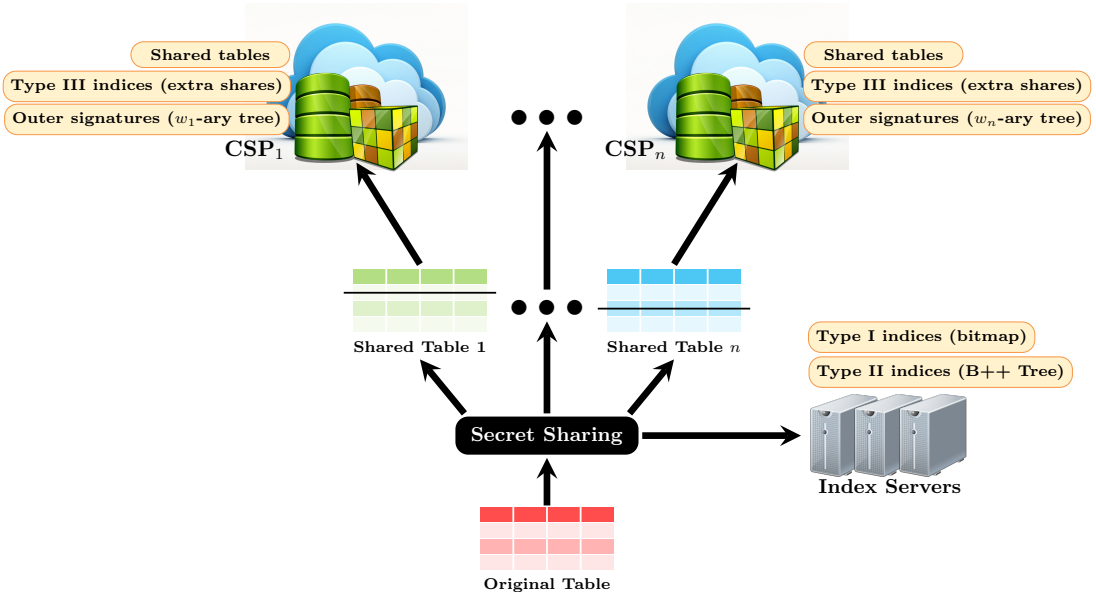


FIGURE 4.1: fVSS framework

whereas classic secret sharing cannot. For example, in table PRODUCT, each record is shared into only three records at three out of five ($n = 5$) CSPs, but four ($t = 4$) shares are required to reconstruct it. Hence, no record can be reconstructed by unauthorized persons. However, the user can reconstruct data with the help of pseudo shares that replace the fourth share in our example (Section 4.2).

To achieve all these new benefits, we proceed as follows. Suppose we want to share piece of data d_{jkl}, e.g., the category ID of product #124 in Figure 4.2(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 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 4.3 plots an example where $t = 4$ and $n = 5$. 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 (Type I indices) coding where shares are stored are maintained in the index server. For example, the bitmap corresponding to product #124 in Figure 4.2 is 10101, with a 1 value at position $\#i$ represents share storage at CSP_i.

The remainder of this chapter is organized as follows. Section 4.2 details the sharing and reconstruction processes. Section 4.3 details our novel outer signatures. Section 4.4 describes the way we share data warehouses. Section 4.5 presents the loading,

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
124	Shirt	Red	1	135
125	Shoe	*NULL*	2	142
126	Ring	*NULL*	1	142

(a) Original data

ProdNo	Share location
124	10101
125	01110
126	11010

(b) Type I indices

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
124	{-46.1288 ; -72.0606 ; -73.2955 ; -84.4091 ; -86.8788}	{-44.8939 ; -68.3561 ; -67.1212}	1	-110.3409
126	{-106.1818 ; -224.3182 ; -250.0000 ; -214.0455}	*NULL*	1	-414.3636

(c) Shares at CSP_1

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
125	{81.4636 ; 91.8682 ; 95.3366 ; 90.3818}	*NULL*	2	110.6955
126	{104.8182 ; 139.8409 ; 147.4545 ; 136.7955}	*NULL*	1	196.1818

(d) Shares at CSP_2

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
124	{94.5732 ; 121.3535 ; 122.6288 ; 134.1061 ; 136.6566}	{93.2980 ; 117.5278 ; 116.2525}	1	160.8864
125	{59.1273 ; 80.7000 ; 87.8909 ; 77.6182}	*NULL*	2	119.7364

(e) Shares at CSP_3

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
125	{-52.6000 ; -31.9818 ; -25.1091 ; -34.9273}	*NULL*	2	5.3273
126	{-431.0909 ; -786.5455 ; -863.8182 ; -755.6364}	*NULL*	1	-1358.3636

(f) Shares at CSP_4

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
124	{-37.0707 ; -57.8586; -58.8485; -67.7576; -69.7374}	{-36.0808; -54.8889 ; -53.8990}	1	-88.5455

(g) Shares at CSP_5

FIGURE 4.2: Sample original and shared data

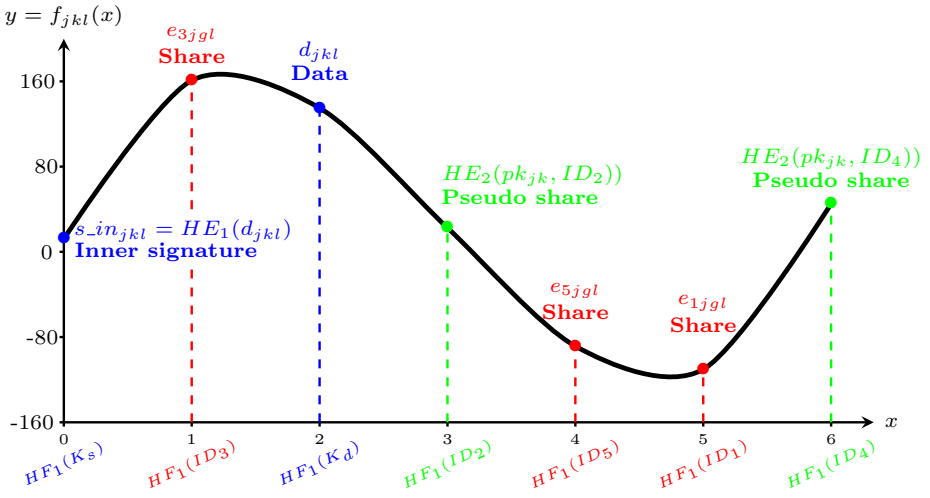


FIGURE 4.3: Sample sharing process

backup and recovery processes. Section 4.6 describes how we perform queries and OLAP operations on shared data warehouses. We finally close this chapter by a summary of highlights in Section 4.7.

4.2 Data Sharing and Reconstruction

As in bpVSS, in fVSS, DB attribute values, except NULL values and primary or foreign keys, are 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.

Although data in a table is shared in n tables at n CSPs', each record R_{jk} is shared in only $n-t+2$ records. Hence, only $n-t+2$ shared tables store R_{jk}'s shares whereas R_{jk}'s shares are absent from $t-2$ shared tables. For example, in Figure 4.2, the PRODUCT table is shared in five ($n=5$) shared tables. However, each record is shared in only three ($n-t+2=5-4+2=3$) shared tables. Type I indices in the index servers help represent which CSPs store shared records.

Each attribute value in each record is encrypted independently. The value d_{jkl} of attribute A_{jl} in record R_{jk} is encrypted into $n-t+2$ shares generated from a polynomial $f_{jkl}(x)$ of degree $t-1$ created by Lagrange interpolation from data d_{jkl}, 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.

fVSS' inner signature is very similar to that of bpVSS. It is created from data d_{jkl} by homomorphic encryption [6]. Inner signature s_in_{jkl} matches with data d_{jkl} in the reconstruction process only if CSPs in RG return correct shares. fVSS' new outer signatures are constructed and verified at CSPs. Details about outer signature are provided in Section 4.3.

The remainder of this section is organized as follows. Section 4.2.1 specifies parameter settings and functions related to the data sharing and reconstruction processes. Sections 4.2.2 and 4.2.3 describe each step of the data sharing and reconstruction processes, respectively. Finally, Section 4.2.4 exemplifies fVSS.

4.2.1 Initialization Phase

1. Set values of n, t and p (a prime number) such that $p > n \geq t$.
2. Define one-variable hash function $HF_1(a)$, where $a \in]0, p[$ and $HF(a) \in]0, p[$. HF's parameter is a CSP identifier, data key or signature key. HF returns an integer value that is a polynomial's parameter in both the sharing and reconstructing processes.

3. Set values of CSP identifiers $ID_{i=1..n}$, data key K_d and signature key K_s such that their integer values range in $]0, p[$. All $HF_1(ID_i)$ must be unique and different from $HF_1(K_d)$ and $HF_1(K_s)$ ($HF_1(ID_{i1}) \neq HF_1(ID_{i2}) \neq HF_1(K_d) \neq HF_1(K_s)$ if $i1 \neq i2$).

4. To create an inner signature, define one-variable one-way homomorphic function $HE_1(d)$, where $d \in \{\mathbb{N}, \mathbb{R}\}$, $HE_1(d) \in \{\mathbb{R}\}$ and $HE_1(d_1) \pm HE_1(d_2) = HE_1(d_1 \pm d_2)$. HE_1 inputs a data piece and outputs an inner signature.

5. To create a pseudo share, define two-variables one-way homomorphic function $HE_2(a, b)$, where $a \in \{\mathbb{N}^+\}$, $b \in \{\mathbb{N}^+\}$, $HE_2(a, b) \in \{\mathbb{R}\}$ and $HE_2(a_1, b) + HE_2(a_2, b) = HE_2(a_1 + a_2, b)$. HE_2's parameters a and b are a primary key and a CSP identifier, respectively. HE_2 returns a pseudo share.

Note that all parameters and functions must be secret and highly protected by the user.

4.2.2 Data Sharing Process

Any record R_{jk} in table T_j 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 4.1):

   $$f_{jkl}(x) = \sum_{\alpha=1}^{t} \prod_{1 \leq \beta \leq t, \alpha \neq \beta} \frac{x - x_\beta}{x_\alpha - x_\beta} \times y_\alpha \tag{4.1}$$

   where $\{(x_1, y_1), \ldots, (x_t, y_t)\} = \{(HF_1(K_d), d_{jkl}), (HF_1(K_s), s\_in_{jkl})\}$ $\cup \{(HF_1(ID_i), HE_2(pk_{jk}, ID_i))_{CSP_i \in UG_{jk}}\}$. $HE_2(pk_{jk}, ID_i)_{CSP_i \in UG_{jk}}$ 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 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, timestamps and reals can be directly shared by fVSS. Data of other types (i.e., characters, strings and binary strings) are first transformed into integers, and then shared as in bpVSS (Section 3.3). Figure 4.4 summarizes the transformation process of data of various types into reals. fVSS indeed outputs reals because integers would not provide enough precision when computing sums on large values. Note that, to minimize share volume at CSPs', shares of string attributes can be compressed with any compression algorithm (i.e., Huffman code) after encryption, since they are not used when performing exact match, range and aggregation queries. Exact match and range queries indeed run directly on Type II indices (B+ trees).

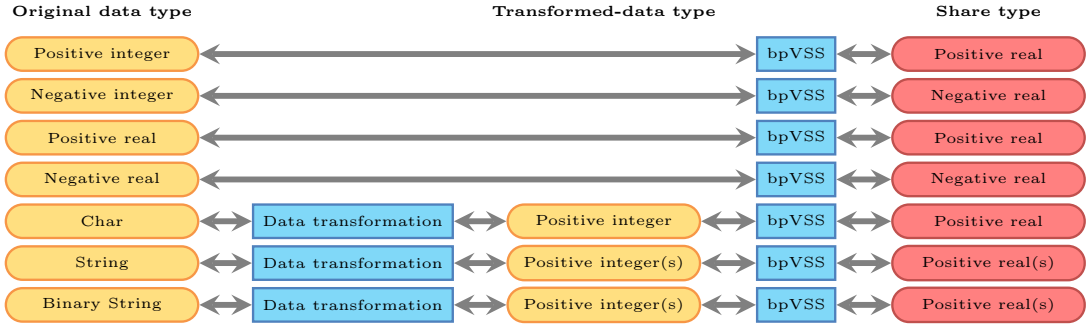


FIGURE 4.4: Data type transformation process

4.2.3 Data Reconstruction Process

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

1. Select t out of n CSPs. This set of CSPs is called the reconstruction group RG.
2. For each $CSP_i \in SG_{jk} \cap RG$, if share at CSP_i is erroneous (cf. verification process from in Section 4.3), 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 such that $pk_{jk} = pk_{ijg}$.
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 4.1) with $y_i = f_{jkl}(x_i)$ and $x_i = HF_1(ID_i)$.
6. Compute data piece $d_{jkl} = f_{jkl}(HF_1(K_d))$.

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.

4.2.4 Recapitulative Example

Let us refer back to Figure 4.2. The sharing and reconstruction step for Record #124 follow.

4.2.4.1 Initialization Phase

1. Let parameters be assigned as follows: $n = 5$, $t = 4$ and $p = 11$.

2. Let us select the following one-variable hash function: $HF_1(a) = a \times 5 \bmod 7$.

3. Let CSP identifiers, data key and signature key be assigned as follows: $ID_1 = 1$, $ID_2 = 2$, $ID_3 = 3$, $ID_4 = 4$, $ID_5 = 5$, $K_d = 6$ and $K_s = 7$. They must meet the condition: all $HF_1(ID_i)$ must be unique and different from $HF_1(K_d)$ and $HF_1(K_s)$.

   - $HF_1(ID_1) = HF_1(1) = 1 \times 5 \bmod 7 = 5$
   - $HF_1(ID_2) = HF_1(2) = 2 \times 5 \bmod 7 = 3$
   - $HF_1(ID_3) = HF_1(3) = 3 \times 5 \bmod 7 = 1$
   - $HF_1(ID_4) = HF_1(4) = 4 \times 5 \bmod 7 = 6$
   - $HF_1(ID_5) = HF_1(5) = 5 \times 5 \bmod 7 = 4$
   - $HF_1(K_d) = HF_1(6) = 6 \times 5 \bmod 7 = 2$
   - $HF_1(K_s) = HF_1(7) = 7 \times 5 \bmod 7 = 0$

4. To create the inner signature, let us select the following one-variable one-way homomorphic function: $HE_1(h) = h/p$.

5. To create pseudo shares, let us select the following two-variables one-way homomorphic function: $HE_2(a, b) = (a \times b)/p$.

4.2.4.2 Sharing Phase

Record #124 (R_{jk}) is encrypted independently as follows.

1. With $n = 5$ and $t = 4$, the group SG_{jk} of CSPs that store shared record #124 is made of $n-t+2 = 5-4+2 = 3$ CSPs. Let us select $SG_{jk} = \{CSP_1, CSP_3, CSP_5\}$. Then $UG_{jk} = \{CSP_2, CSP_4\}$.

2. Two pseudo shares for CSPs in UG_{jk} are computed as follows.

   (a) CSP_2's pseudo share is $HE_2(pk_{jk}, ID_2) = HE_2(124, 2) = 124 \times 2/11 = 22.5454$.

   (b) CSP_4's pseudo share is $HE_2(pk_{jk}, ID_4) = HE_2(124, 4) = 124 \times 4/11 = 45.0909$.

3. We encrypt each attribute value in record #124. For example, unit-price $d_{jkl} = 135$ is shared as follows.

   (a) The inner signature is $s\_in_{jkl} = HE_1(d_{jkl}) = HE_1(135) = 135/11 = 12.2727$.

   (b) By Lagrange interpolation, polynomial $f_{jkl}(x)$ of degree 3 is computed from:

      i. data key and value to share $(HF_1(K_d), d_{jkl}) = (2, 135)$,

      ii. signature key and inner signature $(HF_1(K_s), s\_in_{jkl}) = (0, 12.2727)$,

      iii. CSP_2's identifier and pseudo share $(HF_1(ID_2), HE_2(pk_{jk}, ID_2)) = (3, 22.5454)$,

      iv. CSP_4's identifier and pseudo share $(HF_1(ID_4), HE_2(pk_{jk}, ID_4)) = (6, 45.0909)$.

   The function from Figure 4.3 is $f_{jkl}(x) = \frac{(x-2)(x-3)(x-6)\times 12.2727}{(0-2)(0-3)(0-6)}$
   $+\frac{(x-0)(x-3)(x-6)\times 135}{(2-0)(2-3)(2-6)} + \frac{(x-0)(x-2)(x-6)\times 22.5454}{(3-0)(3-2)(3-6)} + \frac{(x-0)(x-2)(x-3)\times 45.0909}{(6-0)(6-2)(6-3)}$
   $= 14.6553 \times x^3 - 131.2159 \times x^2 + 265.1742 \times x + 12.2727$.

   (c) The three shares of CSPs in SG_{jk} are computed as follows:

      i. $e_{1jkl} = f_{jkl}(HF_1(ID_1)) = f_{jkl}(5)$
      $= 14.6553 \times 5^3 - 131.2159 \times 5^2 + 265.1742 \times 5 + 12.2727 = -110.3409$,

      ii. $e_{3jkl} = f_{jkl}(HF_1(ID_3)) = f_{jkl}(1)$
      $= 14.6553 \times 1^3 - 131.2159 \times 1^2 + 265.1742 \times 1 + 12.2727 = 160.8864$,

      iii. $e_{5jkl} = f_{jkl}(HF_1(ID_5)) = f_{jkl}(4)$
      $= 14.6553 \times 4^3 - 131.2159 \times 4^2 + 265.1742 \times 4 + 12.2727 = -88.5455$.

4. Record R_{jk} is shared into three records ER_{1jk}, ER_{3jk} and ER_{5jk} at CSP_1, CSP_3 and CSP_5, respectively. The bitmap 10101 corresponding to R_{jk} is stored in the index server. All shared records and the bitmap are linked through primary key $pk_{jk} = 124$.

4.2.4.3 Reconstructing Phase

Record #124 (R_{jk}) is reconstructed as follows.

1. Let us select $RG = \{CSP_i\}_{i=\{1,2,3,4\}}$.
2. Let us suppose all shares at CSP_1 and CSP_3 are correct (cf. verification process from Section 4.3).
3. ER_{1jg} and ER_{3jg}, which are shares of Record #124 at CSP_1 and CSP_3, respectively, are transferred to the user.
4. Two pseudo shares are computed as follows:
   (a) CSP_2's pseudo share is $HE_2(pk_{jk}, ID_2) = HE_2(124, 2) = 124 \times 2/11 = 22.5454$;
   (b) CSP_4's pseudo share is $HE_2(pk_{jk}, ID_4) = HE_2(124, 4) = 124 \times 4/11 = 45.0909$.
5. We reconstruct each attribute value in record #124 independently. For example, unit-price $d_{jkl} = 135$ is reconstructed as follows.
   (a) By Lagrange interpolation, polynomial $f_{jkl}(x)$ of degree 3 is computed from:
      i. CSP_1's identifier and share $(HF_1(ID_1), -110.3409) = (5, -110.3409)$,
      ii. CSP_3's identifier and share $(HF_1(ID_3), 160.8864) = (1, 160.8864)$,
      iii. CSP_2's identifier and pseudo share $(HF_1(ID_2), HE_2(pk_{jk}, ID_2)) = (3, 22.5454)$,
      iv. CSP_4's identifier and pseudo share $(HF_1(ID_4), HE_2(pk_{jk}, ID_4)) = (6, 45.0909)$.

      The function from Figure 4.3 is $f_{jkl}(x) = 14.6553 \times x^3 - 131.2159 \times x^2 + 265.1742 \times x + 12.2727$.
   (b) Reconstructed value is $d_{jkl} = f_{jkl}(HF_1(K_d)) = f_{jkl}(HF_1(6)) = f_{jkl}(2)$ $= 14.6553 \times 2^3 - 131.2159 \times 2^2 + 265.1742 \times 2 + 12.2727 = 135$.
   (c) Inner signature is $s\_in_{jkl} = f_{jkl}(HF_1(K_s)) = f_{jkl}(HF_1(7)) = f_{jkl}(0)$ $= 14.6553 \times 0^3 - 131.2159 \times 0^2 + 265.1742 \times 0 + 12.2727 = 12.2727$.
   (d) d_{jkl} is correct since $s\_in_{jkl} = HE_1(d_{jkl}) = HE_1(135) = 135/11 = 12.2727$.

4.3 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 bpVSS uses an attribute value-level signature).

Moreover, we also propose a tree data structure to efficiently exploit outer signatures (Figure 4.5). 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, up to 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, up to the root, which stores the whole DB's signature.

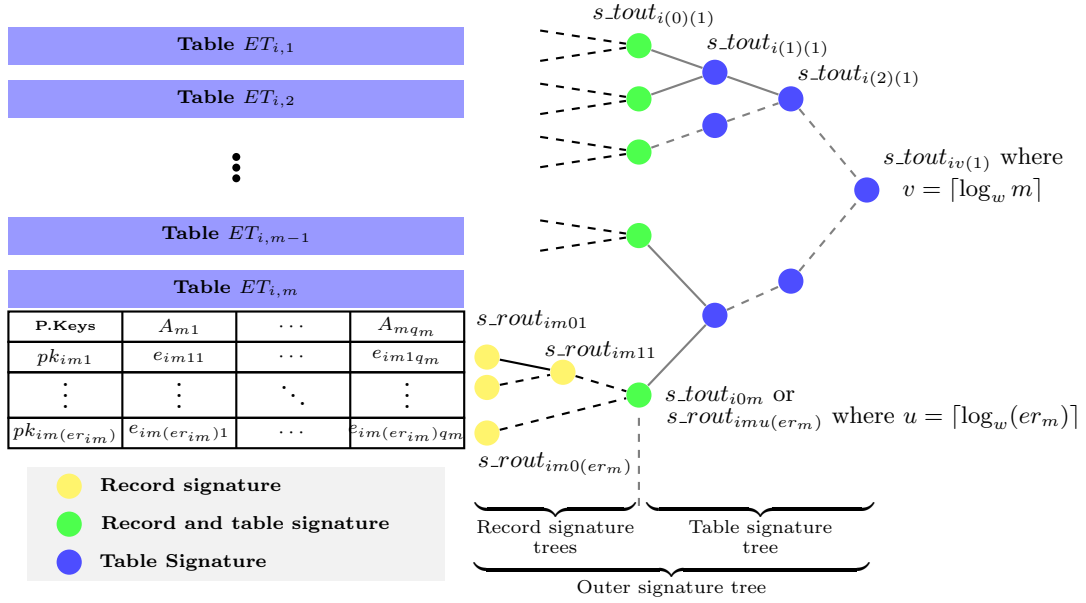


FIGURE 4.5: Outer signature tree at CSP_i

Signatures are checked before reconstructing data (Section 4.2.3). Data integrity can also be verified on-demand. Outer record signatures are created with the help of one-way functions $OF_{iau}(ER_{ijg})$ and $IE_i(h)$. OF_{iau} returns a small integer, and thus helps reduce outer signature volume. IE helps outer signatures be incrementally updated whenever a shared DW is refreshed. Therefore, updating outer signatures speeds up and computation cost is minimized.

Our tree data structure helps outer signatures be easily accessed and updated.

Moreover, the relationship between an outer signature and its shared-record and shared-table combination is easily materialized by their relationship in the tree. Outer signatures are independently created from different combinations of shared records or shared tables with several encryption functions. Hence, each outer signature is unique and independently verified on-demand. There are five verification types: record verification, record-group verification, table verification, table-group verification, and DW verification. Each type is verified as follows.

1. A shared record ER_{ijg} passes integrity check if $s\_rout_{ij0g} = OF_{i00}(ER_{ijg})$.
2. A group of shared record signatures $\{ER_{ijg}\}_{g\in[g_s,g_e]}$ passes integrity check if $s\_rout_{ijuv} = IE_i\left(\sum_{g=g_s}^{g_e} OF_{i0u}\left(ER_{ijg}\right)\right)$, where $g_s = (v-1)\times(w_i)^u+1$, $g_s = \min(v\times(w_i)^u, er_{ij})$ and er_{ij} is the number of shared records in shared table ET_{ij}.
3. A shared table ET_{ij} passes integrity check if $s\_rout_{iju1} = s\_tout_{i0j} = IE_i\left(\sum_{g=1}^{er_{ij}} OF_{i0u}\left(ER_{ijg}\right)\right)$, where $u = \lceil \log_{w_i} er_{ij} \rceil$.
4. A group of shared tables $\{ET_{ij}\}_{j\in[j_s,j_e]}$ passes integrity check if $s\_tout_{iuv} = IE_i\left(\sum_{j=j_s}^{j_e}\sum_{g=1}^{er_{ij}} OF_{i1u}\left(ER_{ijg}\right)\right)$, where $j_s = (v-1)\times(w_i)^u + 1$ and $j_e = \min(v\times(w_i)^u, m)$.
5. A DW at CSP_i passes integrity check if $s\_tout_{iu1} = IE_i\left(\sum_{j=1}^{m}\sum_{g=1}^{er_{ij}} OF_{i1u}\left(ER_{ijg}\right)\right)$, where $u = \lceil \log_{w_i} m \rceil$ and m is the number of tables of the DW at CSP_i.

To improve data integrity, shares should be verified from at least two signature levels. Signature at different levels are created with different functions. Thus, there is a very high probability that incorrect shares are detected by at least one function (Section 5.3.2.3.1). Hence, the rate of incorrect data not being detected (false positives) is reduced. For example, shared table ET_{ij} should be checked with its signature s_rout_{iju1} and checked again with all child signatures $s\_rout_{ij(u-1)v}$, where $u = \lceil \log_{w_i} er_{ij} \rceil$. Any erroneous record can be discovered by recursively verifying signatures in child nodes down to a leaf.

The remainder of this section is organized as follows. Section 4.3.1 specifies parameter settings and functions related to outer signature insertion and update. Sections 4.3.2, 4.3.3 and 4.3.4 describe each step of shared table creation, shared record insertion and shared record update, respectively. Finally, Section 4.3.5 exemplifies the whole process.

4.3.1 Setup

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

1. Determine w_i (the maximum number of child nodes of any node in the signature tree at CSP_i).
2. Define distinct one-way functions $OF_{iau}(ER_{ijg})$. a is a signature type: 0 is a record signature and 1 is a table signature. u is the level of a node in a record or table signature tree. ER_{ijg} is a shared record. OF_{iau} returns a small integer to reduce storage cost.
3. Define a one-way function $IE_i(h)$ that supports incremental addition and subtraction: $IE_i(h_1 \pm h_2) = IE_i\left(IE_i(h_1) \pm IE_i(h_2)\right) = IE_i\left(IE_i(h_1) \pm h_2\right)$. IE_i inputs OF_{iau}'s return value and outputs a record or tree signature.

4.3.2 Shared Table Creation

Whenever a new shared 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 $s\_tout_{i0j} = IE_i(0)$ 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 $s\_tout_{iu1} = IE_i\left(\sum_{b=1}^{j-1} \sum_{g=1}^{er_{ib}} OF_{i1u}\left(ER_{ibg}\right)\right)$.
   (c) Otherwise, stop recursion.

4.3.3 Shared Record Insertion

Whenever a new shared 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 $s\_rout_{ij0g} = OF_{i00}(ER_{ijg})$ 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 $s\_rout_{ijuv} = IE_i\left(OF_{i0u}(ER_{ijg})\right)$.

   (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 $s\_rout_{iju1} = IE_i\left(\sum_{b=1}^{g} OF_{i0u}\left(ER_{ijb}\right)\right)$.

   (c) Otherwise, update the parent node's signature s_rout_{ijuv} with $IE_i\left(s\_rout_{ijuv} + OF_{i0u}(ER_{ijg})\right)$.

3. Starting from the root of ET_{ij}'s record signature tree, which is also leaf s_tout_{i0j} 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 $IE_i\left(s\_tout_{iuv} + OF_{i1u}(ER_{ijg})\right)$.

4.3.4 Shared Record Update

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

1. 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 $IE_i\left(s\_rout_{ijuv} - OF_{i0u}(ER_{ijg,(old)}) + OF_{i0u}(ER_{ijg,(new)})\right)$.

2. 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 $IE_i\left(s\_tout_{iuv} - OF_{i1u}(ER_{ijg,(old)}) + OF_{i1u}(ER_{ijg,(new)})\right)$.

4.3.5 Recapitulative Example

4.3.5.1 Shared Table Creation

Table 4.1 shows sample updates of a table signature tree when five shared tables are created at CSP_i. Let us assign parameter $w_i = 3$ and define function $IE_i(h) = h \mod 11$. Steps in Table 4.1 refer to steps in Section 4.3.2. Red nodes are new. When shared tables $ET_{i(3)}$ and $ET_{i(5)}$ are inserted, only a new node is inserted in the right-most leaf node, and then it is connected with the right-most node of the upper level. When shared

tables $ET_{i(2)}$ and $ET_{i(4)}$ are inserted; a new parent and/or new root must be inserted. Note that signatures in all nodes value zero because there are no shared record in the shared tables and $IE_i(0) = 0 \bmod 11 = 0$.

4.3.5.2 Shared Record Insertion

Figure 4.6 shows sample updates of an outer signature tree when four shared records are inserted into shared table $ET_{i(3)}$ at CSP_i. Let us assign parameter $w_i = 3$ and define functions as follows:

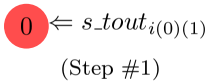
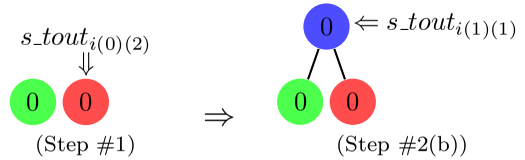
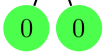
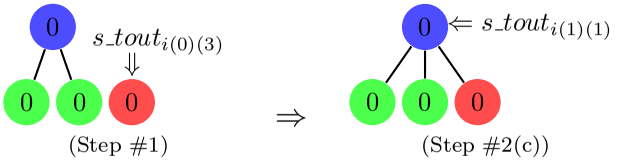
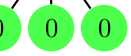
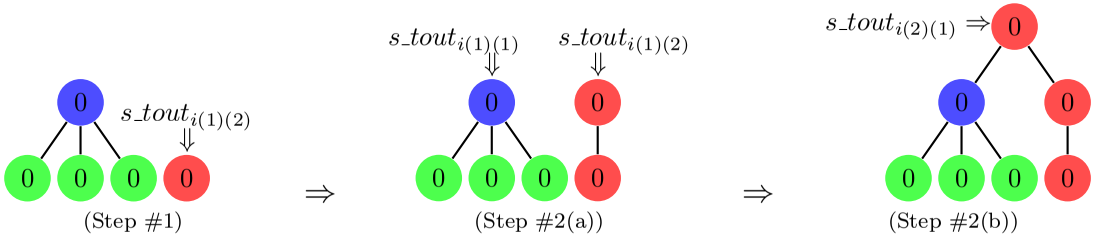
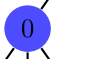
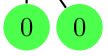
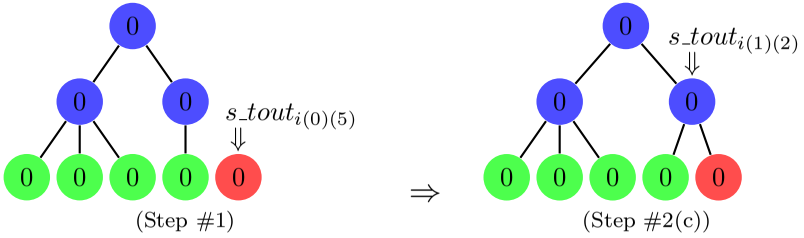
- $IE_i(h) = h \bmod 11$,
- $OF_{i00}(ER_{ijg}) = \lceil ER_{ijg} \rceil \bmod 3$,
- $OF_{i01}(ER_{ijg}) = \lceil ER_{ijg} \rceil \bmod 5$,
- $OF_{i02}(ER_{ijg}) = \lceil ER_{ijg} \rceil \bmod 7$,
- $OF_{i11}(ER_{ijg}) = \lfloor ER_{ijg} \rfloor \bmod 5$,
- $OF_{i12}(ER_{ijg}) = \lfloor ER_{ijg} \rfloor \bmod 7$.

Suppose shared table $ET_{i(3)}$ has a primary key and one attribute, and record values as follows: $ER_{i(3)(1)} = \{1, 3.17\}$, $ER_{i(3)(2)} = \{2, 11.12\}$, $ER_{i(3)(3)} = \{3, 1.81\}$ and $ER_{i(3)(4)} = \{4, 5.00\}$. Each subfigure of Figure 4.6 shows a path for updating an outer signature tree from a new leaf node to the root node. Red nodes are new and orange nodes are updated. All nodes in the update paths of the a table signature tree are incrementally updated. However, nodes in the update path of the record signature tree may either be updated or inserted. For instance, when shared record $ER_{i(3)(1)}$ or $ER_{i(3)(3)}$ is inserted, only one node in the record signature tree is updated/inserted and merged with its parent node. In contrast , when shared record $ER_{i(3)(2)}$ or $ER_{i(3)(4)}$ is inserted, a right-most leaf node and all its parent nodes are inserted.

4.3.5.3 Shared Record Update

Table 4.7 shows a sample update of the outer signature tree from Figure 4.6(e) when a shared record is updated. Let shared record $ER_{i(3)(3)}$ in shared table $ET_{i(3)}$ at CSP_i be updated from $ER_{i(3)(3)} = \{3, 1.81\}$ to $ER_{i(3)(3)} = \{3, 4.95\}$. All nodes (featured in orange) in the update path of the outer signature tree are incrementally updated. Parameter and functions are assigned as in Section 4.3.5.2.

TABLE 4.1: Sample outer signature tree update upon table creation

Shared Table	**Table-signature tree**	**Updated table-signature tree**
#1		
#2	$(u = 0) \Rightarrow$ 	
#3	$(u = 1) \Rightarrow$ $(u = 0) \Rightarrow$ 	
#4	$(u = 1) \Rightarrow$ $(u = 0) \Rightarrow$ 	
#5	$(u = 2) \Rightarrow$  $(u = 1) \Rightarrow$  $(u = 0) \Rightarrow$ 	

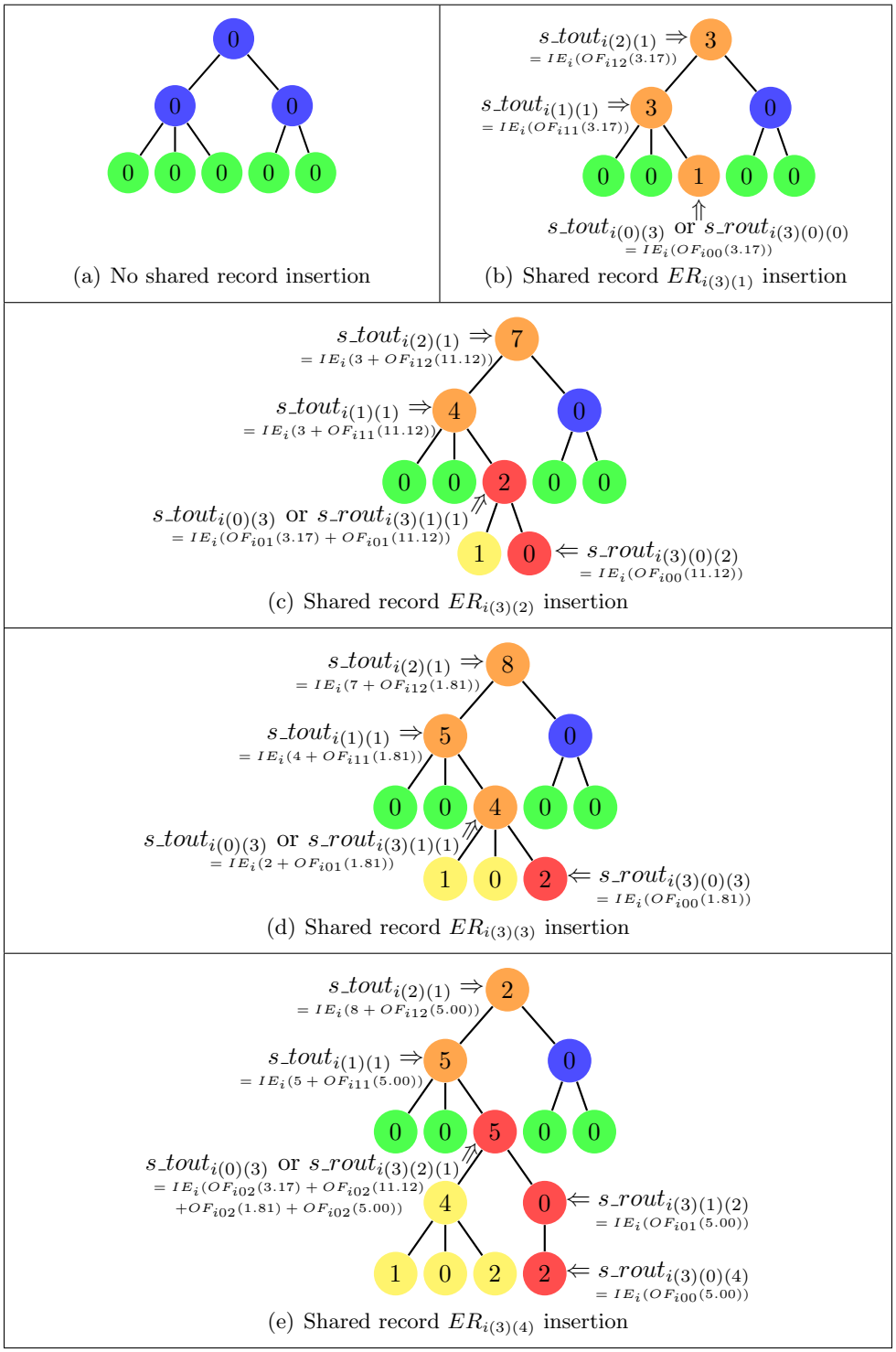


FIGURE 4.6: Sample outer signature tree update upon record insertion

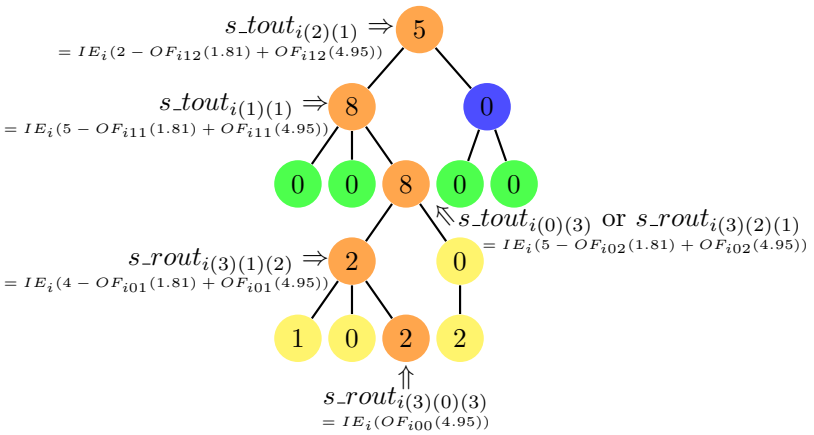


FIGURE 4.7: Sample outer signature tree update upon record update

4.4 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 attribute types are transformed into reals by the data sharing process. Unlike in bpVSS, outer signatures are stored apart from shared tables, i.e., in the signature tree.

Since fewer than n shared records encrypt any original record, fewer than n shared tables store them. Hence, the number of records in any shared table is lower than that of the original table, and differs from that of shares at other CSPs'. To enhance query performance and reduce computation and storage costs, numbers of shared records in shared table should be adjusted to respect with CSP pricing policies. Storage and computation cost models are detailed in Section 5.3.3 and experiments are presented in Section 6.3.

Finally, to improve query performance, computation and data transfer costs, DB and ROLAP operations can be performed on shares with the help of indices and cloud cubes, which are described in the following sections.

4.4.1 Indices

Let us refer back to Figure 4.1. 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 4.2), Type II and III indices are specifically aimed at enhancing query performance and thus computation cost.

Type II indices are customarily used by some secure database approaches (Section 2.2) [21, 57, 60]. 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.

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.

4.4.2 Cloud Cubes

As bpVSS, fVSS supports the storage of data cubes that optimize response time and bandwidth when performing ROLAP operations. In addition, in fVSS, 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. However, cloud cubes can be refreshed even if some CSPs fail. 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 4.8 features a cloud cube named Cube-II 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-II without reconstruction. Note that the cloud cube structure of fVSS is similar to that of bpVSS. However, shares are reals and outer signatures are not stored in cloud cubes.

Foreign Keys						
Time Keys			Product Keys		Aggregate Shares	
YearID	**MonthID**	**DateID**	**CategoryID**	**ProdNo**	**TotalPrice**	**Number**
NULL	*NULL*	*NULL*	*NULL*	*NULL*	83231.9304	58244.8340
NULL	*NULL*	*NULL*	1	*NULL*	26701.0484	18254.3983
NULL	*NULL*	*NULL*	1	1	8958.2892	7113.3832
NULL	*NULL*	*NULL*	1	⋮	⋮	⋮
NULL	*NULL*	*NULL*	1	2	4348.3043	1844.2832
NULL	*NULL*	*NULL*	⋮	⋮	⋮	⋮
1	*NULL*	*NULL*	*NULL*	*NULL*	44574.2802	54542.2232
1	*NULL*	*NULL*	1	*NULL*	21158.2382	8954.3290
1	*NULL*	*NULL*	1	1	9754.3802	4544.1920
1	*NULL*	*NULL*	1	⋮	⋮	⋮
1	*NULL*	*NULL*	2	1	18444.2823	5747.2832
1	*NULL*	*NULL*	⋮	⋮	⋮	⋮
1	1	*NULL*	*NULL*	*NULL*	8312.2383	5812.2931
1	1	*NULL*	1	*NULL*	2312.8234	1822.9020
1	1	*NULL*	1	1	988.9019	586.8936
1	1	*NULL*	1	⋮	⋮	⋮
1	1	*NULL*	2	1	756.9033	458.8023
1	1	*NULL*	⋮	⋮	⋮	⋮
1	2	*NULL*	*NULL*	*NULL*	9758.2382	6254.0098
1	⋮	*NULL*	⋮	⋮	⋮	⋮
1	1	1	*NULL*	*NULL*	2578.3280	1587.3792
1	1	1	1	*NULL*	548.8923	425.2382
1	1	1	1	1	56.8032	24.9082
1	1	1	1	⋮	⋮	⋮
1	1	1	1	2	95.0882	67.8923
1	1	1	1	⋮	⋮	⋮
1	1	1	2	*NULL*	689.8433	357.9877
1	1	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮

FIGURE 4.8: Sample cloud cube Cube-II

4.5 Loading, Backup and Recovery Processes

4.5.1 Loading Data

As in bpVSS, 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. However, in fVSS, new data can be loaded even though some CSPs fail. For instance, in Figure 4.9, data from Figure 4.2 are already shared and the two last records (records #127 and #128) are new. Data can be loaded even if CSP_1 fails. Moreover, after new shared records are loaded, the outer signature tree must be updated (Section 4.3), and indices and possible cloud cubes refreshed. Note that all shares of ProdName and ProdDescr which are string attributes in Figures 4.2 and 4.9 can be compressed to reduce share volume.

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
124	Shirt	Red	1	135
125	Shoe	*NULL*	2	142
126	Ring	*NULL*	1	142
127	Hat	*NULL*	3	25
128	Dress	Red	1	139

(a) Original data

ProdNo	Share location
124	10101
125	01110
126	11010
127	00111
128	01110

(b) Type I indices

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
124	{-46.1288 ; -72.0606 ; -73.2955 ; -84.4091 ; -86.8788}	{-44.8939 ; -68.3561 ; -67.1212}	1	-110.3409
126	{-106.1818 ; -224.3182 ; -250.0000 ; -214.0455}	*NULL*	1	-414.3636

(c) Shares at CSP_1

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
125	{81.4636 ; 91.8682 ; 95.3366 ; 90.3818}	*NULL*	2	110.6955
126	{104.8182 ; 139.8409 ; 147.4545 ; 136.7955}	*NULL*	1	196.1818
128	{75.0000 ; 97.7909 ; 91.3500 ; 98.2864 ; 98.2864}	{517.2000 ; 448.8000 ; 453.3000}	1	110.1773

(d) Shares at CSP_2

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
124	{94.5732 ; 121.3535 ; 122.6288 ; 134.1061 ; 136.6566}	{93.2980 ; 117.5278 ; 116.2525}	1	160.8864
125	{59.1273 ; 80.7000 ; 87.8909 ; 77.6182}	*NULL*	2	119.7364
127	{83.1212 ; 117.0606 ; 142.8545}	*NULL*	3	19.3152
128	{43.0909 ; 90.3455 ; 76.9909 ; 91.3727 ; 91.3727}	{57.4727 ; 76.9909 ; 75.9636}	1	116.0273

(e) Shares at CSP_3

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
125	{-52.6000 ; -31.9818 ; -25.1091 ; -34.9273}	*NULL*	2	5.3273
126	{-431.0909 ; -786.5455 ; -863.8182 ; -755.6364}	*NULL*	1	-1358.3636
127	{148.7273; 222.8182 ; 279.1273}	*NULL*	3	9.4364
128	{-70.5455; -25.3818 ; -38.1455 ; -24.4000 ; -24.4000}	{-56.8000 ; -38.1455 ; -39.1273}	1	-0.8364

(f) Shares at CSP_4

ProdNo	ProdName	ProdDescr	CategoryID	UnitPrice
124	{-37.0707 ; -57.8586; -58.8485; -67.7576; -69.7374}	{-36.0808; -54.8889 ; -53.8990}	1	-88.5455
127	{-13.6970 ; -30.2121 ; -42.7636}	*NULL*	3	17.3515

(g) Shares at CSP_5

FIGURE 4.9: Example of sharing new data

4.5.2 Updating Indices

All types of indices are updated when new data are loaded. In Figure 4.9(b), Type I indices from Figure 4.2(b) at the index server are already updated. The two last records (#127 and #128) are new. Then, Type II indices are updated at the index server as in other approaches [21, 57, 60]. Type III indices are shared as normal data (Section 4.2.2) and are stored in extra shared attributes at CSPs.

4.5.3 Cloud Cube Creation

New cloud cubes must be created at all n CSPs'. Reconstructing data is unnecessary. At CSP_i's, aggregates are computed from shares and related indices. Figure 4.10 shows an example of cube named Cube-III that is created from shares and indices of table PRODUCT (Figure 4.2).

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	-513.3409	-110.3409	11.3636	12.5739
1	*NULL*	-524.7045	-110.3409	-414.3636	9.6420
2	*NULL*	11.3636	11.3636	11.3636	9.2197
1	124	-110.3409	-110.3409	-110.3409	1.8636
1	126	-414.3636	-414.3636	-414.3636	-5.6818
2	125	11.3636	11.3636	11.3636	6.4924

(a) Shares at CSP_1

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	329.4227	22.5455	110.6955	7.4943
1	*NULL*	218.7273	22.5455	196.1818	5.5398
2	*NULL*	110.6955	110.6955	110.6955	4.1818
1	124	22.5455	22.5455	22.5455	-1.0136
1	126	196.1818	196.1818	196.1818	-3.8500
2	125	110.6955	110.6955	110.6955	3.0909

(b) Shares at CSP_2

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	314.9864	160.8864	119.7364	-0.0852
1	*NULL*	195.2500	160.8864	34.3636	-0.3352
2	*NULL*	119.7364	119.7364	119.7364	-0.8157
1	124	160.8864	160.8864	160.8864	1.9091
1	126	34.3636	34.3636	34.3636	3.5455
2	125	119.7364	119.7364	119.7364	-0.2702

(c) Shares at CSP_3

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	-1307.9455	45.0909	5.3273	9.0909
1	*NULL*	-1313.2727	45.0909	-1358.3636	6.9091
2	*NULL*	5.3273	5.3273	5.3273	8.3636
1	124	45.0909	45.0909	45.0909	2.5455
1	126	-1358.3636	-1358.3636	-1358.3636	4.7273
2	125	5.3273	5.3273	5.3273	6.1818

(d) Shares at CSP_4

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	25.5455	-88.5455	56.8182	11.3636
1	*NULL*	-31.2727	-88.5455	57.2727	8.6364
2	*NULL*	56.8182	56.8182	56.8182	7.3737
1	124	-88.5455	-88.5455	-88.5455	-2.5091
1	126	57.2727	57.2727	57.2727	-7.3091
2	125	56.8182	56.8182	56.8182	5.1919

(e) Shares at CSP_5

FIGURE 4.10: Sample cloud cube Cube-III

There are four cases when computing aggregate shares. They are detailed in the following sections.

4.5.3.1 MAX, MIN, MEDIAN and MODE

For aggregations by MAX, MIN, MEDIAN and MODE, 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.

4.5.3.2 COUNT

COUNT aggregates can be extracted from a Type II index or directly computed from a share. $t-2$ pseudo shares of the aggregate are random numbers. $n-t+2$ shares of the aggregate are computed from the aggregate itself, its signature, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (Section 4.2.2). All pseudo shares and shares are stored in the cloud cube.

4.5.3.3 SUM

For SUM, thanks to the homomorphism property, aggregate shares are incrementally computed by summing shares and pseudo shares at each CSP's. For example, to compute SUM(X) at CSP_i, the aggregate share is the sum of shares and pseudo shares (Equation 4.2), 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 (i.e, Query #6.3 in Figure 4.15).

$$SUM(X) = SUM_{CSP_i}(X) + HE_2(SUM_{index}(PK_i), ID_i) \tag{4.2}$$

4.5.3.4 Other Cases

More complex aggregations require combining the above cases. For example, SUM($X+Y$) and SUM($X-Y$) are computed from shares and pseudo shares by Equations 4.3 and 4.4, respectively, where $\text{SUM}_{CSP_i}(X \pm Y)$ is trivially computed at CSP_i and $SUM_{index}(PK_i)$ is computed with the help of a Type I index.

$$SUM(X+Y) = SUM_{CSP_i}(X+Y) + 2 \times HE_2(SUM_{index}(PK_i), ID_i) \tag{4.3}$$

$$SUM(X-Y) = SUM_{CSP_i}(X-Y) \tag{4.4}$$

4.5.3.5 Recapitulative Example

Let us refer back to Figure 4.10. The first record in Cube-III stores aggregate (SUM(UnitPrice), MIN(UnitPrice), MAX(UnitPrice) and COUNT(UnitPrice), respectively) shares for all products and all categories. Each aggregate share is computed from shares and indices (Figure 4.2) as follows.

For SUM, aggregate shares are independently computed. For example, at CSP_1 and CSP_5, total price are computed as follows.

- At CSP_1, we sum shirt and ring prices in records #124 and #126 (-110.3409 and -414.3636, respectively) and pseudo share of shoe price in record #125 ($HE_2(125, ID_1) = HE_2(125, 1) = 125 \times 1/11 = 11.3636$). Hence, the aggregate share is -110.3409-414.3636+11.3636=-513.3409.
- At CSP_5, we sum shirt price in record #124 (-88.5455) and pseudo shares of shoe and ring prices in records #125 and #126 ($HE_2(125+126, ID_5) = HE_2(251, 5) = 251 \times 5/11 = 114.0909$), respectively. Hence, the aggregate share is -88.5455+114.0909=25.5455.

For MAX, maximum price is 142 which is the shoe and the ring prices stored in record #125 and #126, respectively (discovered by Type-II index). Let us select only shoe price in record #125. Then, shares 110.6955, 119.7364 and 5.3273 of shoe price are shared at CSP_2, CSP_3, and CSP_4, respectively. CSP_1 and CSP_5's pseudo shares of shoe price are computed by homomorphic function HE_2 (Section 4.2.2). CSP_1's pseudo share is $HE_2(125, ID_1) = HE_2(125, 1) = 125 \times 1/11 = 11.3636$. CSP_5's pseudo share is $HE_2(125, ID_5) = HE_2(125, 5) = 125 \times 5/11 = 56.8182$. All pseudo shares and shares are stored in Cube-III.

For COUNT, the number of products is 3. It is found from a Type II index. Let CSP_4 and CSP_5's pseudo shares of 3 be 9.0909 and 11.3636, respectively (random numbers). Then, CSP_1, CSP_2 and CSP_3's shares (12.5739, 7.4943 and -0.0852, respectively) of 3 are computed from 3, the signature of 3 =0.2727, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (example in Section 4.2.4.2). All pseudo shares and shares are stored in table Cube-III. All pseudo shares and shares are stored in Cube-III.

4.5.4 Refreshing Cloud Cubes

Thanks to the homomorphism property, aggregate shares in cubes can be incrementally updated from new shares and indices. To refresh a cloud cube, there are two

scenarios: a normal scenario where all CSPs are available and an abnormal scenario where some CSPs are not.

4.5.4.1 Normal Scenario

In this scenario, all CSPs are up. Hence, shared cubes must be updated at all CSPs. For instance, if new records #127 and #128 show up (Figure 4.9), Cube-III (Figure 4.10) must be refreshed, resulting in the cloud cube featured in Figure 4.11. The processes to refresh aggregate shares are presented in the following sections.

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	-490.1591	11.5455	11.3636	17.8314
1	*NULL*	-513.0682	-110.3409	-414.3636	19.5909
2	*NULL*	11.3636	11.3636	11.3636	6.4924
3	*NULL*	11.5455	11.5455	11.5455	-1.8636

(a) Shares at CSP_1

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	462.6909	23.0909	110.6955	10.5852
1	*NULL*	328.9045	22.5455	110.6955	10.5227
2	*NULL*	110.6955	110.6955	110.6955	4.1818
3	*NULL*	23.0909	23.0909	23.0909	-1.0136

(b) Shares at CSP_2

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	450.3288	19.3152	119.7364	0.9198
1	*NULL*	311.2773	160.8864	34.3636	-1.6364
2	*NULL*	119.7364	119.7364	119.7364	-0.8157
3	*NULL*	19.3152	19.3152	19.3152	1.9091

(c) Shares at CSP_3

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	-1299.3455	9.4364	5.3273	15.2727
1	*NULL*	-1314.1091	45.0909	-1358.3636	13.8182
2	*NULL*	5.3273	5.3273	5.3273	8.3636
3	*NULL*	9.4364	9.4364	9.4364	2.5455

(d) Shares at CSP_4

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	101.0788	17.3515	56.8182	15.5657
1	*NULL*	26.9091	-88.5455	57.2727	17.2727
2	*NULL*	56.8182	56.8182	56.8182	7.3737
3	*NULL*	17.3515	17.3515	17.3515	-2.5091

(e) Shares at CSP_5

FIGURE 4.11: Cube-III after refreshing

4.5.4.1.1 MAX, MIN, MEDIAN and MODE

For aggregations by MAX, MIN, MEDIAN and MODE, aggregate shares are updated only if new aggregate values differ from old aggregates. The process to refresh aggregate shares is the same as in Section 4.5.3.1.

4.5.4.1.2 COUNT

COUNT aggregates are incrementally added to shares or pseudo shares of the number of new data records. $t-2$ pseudo shares are random numbers. $n-t+2$ shares are computed from the number of new records, its signature, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (Section 4.2.2).

4.5.4.1.3 SUM

For SUM, at each CSP's, aggregate shares are independently incrementally added to the sum of new shares and pseudo shares. For example, to update SUM(X) at CSP_i, the aggregate share is the summation of the old aggregate share, new shares and new pseudo shares (Equation 4.5), where A is the old aggregate share, $\text{SUM}_{CSP_i}(X_{new})$ is summation of new shares at CSP_i and $SUM_{index,new}(PK_i)$ is computed with the help of a Type I index.

$$SUM_{CSP_i}(X) = A + SUM_{CSP_i}(X_{new}) + HE_2(SUM_{index,new}(PK_i), ID_i) \tag{4.5}$$

4.5.4.1.4 Other Cases

More complex aggregations require combining the above. For example, SUM($X+Y$) and SUM($X-Y$) are computed from the old aggregate shares, new shares and new pseudo shares by Equations 4.6 and 4.7, respectively, where $\text{SUM}_{CSP_i}(X_{new} \pm Y_{new})$ is trivially computed from only new data at CSP_i and $SUM_{index,new}(PK_i)$ is computed from only new data records with the help of a Type I index. Note that $2 \times HE_2(SUM_{index,new}(PK_i), ID_i)$ is equal to value of the sum of CSP_i's pseudo shares of all X_{new} and all Y_{new}.

$$SUM(X+Y) = A+SUM_{CSP_i}(X_{new}+Y_{new})+2\times HE_2(SUM_{index,new}(PK_i), ID_i) \tag{4.6}$$

$$SUM(X-Y) = A + SUM_{CSP_i}(X_{new} - Y_{new}) \tag{4.7}$$

4.5.4.1.5 Recapitulative Example

Let us refer back to Figure 4.11. The fist record in Cube-III stores aggregate (SUM(UnitPrice), MIN(UnitPrice), MAX(UnitPrice) and COUNT(UnitPrice), respectively) shares for all products and all categories. Each aggregate share is computed from old aggregate shares (Figure 4.10) and original data (before sharing), shares and indices (Figure 4.9).

For SUM, aggregate shares are independently refreshed at each CSP's. For example, at CSP_2's, total price is incrementally updated by summing the share and the pseudo share of hat and dress prices stored in new records #127 and #128. It is computed as 329.4227+110.1773+23.0909 = 462.6909, where 329.4227 is the old aggregate share, 110.1773 is the share of dress price in record #128 and 23.0909 is the pseudo share of hat price in record #127.

For MIN, the lowest price is changed to 25, i.e., hat price stored in record #127 (obtained from a Type-II index). Next, shares 19.3152, 9.4364 and 17.3515 of hat price are shared at CSP_3, CSP_4 and CSP_5, respectively. CSP_1 and CSP_2's pseudo shares of hat price are computed by homomorphic function HE_2 (Section 4.2.2) at the user's and are shared at CSP_1 and CSP_2, respectively. CSP_1's pseudo share is $HE_2(127, ID_1) = HE_2(127, 1) = 127 \times 1/11 = 11.5455$. CSP_2's pseudo share is $HE_2(127, ID_2) = HE_2(127, 2) = 127 \times 2/11 = 23.0909$.

For COUNT, the number of new products (hats and dresses) is 2. Let CSP_2 and CSP_4's pseudo shares of 2 be 3.0909 and 6.1818, respectively (random numbers). CSP_1 CSP_3 and CSP_5's shares (5.2576, 1.0051 and 4.2020, respectively) of 2 are computed from 2, the signature of 2 =0.1818, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (Section 4.2.4.2). Examples of new aggregate shares are computed as follows.

- CSP_1's new aggregate share is the sum of the share of 2 and the old aggregate share (5.2576+12.5739=17.8314).
- CSP_2's new aggregate share is the sum of the pseudo share of 2 and the old aggregate share (3.0909+7.4943=10.5852).

4.5.4.2 Abnormal Scenario

In this scenario, some CSPs are down. Hence, shared cubes are updated at only some CSPs.For instance, if new records #127 and #128 show up (Figure 4.9), Cube-III (Figure 4.10) at only CSP_2, CSP_3, CSP_4 and CSP_5 must be refreshed since CSP_1 fails,

resulting in the cloud cube featured in Figure 4.12. The processes to refresh aggregate shares are presented in the following sections.

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	-513.3409	-110.3409	11.3636	12.5739
1	*NULL*	-524.7045	-110.3409	-414.3636	9.6420
2	*NULL*	11.3636	11.3636	11.3636	9.2197

(a) Shares at offline CSP_1

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	332.5136	3.0909	110.6955	10.5852
1	*NULL*	222.1818	22.5455	196.1818	8.9943
2	*NULL*	110.6955	110.6955	110.6955	4.1818
3	*NULL*	1.2727	2.3636	3.0909	3.4545

(b) Shares at online CSP_2

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	535.5682	24.5227	119.7364	0.5693
1	*NULL*	381.6500	160.8864	34.3636	-1.2807
2	*NULL*	119.7364	119.7364	119.7364	-0.8157
3	*NULL*	33.0909	32.3636	31.8788	-0.9455

(c) Shares at online CSP_3

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	-834.2727	-203.0909	5.3273	2.6545
1	*NULL*	-915.1455	45.0909	-1358.3636	-3.9455
2	*NULL*	5.3273	5.3273	5.3273	8.3636
3	*NULL*	69.0000	64.6364	61.7273	-10.8545

(d) Shares at online CSP_4

CategoryID	ProdNo	Total Price	Min Price	Max Price	Count
NULL	*NULL*	-78.6727	-41.8182	56.8182	14.1636
1	*NULL*	-118.4909	-88.5455	57.2727	12.5818
2	*NULL*	56.8182	56.8182	56.8182	7.3737
3	*NULL*	-14.8182	-13.3636	-12.3939	3.9455

(e) Shares at online CSP_5

FIGURE 4.12: Cube-III after refreshing when CSP_1 fails

4.5.4.2.1 MAX, MIN, MEDIAN and MODE

For aggregations by MAX, MIN, MEDIAN and MODE, aggregate shares are pseudo shares and shares of the new aggregate. Pseudo shares are the offline CSPs' old aggregate shares which are reconstructed from other online CSPs' old aggregate shares. If the number of the pseudo shares is lower than $t-2$, other pseudo shares are random numbers. $n-t+2$ shares of the aggregate are computed from the aggregate itself, its signature, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (Section 4.2.2). All random pseudo shares and all shares are refreshed in the cloud cube at online CSPs.

4.5.4.2.2 COUNT

For COUNTs, aggregates are incrementally added to shares or pseudo shares of the number of new data records. Offline CSPs' pseudo shares are zero. If the number of the pseudo shares is lower than $t-2$, other pseudo shares are random numbers. $n-t+2$ shares are computed from the number of new data records, its signature, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (Section 4.2.2). All random pseudo shares and all shares are incrementally added to old aggregate shares in the cloud cube at online CSPs.

4.5.4.2.3 SUM

For SUMs, at only online CSPs', aggregate shares are incrementally updated with shares or pseudo shares. Offline CSPs' pseudo shares are zero. If the number of pseudo shares is lower than $t-2$, other pseudo shares are random numbers. $n-t+2$ shares are computed from the sum of new data, its signature, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (Section 4.2.2). All random pseudo shares and all shares are incrementally added to old aggregate shares in the cloud cube at online CSPs.

4.5.4.2.4 Other Cases

More complex aggregations require combining the above. For example, for SUM($X \pm Y$), aggregate shares are incrementally updated with shares or pseudo shares of SUM($X_{new} \pm Y_{new}$) (form a set A_{new} of new records). Offline CSPs' pseudo shares of A_{new} are zero. If the number of the pseudo shares is lower than $t-2$, other pseudo shares are random numbers. The $n-t+2$ shares of A_{new} are computed from A_{new}, its signature, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (Section 4.2.2). All random pseudo shares and all shares are incrementally added to old aggregate shares in the cloud cube at online CSPs.

4.5.4.2.5 Recapitulative Example

Let us refer back to Figure 4.12. The fist record in Cube-III stores aggregate (SUM(UnitPrice), MIN(UnitPrice), MAX(UnitPrice) and COUNT(UnitPrice), respectively) shares for all products and all categories. Each aggregate share is computed from old aggregate shares (Figure 4.10) and original data (before sharing) shares and indices (Figure 4.9) at each online CSP's.

For SUM, the total price of new products is 25+139=146 where 25 and 139 are hat and dress prices, respectively. Let CSP_1's pseudo share of 146 be zero because CSP_1 is offline. Let CSP_2's pseudo share of 146 be 3.0909 (a random number). CSP_3, CSP_4 and CSP_5's shares (220.5818, 473.6727 and -104.2182, respectively) of 146 are computed from 146, the signature of 146 =13.2727, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (example in Section 4.2.4.2). Examples of new aggregate shares are computed as follows.

- CSP_2's new aggregate share is the sum of a pseudo share and the old aggregate share (3.0909+329.4227=332.5136).
- CSP_3's new aggregate share is the sum of a share and the old aggregate share (220.5818+314.9864=535.5682).

For MIN, the lowest price is changed to 25, which is hat price in record #127. Because CSP_1 is offline, CSP_1's pseudo share of 25 is -110.3409, which is CSP_1's old aggregate share reconstructed from other CSPs' old aggregate shares (22.5455, 160.8864, 45.0909 and -88.5455). Let CSP_2's pseudo share of 25 be 3.0909 (a random number). Then, CSP_3, CSP_4 and CSP_5's shares (24.5227, -203.0909 and -41.8182, respectively) of 25 are computed from 25, signature of 25 =2.2727, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (example in Section 4.2.4.2). Pseudo share and all shares are replaced in table Cube-III at online CSPs.

For MAX, maximum price are not updated because all new prices are lower than the old maximum price.

For COUNT, the number of new products (hat and dress) is 2. Let CSP_1's pseudo share of 2 be zero because CSP_1 is offline. Let CSP_2's pseudo share of 2 be 3.0909 (a random number). CSP_3, CSP_4 and CSP_5's shares (0.6545, -6.4364 and 2.8000, respectively) of 2 are computed from 2, the signature of 2 =0.1818, pseudo shares, data and signature keys and CSP identifiers by Lagrange interpolation (example in Section 4.2.4.2). Examples of new aggregate shares are computed as follows.

- CSP_2's new aggregate share is the sum of a pseudo share and the old aggregate share (3.0909+7.4943=10.5852).
- CSP_3's new aggregate share is the sum of a share and the old aggregate share (0.6545-0.0852=0.5693).

4.5.5 Backup and Recovery

In fVSS, as in bpVSS and all secret sharing approaches, backups are unnecessary because each shared table is actually a backup of its other shares. In case shared tables or shared records are detected as erroneous by outer signature verification (Section 4.3), their can be recovered from t other shares and Type I indices. The recovery process in fVSS is similar to that in bpVSS (Section 3.5.3).

4.6 Data Analysis over Shares

As with bpVSS, simple SELECT/FROM queries directly apply onto shares. However, primary keys are required to match query results when reconstructing data, since keys are unique and unencrypted. For example, ProdNo is inserted into query "SELECT ProdName FROM Product" to match the shares of each ProdName.

For JOIN operators, only FULL OUTER JOIN can apply directly on unencrypted keys. Other joins must be transformed to FULL OUTER JOIN, because each record is not shared at all CSPs. Then, query results must be filtered with an external program after reconstruction. For example, when Query 7 (Figure 4.13) is run at CSP_4, record #127 from Figure 4.9 is not retrieved, if categoryID #3 is not shared in shared table CATEGORY. Hence, the INNER JOIN in Query 7 is transformed to an OUTER JOIN to allow retrieving record #127.

```
SELECT P.ProdName, P.ProdNo, C.CategoryName, C.CategoryID
FROM Product AS P INNER JOIN Category AS C
ON P.CategoryID=C.CategoryID
```

FIGURE 4.13: Query 7

When expressing conditions in a WHERE or HAVING clause, Type II indices must be used [21, 57, 60]. Almost all comparison operators (=, $\neq$, EXISTS, IN, $>$, $\geq$, $<$, $\leq$, BETWEEN...) can be evaluated against such B+ trees.

Similarly, aggregation functions such as MAX, MIN and COUNT can directly apply on shares with the help of Type II indices [21, 57, 60]. In contrast, a SUM must combine relevant aggregates of shares and pseudo shares (Equation 4.2) 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. AVG, VAR and STDDEV are computed by Equations 4.8, 4.9 and 4.10, respectively, where X^2 is a Type III

index (shares of square values of attribute X), $DC()$ is the reconstruction process (Section 4.2.3) and $SUM()$ and $COUNT()$ are aggregation (SUM and COUNT, respectively) queries.

$$AVG(X) = DC(SUM(X))/DC(COUNT(X)) \tag{4.8}$$

$$VAR(X) = \frac{DC(SUM(X^2))}{DC(COUNT(X))} + \frac{DC(SUM(X))^2}{DC(COUNT(X))^2} \tag{4.9}$$

$$STDDEV(X) = \sqrt{\frac{DC(SUM(X^2))}{DC(COUNT(X))} + \frac{DC(SUM(X))^2}{DC(COUNT(X))^2}} \tag{4.10}$$

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 in shares and pseudo shares with an external program before reconstruction. For example, $SUM(X+Y)$ and $SUM(X-Y)$ are computed by Equations 4.11 and 4.12, where $SUM_{CSP_i}(X \pm Y)$ is trivially computed at CSP_i by aggregating $SUM(X \pm Y)$ queries, and $SUM_{index}(PK_i)$ is computed with the help of a Type I index.

$$SUM(X+Y) = DC(SUM_{CSP_i}(X+Y) + 2 \times HE_2(SUM_{index}(PK_i), ID_i)) \tag{4.11}$$

$$SUM(X-Y) = DC(SUM_{CSP_i}(X-Y)) \tag{4.12}$$

GROUP BY queries 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 them, depending on their clauses and operators, following the above guidelines. Figure 4.15 shows the way a sample query (Query #6 from Section 3.6.1) runs at the user's, at one index server and at four CSPs' (CSP_1, CSP_2, CSP_3 and CSP_4). Query #6 is recalled in Figure 4.14 and inputs from Figure 4.9.

```
SELECT P.ProdName, AVG(S.Price)
FROM Sale AS S RIGHT JOIN Product AS P
ON S.ProdNo = P.ProdNo
GROUP BY S.ProdNo
WHERE P.UnitPrice = 135
```

FIGURE 4.14: Query 6

Query #6 is split into three queries (Queries #6.1, #6.2 and #6.3)[1]. Query #8.1 is a match query created with the help of a type-II index. It is run at all four CSPs'. Only record #124 matches with Query #6.1. However, this record is shared only at CSP_1 and CSP_3. Thus, CSP_2 and CSP_4's pseudo shares are computed by homomorphic function HE_1 from CSPs identifier and bitmap located on the index server to help reconstruct ProdName in record #124. To compute the average price of each product, the sum of prices and the number of records must be computed first. The number of records is found in a Type II index at the index server's. The sum of prices is reconstructed by summing shares and pseudo shares. Query #6.2 is run at all four CSPs. Finally, query #6.3 is run on Type I indices at the index server's and only sums pseudo shares.

Finally, as bpVSS, fVSS directly supports all basic OLAP operations by directly querying cloud cubes and reconstructing the global result. However, global results are only reconstructed from retrieved shares. To retrieve aggregate shares from cloud cubes, example queries are shown in Table 3.2 (Chapter 3).

4.7 Summary

In this chapter, 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 extended from classic secret sharing [23] to share any piece of data fewer than n times. Since each data piece is shared at only some out of n CSPs, fVSS is the first flexible secret sharing that allows users adjusting share volume with respect to CSP pricing polices. Unbalancing share volume at CSPs' indeed helps minimize storage and computing costs in the pay-as-you-go paradigm by design (discussed in Chapter 5 and experimented in Chapter 6). Privacy and availability are achieved with secret sharing. Moreover, fVSS achieves a higher security level since it can protect data even if all CSPs are malicious (Chapter 5). Moreover, fVSS allows refreshing DWs even though some CSPs fail, it is not necessary to share data at all CSPs' (Chapter 5).

Data integrity is reinforced with both inner and outer signatures that help detect errors in query results and shares, respectively. Inner signatures hidden in shares are checked after reconstruction with a one-way function, as in bpVSS. Outer signatures are stored in a tree data structure. Several types of outer signature verifications (record verification, record-group verification, table verification, table-group verification, and DW verification) are available on-demand. Moreover, outer signatures at different levels in the outer signature tree may be created and verified to reduce the rate of undetected incorrect data, and thus an integrity is improved (Chapter 5).

[1]These queries are different from Queries #6.1 and #6.2 in Chapter 3.

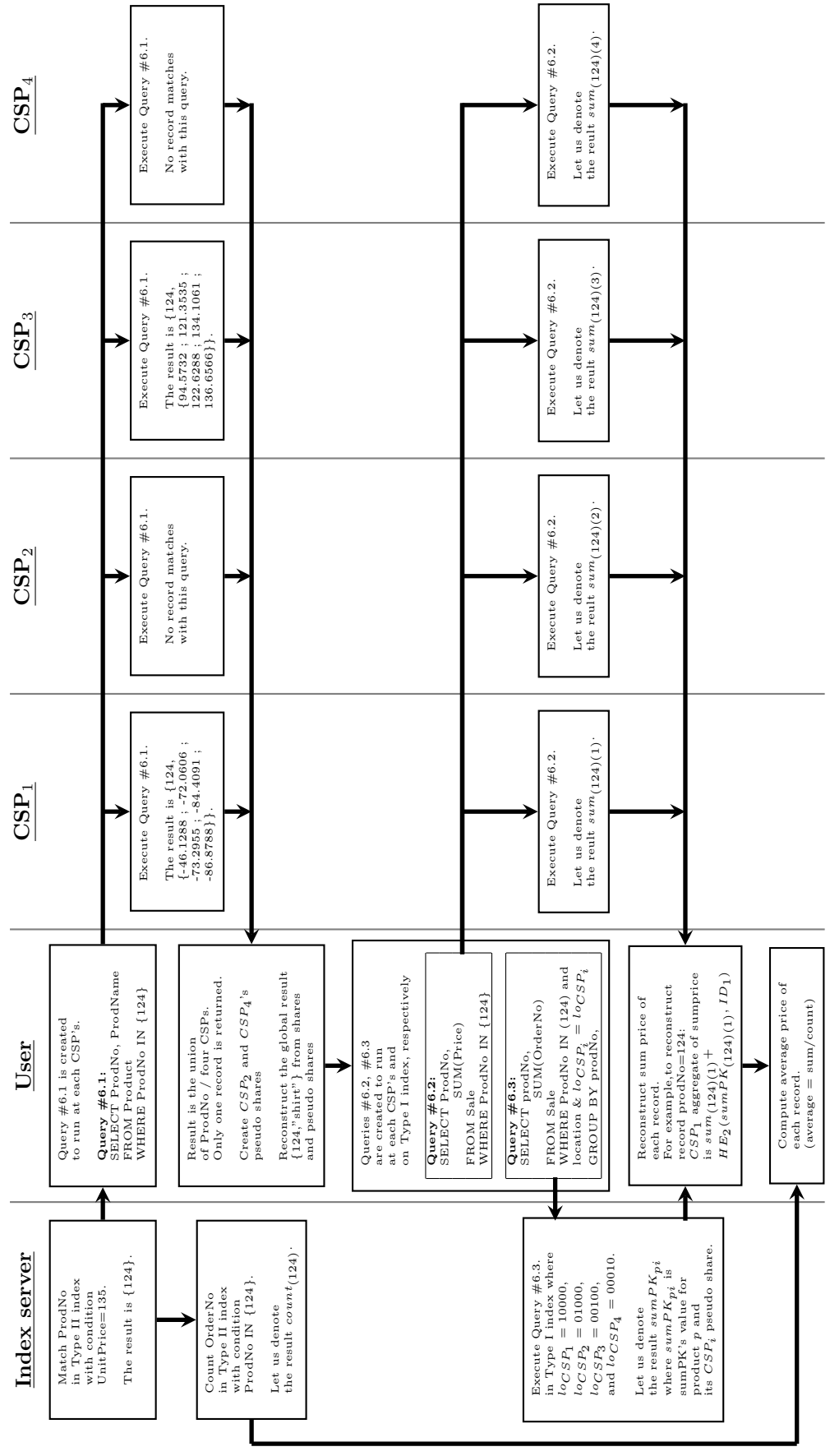


FIGURE 4.15: Example of a complex query execution over shares and index server

As other secure database approaches, fVSS allows data analysis on shares. To analyze data, some traditional SQL queries can be performed on shares at CSPs' without decryption, and then only target results are reconstructed at the user's. This helps reduce communication and computation costs at the user's (Chapter 5). Three types of indices help improve query performance. As in [21, 57, 60], Type II indices are used in exact match, range and aggregation (e.g., MAX, MIN, MEDIAN and COUNT) queries. Type I and III indices are used in some aggregation queries such as SUM, AVG and STDDEV. Moreover, as bpVSS, fVSS supports the storage of data cubes that optimize response time and bandwidth when performing ROLAP operations. However, fVSS creates a shared cube from shares and indices without reconstructing any data first, unlike bpVSS. Finally, shared cubes can be refreshed even though some CSPs fail.

Chapter 5

Security and Performance Analysis of bpVSS and fVSS

5.1 Introduction

In this chapter, we illustrate the relevance of our SSSs along three axes. First, we theoretically and experimentally study the factors that influence performance (the time complexity and execution time of the data sharing and reconstruction processes). Second, we mainly theoretically study the security features of bpVSS and fVSS, which are our primary focus. Third, since our approaches apply in the cloud, we both theoretically and experimentally study the factors that influence cost in the pay-as-you-go paradigm, i.e., computing, storage and data transfer costs. To close the chapter, we compare our SSSs with existing, related approaches and discuss the tradeoff between security and performance.

5.2 bpVSS

5.2.1 Complexity

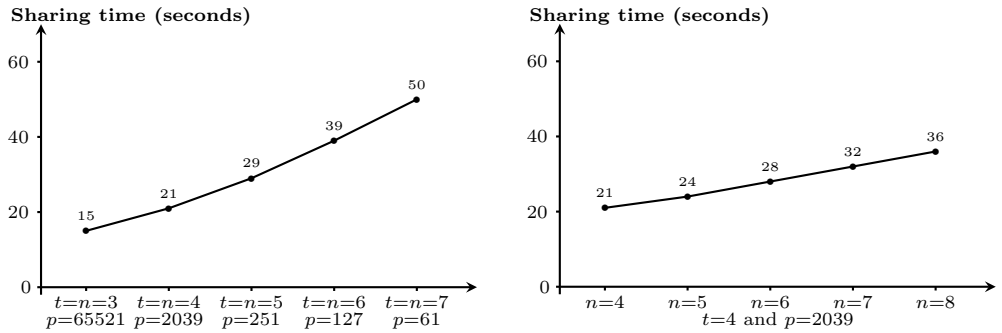
Time complexity helps understanding the factors that influence execution time. This section describes the time complexities of the data sharing and reconstruction processes at the user's side. In addition to theoretical considerations, we run 100 1 GB test cases made of random 32-bit signed integers and vary parameter n and t. Experiments are conducted with Bloodshed Dev-C++ 5.5.3 on a PC with an Intel(R) Core(TM) i5 2.76 GHz processor with 3 GB of RAM running Microsoft Windows 7.

5.2.1.1 Data Sharing Process

To share a decimal integer, it is transformed into a base-p integer of $t-1$ digits, and then n t-variables linear equations are computed. Thus, the time complexity is $t + nt \approx O(nt)$.

Moreover, two signature types (inner and outer signatures) are created. Inner signatures are created by a user-defined homomorphic function, thus the time complexity is $O(H_{in})$, where H_{in} is the time to create an inner signature with the homomorphic function. The outer signature of each share is created independently by a user-defined hash function. Thus, to create the outer signatures of n shares, the time complexity is $O(nH_{out})$, where H_{out} is the time to create an outer signature with the hash function.

For example, the execution times of sharing 1 GB random 32-bits signed integers with bpVSS are plotted in Figures 5.1(a) and 5.1(b) with respect to t and n, respectively. User-defined functions for creating inner and outer signatures are $s\_in_{jkl} = \sum_{h=1}^{t-1} d\_s_{jklh} (\text{mod } p)$ and $s\_out_{ijkl} = e_{ijkl} \bmod 31$, respectively, where p is the base of transformed integers. The execution time is about 15 seconds (throughput is 68 MB/s) when $n = t = 3$. In Figure 5.1(a), the execution time quickly increases with t when $n = t$. In Figure 5.1(b), the execution time linearly increases with n when $t = 4$. t higher impacts the execution time than n because only t impacts the time to transform decimal integers into base-p integers and the time to create inner signatures. Note that p does not impact execution time. However, p is assigned with respect to t and the maximum value of decimal integers to achieve the lowest possible share volume (Section 5.4.1).



(a) W.r.t. t (b) W.r.t. n

FIGURE 5.1: Data sharing time with bpVSS

5.2.1.2 Data Reconstruction Process

To reconstruct an integer, a base-p integer of $t-1$ digits is reconstructed by multiplying the inverse t-square matrix of the linear equations' coefficients by the transposed vector of t shares. Then, it is transformed back into a decimal integer. Thus, the time complexity is $t^2 + t \approx O(t^2)$.

Moreover, inner and outer code verification are parts of the data reconstruction process. Inner code verification checks the secret against its inner signature at the user's. Its time complexity is $O(H_{in})$, where H_{in} is the time to create an inner signature with a user-defined homomorphic function. Outer code verification is done at each CSP's. It matches shares and outer signatures. Its time complexity is $O(H_{out})$, where H_{out} is the time to create an outer signature with a user-defined hash function. Since outer code verification is done in parallel at t CSPs', global time complexity is the same than at one CSP's.

For example, the execution time of reconstructing 1 GB of random 32-bits signed integers with bpVSS is plotted in Figure 5.2 with respect to t. User-defined functions for creating inner and outer signatures are defined as in Section 5.2.1.1. The execution time is about 10 seconds (throughput is about 102 MB/s) when $t = 3$. It increases in polynomial time. Note that p does not impact execution time.

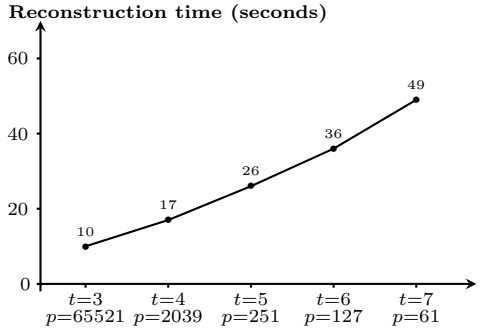


FIGURE 5.2: Reconstruction time with bpVSS

5.2.2 Security

5.2.2.1 Privacy

We focus here on data pilfering. Neither a CSP nor any intruder can decrypt a secret from only one share, and data transferred between the user and CSPs are all encrypted. In case an intruder can steal shares from $x \leq t$ CSPs, the probability of discovering the secret depends on the following.

1. The user-defined value of t. The higher t is, the lower the probability of breaking the secret is.

2. The size $\|p\|$ of control parameter p. The higher $\|p\|$ is, the lower the probability of breaking the secret is.

3. The number of pilfered shares x. The probability of breaking the secret obviously increases with x. However, bpVSS is secure enough, since it is difficult to retrieve shares from at least t CSPs by attacking them simultaneously.

The probability of discovering the secret, $1/p^{2t-x-1}$, is ploted in Figure 5.3.

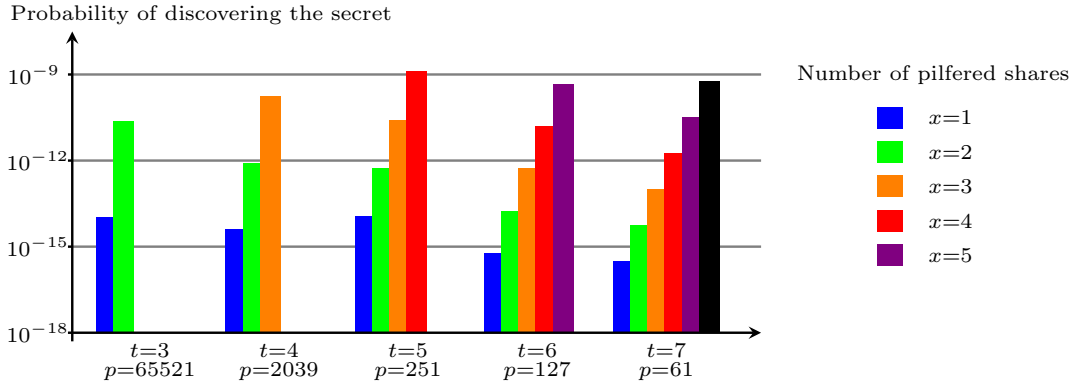


FIGURE 5.3: Probability of decrypting a data block from its shares

In bpVSS, although some secret data can be decrypted, if an intruder steals all shares from one CSP, s/he must discover the pattern of shares and generate all p^{t-1} combinations of shares stored at the $t-1$ other CSPs' by brute force. The complexity of bpVSS' reconstructing process is $O(\gamma t^2)$, since the $t \times t$ C matrix must be computed for γ secret data. Thus, with $p = 65521$, $t = 3$ and $\gamma = 100$ (11 KB of data), breaking the secret with the same machine as in Section 5.2.1 would take more than 21 years. Thence, even with a botnet available, even partially decrypting a giga or terabyte-scale DW cannot be achieved in reasonable time.

However, if an intruder exploits the meaning of shares with, e.g., banburismus, dictionary or frequency attack (with the help of a knowledge base), breaking time is lower than 21 years because bpVSS uses block encryption, and table names, attribute names and primary and foreign keys are unencrypted. Yet, discovering pattern of shares is still difficult because sharing data on one node uses a surjective function, i.e., two different secrets may have the same share value. Note that we envisage encrypting table names, attribute names and keys in future research, though.

5.2.2.2 Data Availability

Data availability is achieved by design with secret sharing. bpVSS guarantees that the user can reconstruct its secret if t or more CSPs are honest and their shares are accessible. Since the number $(n!/(t!(n-t)!))$ of possible groups of t CSPs increases with $n-t$, n must be greater than t to guarantee data availability.

5.2.2.3 Data Integrity

5.2.2.3.1 Efficiency of Inner and Outer Signatures

bpVSS verifies the honesty of CSPs and the correctness of shares by inner and outer code verification. Verification performance depends on the user-defined hash functions that define inner and outer signatures.

To test the reliability of inner and outer signatures, we generate 31 data pieces (a zero, 15 random negative integers and 15 random positive integers with different sizes) and share them. Then, we generate erroneous values in one share first, then in two shares, and so on up to t shares. Finally, we account for the number of incorrect data pieces that are not detected as such. Figure 5.4 plots the ratio of false positives achieved with inner signature $s\_in_{jkl} = \sum_{h=1}^{t-1} d\_s_{jklh} (\mathrm{mod} p)$ and outer signature $s\_out_{ijkl} = e_{ijkl} \bmod p_2$, where p and p_2 are primes. If only the inner signature is used to verify data, i.e., only the honesty of CSPs is verified, the ratio ranges between 4.03×10^{-4} and 7.68×10^{-1}, inversely depending on p. However, all incorrect data pieces can be detected if data are verified by both inner and outer signatures (i.e., share correctness is also verified) and $p_2 > 61$.

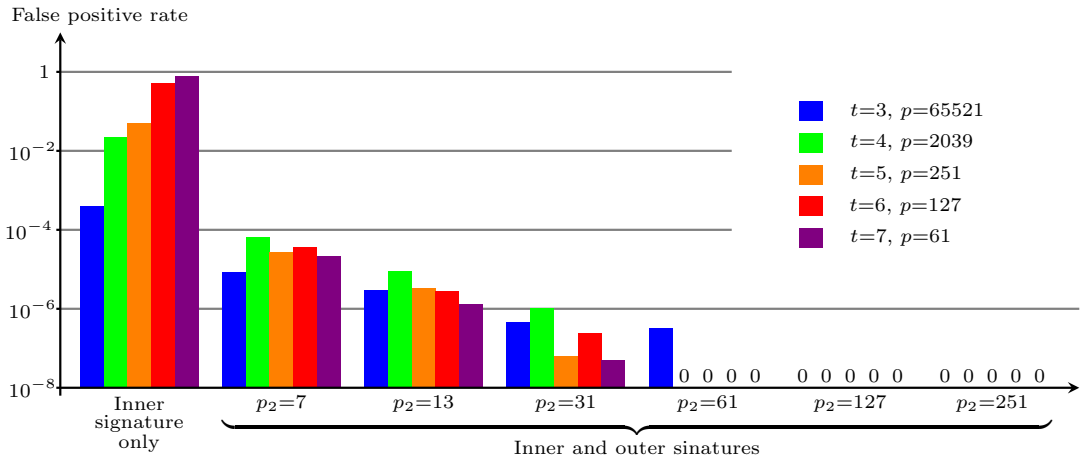


FIGURE 5.4: Rate of incorrect data not being detected

5.2.2.3.2 Correctness of Query Results

bpVSS allows exact match, SUM and AVG queries running on shares. The correctness of query results is proofed as follows.

For exact match queries, sharing data on one node uses a surjective function, i.e., two different initial values may have the same share. However, since the reconstruction process is achieved by intersecting all nodes, data sharing is overall a bijective function. Thus, querying shares always results in a 100% hit rate.

Thanks to the homomorphic property when adding t-variable linear equations, SUM queries can be computed by summing shares. Let us illustrate this through an example. Let γ values of attribute A_{jl} in table T_j be decimal integers $d_{j1l}, d_{j2l}, \cdots, d_{j\gamma l}$. They are rewritten as base-p integers $d_{jkl} = d\_s_{jkl1} + d\_s_{jkl2} \times p + \cdots + d\_s_{jkl\gamma} \times p^{(t-2)}$ (Figure 5.5), where d_s_{jklh} is the h^{th} base-p digit. Thanks to addition and multiplication's distributivity, the sum D of γ decimal integers can proceed by Equation 5.1, where D_S_h is the sum of the h^{th} digits of γ base-p integers ($D\_S_h = d\_s_{j1lh} + d\_s_{j2lh} + \cdots + d\_s_{j\gamma lh}$). Moreover, $t-1$ base-p digits $D\_S_1, D\_S_2, \cdots, D\_S_{t-1}$ and inner signature S_IN can be reconstructed by multiplying the inverse t-square matrix of the coefficients of t-variable functions f_i and the transposed vector of the sum of shares $E_i = e_{ij1l} + e_{ij2l}, + \cdots + e_{ij\gamma l}$, because functions f_i support the homomorphic property for addition (Figure 5.6). Since an inner signature is also created by an homomorphic function, inner signature S_IN can verify the correctness of summing decimal integer D.

$$D = d_{j1l} + d_{j2l}, + \cdots + d_{j\gamma l} = D\_S_1 + D\_S_2 \times p + \cdots + D\_S_{t-1} \times p^{t-2} \tag{5.1}$$

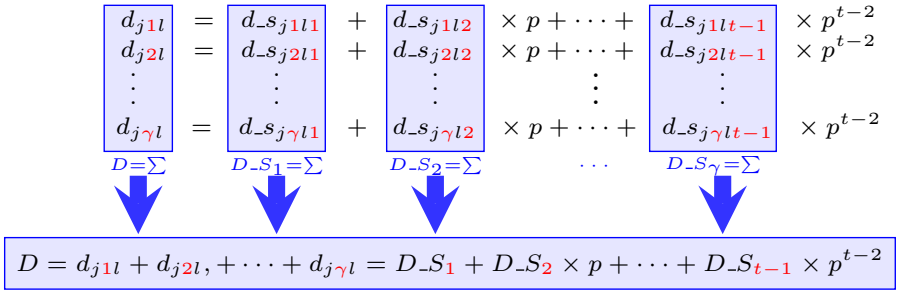


FIGURE 5.5: Addition and multiplication's distributivity of the summing D of γ decimal integers

For AVG queries, thanks to the homomorphic property (for addition and division by a constant value) of t-variable linear equations, the average of base-p integers can be computed from t couples of t-variable functions and average values of shares. Then, the average of a base-p integer is the conversion of the average of a decimal integer.

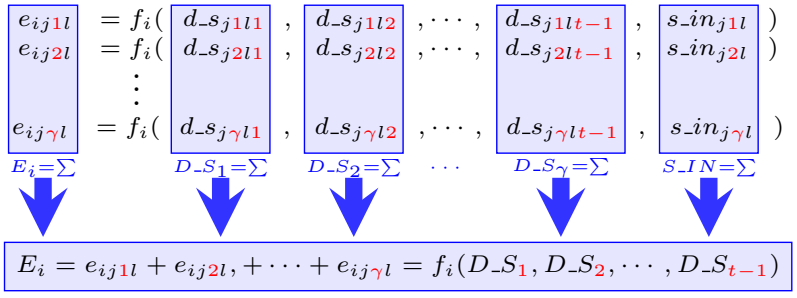


FIGURE 5.6: Illustration of homomorphism when summing t-variable functions f_i

5.2.3 Monetary Cost

Since bpVSS aims to operate in the cloud, we theoretically and experimentally study the factors that influence cost in the pay-as-you-go paradigm, i.e., computing, storage and data transfer costs.

5.2.3.1 Storage Cost

One advantage of our approaches is the low volume of shares with respect to existing SSSs. In bpVSS, the volume of an individual share is lower than that of the secret. Thus, global share volume is lower than n times the original data volume.

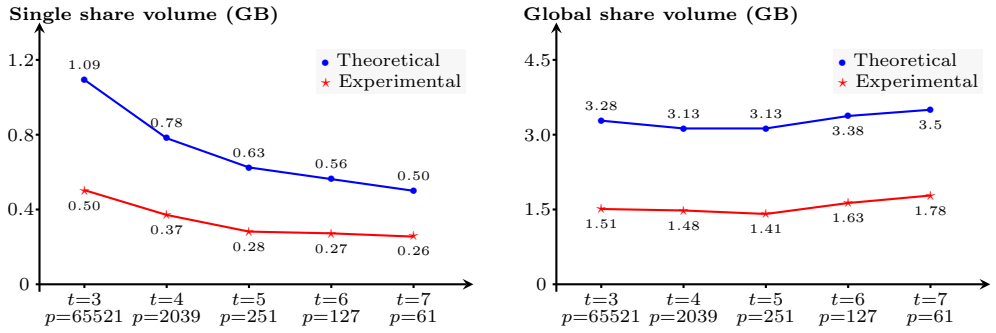
5.2.3.1.1 Share Volume

bpVSS transforms a decimal integer ranging in $]-p^{t-1}, p^{t-1}[$ into a base-p integer. Then, all $t-1$ digits ranging in $]-p, p[$ of the base-p integer and an inner signature are encrypted into each share with a random t-variable linear equation, such that coefficient values range in $[0, p[$. Hence, the value of a share lies between $\sum_{i=1}^{t}(p \times -p) = -tp^2$ and $\sum_{i=1}^{t}(p \times p) = tp^2$. Thus, the maximum volume of shares is $\max(\|e\|) = \|t\| + 2 \times \|p\| + 1 = \log_2 t + 2\log_2 p + 1$ bits, where $\|t\|$ and $\|p\|$ are the volumes of t and p, respectively.

Since the value of a decimal (secret) integer ranges in $]-p^{t-1}, p^{t-1}[$, its maximum volume is $\max(\|d\|) = (t-1)\|p\| = \lceil (t-1)\log_2 p \rceil$ bits. Ratios of single share and global share volumes by the secret's volume are $\frac{\max(\|e\|)}{\max(\|d\|)} = \frac{\lceil \log_2 t + 2\log_2 p + 1 \rceil}{(t-1)\log_2 p}$ and $\frac{n \times \max(\|e\|)}{\max(\|d\|)} = \frac{n\lceil \log_2 t + 2\log_2 p + 1 \rceil}{(t-1)\log_2 p}$, respectively.

For example, shares of a 32-bit signed integer (-2,147,483,648 to 2,147,483,648) value between $-4 \times 2,039^2 = -16,630,084$ and $3 \times 2,053^2 = 16,630,084$ with $t = 4$, $n = 5$ and $p = 2,039$. Then, the maximum size of shares is $\lceil \log_2 4 + 2\log_2 2039 + 1 \rceil = 25$ bits. Ratios of one share and global share volumes by the secret's volume are 25 bits/32 bits = 0.7813 and 5×25 bits/32 bits=3.9065, respectively.

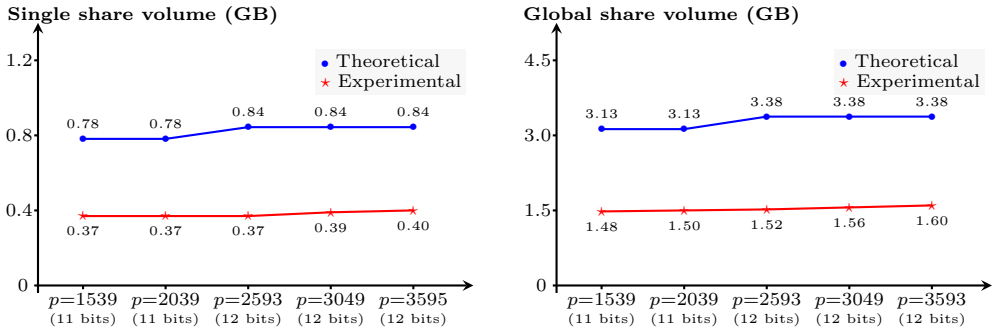
Figures 5.7, 5.8 and 5.9 plots the theoretical (blue lines) and experimental (red lines) volumes of shared data constructed from 1 GB of random 32-bit signed integers. Figures over/below lines are ratios of share volume by original data volume. Experimental volumes are lower than theoretical volumes (that are worst-case estimated volumes). In Figure 5.7(a), when $n = t$ and p is assigned with to respect t and the maximum value of decimal integers, the higher t is, the lower single share volume is. In contrast, in Figure 5.7(b), global share volume linearly increases with t when $n = t$. Note that theoretical and experimental global share volume when $t = 4$ and $t = 5$ are lower than when $t = 3$ because volume increases with t but decreases with the size of p.



(a) Single share volume (b) Global share volume

FIGURE 5.7: Shared data volume with bpVSS when $n = t$

In Figure 5.8, p does not impact theoretical volumes, but $\|p\|$ does. However, experimentally measured volumes slowly increase with p when $n = t = 4$, because of the impact of transformation between decimal and base-p integers.

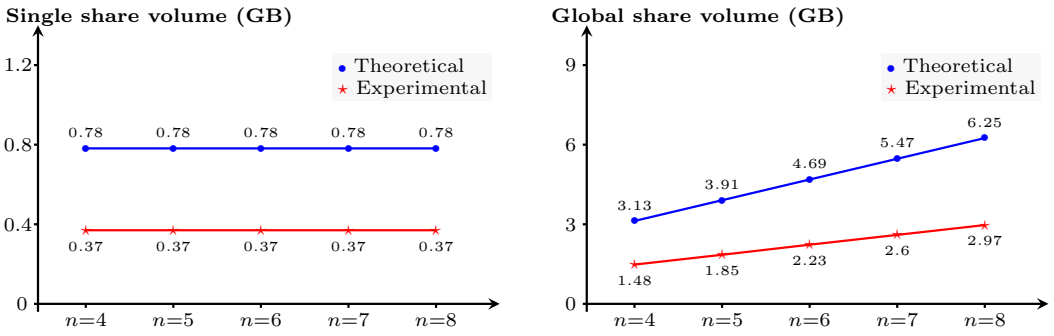


(a) Single share volume (b) Global share volume

FIGURE 5.8: Shared data volume with bpVSS when $n = t = 4$

In Figure 5.9(a), n, i.e., the number of CSPs, does not impact single share volume when t and p are fixed. However, in Figure 5.9(b), global share volume (n times single share volume) naturally increases with n.

To share table T_j, each attribute value is independently encrypted. However, primary keys are not encrypted. They are duplicated at all CSPs'. Thus, at CSP_i, the



(a) Single share volume (b) Global share volume

FIGURE 5.9: Shared data volume with bpVSS when $t = 4$ and p=2039

volume of shared record ER_{ijk} is the sum of its primary key size $\|pk_{ijk}\|$ and the sizes $\{\|e_{ijk1}\|, \cdots, \|e_{ijkq_j}\|\})$ of all attribute shares, i.e., $\|pk_{ijk}\| + \sum_{l=1}^{q_j} \|e_{ijkl}\|$. Since the volume of an attribute share is lower than that of its original volume, the volume of a shared record is lower than that of the original record. Similarly, the volume of a shared table is lower than that of the original table. To share any table at n CSPs, bpVSS guarantees that the volume of all shared tables is lower than n times the volume of original tables (cf. experiment in Section 6.3.1) if $t > 3$.

Note that a cube is shared in a relational table. Thus, global share volume of a cloud cube is lower than n times that of the original cube because the size of aggregate shares is lower than that of original aggregates.

5.2.3.1.2 Signature Volume

Inner signatures are data signatures. They are encrypted with data and hidden in shares. Outer signature are share signatures. They are stored in extra attributes in shared tables. Since one outer signature is created for one share with a user-defined hash function, the volume of signatures in shared table ET_{ij} is $r_j \times q_l \times \|s\|$, where $\|s\|$ is the size of a signature and r_j and q_l are the numbers of records and attributes of the shared table, respectively.

5.2.3.1.3 Monetary Cost

Storage cost depends on storage volume and CSP pricing policy. It is estimated by Equation 5.2, where V_i is data volume shared at CSP_i and $C_{s,i}$ is CSP_i's unitary storage price.

$$C_{s,all} = \sum_{i=1}^{n} (V_i \times C_{s,i}) \tag{5.2}$$

Table 5.2 shows the theoretical storage cost of sharing 1 GB of 32-bit integers at $n = 5$ CSPs, with $t = 4$ and $p = 2039$. CSP pricing policies for this range of data volume are depicted in Table 5.1. They are inspired from real prices from CSPs such as Amazon web services, Windows azure and Google compute engine. sVM, mVM and lVM stand for small, medium and large virtual machine (VM), respectively.

TABLE 5.1: CSP pricing policies

	CSP_1	CSP_2	CSP_3	CSP_4	CSP_5
Storage ($/GB/month)	0.030	0.040	0.053	0.120	0.325
Data transfer ($/GB)	0.023	0.053	0.090	0.111	0.120
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 5.2: Storage cost

Approach	**Share volume (GB)**						**Storage cost ($)**
	CSP_1	CSP_2	CSP_3	CSP_4	CSP_5	**Global**	
Unencrypted data	1					1	0.0300 to 0.3250
Shamir's [23]	1	1	1	1	1	5	0.5680
bpVSS	0.7813	0.7813	0.7813	0.7813	0.7813	3.9065	0.4438

If signatures are not stored at CSPs', theoretical storage volume for 1 GB of 32-bit integer (x data pieces) at each CSP is $V = \frac{(x)\lceil \log_2 t+2\log_2 p+1\rceil}{(x)(t-1)\log_2 px} = \frac{(x)\lceil \log_2 4+2\log_2 2039+1\rceil}{(x)(4-1)\log_2 2039} = 0.7813$, with $n = 5$, $t = 4$ and $p = 2039$. Storage cost is $C_{s,all} = (0.7813 \times 0.03) + (0.7813 \times 0.04) + (0.7813 \times 0.053) + (0.7813 \times 0.120) + (0.7813 \times 0.325) = \$0.4438$. This cost is lower than [23]'s storage cost ($0.5680) but higher than the maximum ($0.3250) storage cost of unencrypted data. Note that storage cost of unencrypted data varies with CSP pricing policy, i.e., the maximum cost is the cost for storing shares at the CSP's that has the most expensive unitary storage cost.

5.2.3.2 Computing Cost

5.2.3.2.1 Data Sharing

The monetary cost of sharing data depends on loading time, i.e., the time to load new shared records. At each CSP's, shared records are inserted into shared tables. Loading time depends on three factors: the size of shared records, the number of new records and computing power.

To estimate bpVSS' loading time, we compare it with loading unencrypted data. Shared record size without outer signature is lower than that of an unencrypted record, more precisely $1/(t-1)$ times the size of an unencrypted records (Section 5.2.3.1.1). Hence, bpVSS' loading time is lower than the loading time of unencrypted records if

they ran on the same VM (Section 6.3.2). However, since signatures are stored in shared records, actual loading time is greater.

5.2.3.2.2 Data Access

Data access cost directly depends on execution time for running queries and verifying matched shares. Although SUM, AVG and exact matched queries can run on shares in bpVSS, their total execution time can still be higher than on unencrypted data or using other SSSs, because only bpVSS verifies shares at data access time. Hence, bpVSS exhibits the greatest data access cost (Section 6.3.3.1).

5.2.3.3 Data Transfer

The monetary cost of data transfer directly relates to the volume of query results. When SUM and AVG queries are run, the global volume of bpVSS' query results is lower than t times the volume of unencrypted query results, because the size of a share is lower than that of an unencrypted data piece. However, the global volume of bpVSS' exact match query results may be greater than t times that of unencrypted data, because not only true but also false positive query results are transferred back.

5.3 fVSS

5.3.1 Complexity

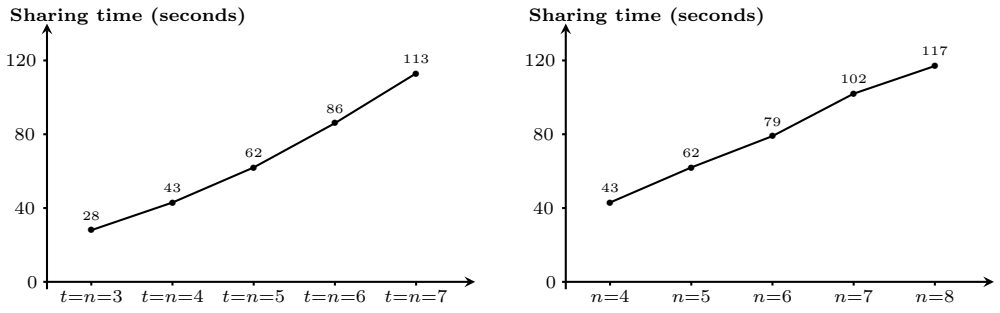
This section describes the time complexities of the data sharing and reconstruction processes and the factors that influence execution time. In addition to theoretical consideration, we run 100 1 GB test cases made of random 32-bit signed integers and vary parameter n and t. Experiments are conducted with Bloodshed Dev-C++ 5.5.3 on the same machine as in Section 5.2.1.

5.3.1.1 Data Sharing Process

To share an integer, a polynomial equation of degree $t-1$ is constructed by Lagrange interpolation, and then the constructed polynomial is computed $(n-t+2)$ times to create $n-t-2$ shares. Thus, the time complexity is $t^2 + t(n-t+2) = t^2 + nt - t^2 + 2t = nt + 2t \approx O(nt)$.

Moreover, inner and outer signatures are created in the data sharing process at the user's side and at CSPs', respectively. Since inner signatures are created by a user-defined homomorphic function, the time complexity is $O(H_{in})$, where H_{in} is the time to create an inner signature with the homomorphic function. Outer signatures are created and/or updated in the signature tree from leaf nodes to the root when new records are shared. Since the depth of the outer signature tree is $\lceil log_{w_i} m \rceil + \lceil log_{w_i} er'_{ij} + er_{ij} \rceil + 1$, update time complexity when er_{ij} shared record are loaded into table ET_{ij} is $O((\lceil log_{w_i} m \rceil + \lceil log_{w_i} er'_{ij} + er_{ij} \rceil) er_{ij} H_{out})$, where m is the number of shared tables, er'_{ij} is the total number of shared records in table ET_{ij} and H_{out} is the time to create an outer signature with the incremental function.

For example, the execution times of sharing 1 GB 32-bits signed integers with bpVSS are plotted in Figures 5.10(a) and 5.10(b) with respect to t and n, respectively. Inner signatures are created with function $s\_in_{jkl} = \sum_{h=1}^{t-1} d\_s_{jklh} (\mathrm{mod}\, 65647)$ such that the size of signatures is a half the size of data (Section 5.3.2.3.1). The execution time is about 28 seconds (throughput is 36 MB/s) when $n = t = 3$. In Figure 5.10(a), the execution time increases linearly with t when $n = t$. In Figure 5.10(b), the execution time linearly increases with n ($t = 4$). t has a higher impact on execution time than n because only t impacts the time to construct polynomial equations of degree $t - 1$ by Lagrange interpolation.



(a) W.r.t. t (b) W.r.t. n

FIGURE 5.10: Data sharing time with fVSS

5.3.1.2 Data Reconstruction Process

To reconstruct an integer, a polynomial equation of degree t is constructed by Lagrange interpolation, and then the constructed polynomial is computed two times to reconstruct an integer and an inner signature. Thus, the time complexity is $t^2 + 2t \approx O(t^2)$.

Moreover, inner and outer code verification are parts of the data reconstruction process. Inner code verification checks the secret against its inner signature at the user's.

The time complexity is $O(H_{in})$, where H_{in} is the time to create an inner signature with a user-defined homomorphic function. Outer code verification is done on-demand at each CSP's. Shared records, groups of shared records, shared tables, groups of shared tables and shared DWs are verified from only one node in the signature tree. Thus, the time complexity is $O(H'_{out})$, where H'_{out} is the time to create an outer signature with an incremental function.

For example, the execution time for reconstructing 1 GB of 32-bits signed integers with fVSS is plotted in Figure 5.11 with respect to t. Inner signatures are created as in Section 5.3.1.1. The execution time is about 26 seconds (throughput is 40 MB/s) when $t = 3$. It grows with polynomial time.

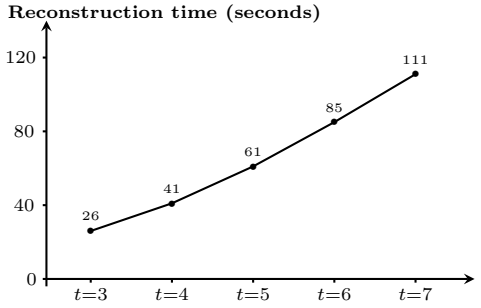


FIGURE 5.11: Reconstruction time with fVSS

5.3.2 Security

5.3.2.1 Privacy

As in bpVSS, data pilfering is the focus of fVSS. Neither a CSP nor any intruder can decrypt the secret from only one share, and data transferred between the user and CSPs are all encrypted. Moreover, fVSS does not use block encryption. It creates unpatterned shares by semi-random polynomials [23]. Hence, its security level is equal to [23]'s, i.e., any breaking approach (i.e., brute force, banburismus, dictionary or frequency attack) cannot access data from fewer than t shares. Thus, the higher t is, the higher privacy is.

Although a coalition or the compromission of at least t CSPs is necessary to break the secret in other SSSs, privacy is further improved in fVSS, because data is not shared at all CSPs', but only at $n - t + 2$ CSPs. Thence, fVSS imposes a new constraint: no CSP *group* can hold enough shares to reconstruct the secret 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.

5.3.2.2 Data Availability

As other SSSs, fVSS guarantees data access even though $n-t$ CSPs fail. Moreover, erroneous shares can be recovered from t other available shares. Unlike all other SSSs, fVSS allows updating shares in case of CSP failure, simply by not selecting the failing CSPs for sharing new data. This is possible because a secret is shared at $n-t+2$ CSPs' instead of n.

5.3.2.3 Data Integrity

5.3.2.3.1 Efficiency of Inner and Outer Signatures

As bpVSS, fVSS verifies the honesty of CSPs and the correctness of shares by inner and outer code verification. Verification performance depends on the functions that define inner and outer signatures. Moreover, verification performance of outer code verification also depends on the maximum number of nodes w_i in the outer signature tree, the number of verified signatures and the diversity of hash functions.

To test the reliability of inner signatures, we generate 31 data (a zero, 15 random negative integers and 15 random positive integers with different size in bits) and share them. Then, we generate erroneous values in one share first, then in two shares, and so on up to t shares. Finally, we account for the number of incorrect data pieces that are not detected as such. Figure 5.12 plots the ratio of false positives achieved with inner signature $s\_in_{jkl} = d_{jkl} \bmod p$, where p is a prime. The ratio varies between 0 and 7.69×10^{-4}, inversely depending on p (p controls the volume of inner signatures). However, all incorrect data pieces can be detected if $p > 32749$ or the size of inner signatures is greater than half the size of one data piece.

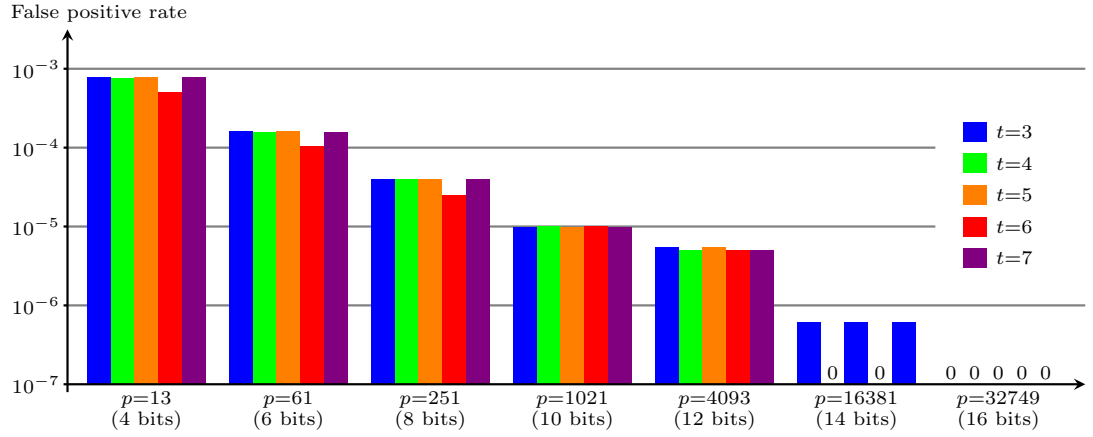


FIGURE 5.12: Rate of incorrect data not being detected by inner code verification

Since share correctness can be verified on-demand, the reliability of outer signatures is tested separately from that of inner signatures. We generate a random shared table with 10^5 records and three attributes (3×10^5 32-bits reals). A 100-ary signature tree is created from them. Then, we generate erroneous values in one, two, etc., and eventually in all shared records. Finally, we account for the number of incorrect cases that are not detected as such. Figure 5.13(a) plots the ratio of false positives achieved with the table signature stored at the root node. Table signature is $s\_out_{jkl} = \sum e_{ijkl} \bmod p$, where p is a prime. The ratio varies between 3.05×10^{-5} and 7.69×10^{-2}, inversely depending on p.

Figure 5.13(b) plots the ratio of false positives achieved with outer signatures stored at all nodes of the signature tree. Since there are only one shared table with 10^5 records, the depth of the 100-ary signature tree is $\lceil log_{w_i} m \rceil + \lceil log_{w_i} er_{ij} \rceil + 1 = \lceil log_{100} 1 \rceil + \lceil log_{100} 10^5 \rceil + 1 = 4$. A node stored at the l^{th} level has $100^{(4-l)}$ leaves, e.g., a node stored at the 2^{nd} level has $100^{(4-2)} = 100^2$ leaves (the number of shared records). Signatures stored at one node level are created with the same function. Signatures from different levels use different functions. Functions are defined with respect to the size of signatures and the number of shared records.

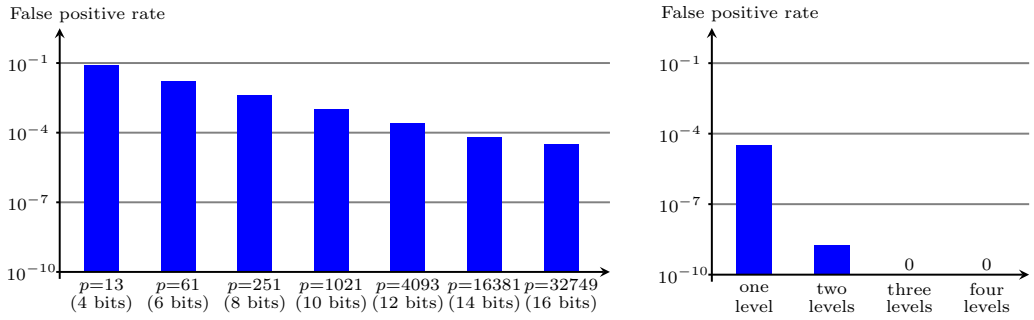
1. Table signature (1^{st} level) is $s\_out_{jkl} = \sum \lceil e_{ijkl} \rceil \bmod 32749$.
2. Signatures of groups of 100^2 shared records (2^{nd} level) are $s\_out_{jkl} = \sum \lfloor e_{ijkl} \rfloor \bmod 16381$.
3. Signatures of groups of 100 shared records (3^{rd} level) are $s\_out_{jkl} = \sum \lceil e_{ijkl} \rceil \bmod 4093$.
4. Signatures of shared record (4^{th} level) are $s\_out_{jkl} = \sum \lfloor e_{ijkl} \rfloor \bmod 1021$.

False positive ratios are 3.05×10^{-5}, 1.86×10^{-9}, 0 and 0 when signatures are verified on one level (1^{st} level), two levels (1^{st} and 2^{nd} levels), three levels (1^{st} to 3^{rd} levels) and four levels (1^{st} to 4^{th} levels), respectively. All incorrect shares can be detected if signatures are verified at at least three levels.

5.3.2.3.2 Correctness of Query Results

fVSS allows exact match, range and aggregation (SUM, AVG, COUNT, MAX, MIN, MEDIAN, MODE, STDDEV and VARIANCE) queries running on shares with the help of indices. The correctness of query results is proofed as follows.

For exact match, range and aggregation (COUNT, MAX, MIN, MEDIAN and MODE) queries, the correctness of query results is guaranteed by Type II indices (B+



(a) one-level verification (b) multi-level verification

FIGURE 5.13: Rate of incorrect shares not being detected by outer code verification

trees), that store primary keys of matched, ranged and aggregate results. Then, only matching shares are reconstructed.

For SUM queries, thanks to the homomorphic property (for addition) of the polynomial function, the sum of γ numbers (integers and reals) can be reconstructed from t sums of γ shares and pseudo shares (created from γ distinct polynomials). Summing pseudo shares is achieved from CSP identifiers and values of primary keys, because pseudo shares must be computed with a two-variables one-way function from CSP identifiers and primary keys, and the function is homomorphic for addition: $HE_2(pk_1, ID_i) + HE_2(pk_2, ID_i) + \cdots + HE_2(pk_\gamma, ID_i) = HE_2(pk_1 + pk_2 + \cdots + pk_\gamma, ID_i)$. Since an inner signature is created by an homomorphic function for addition, summing inner signatures help verify the correctness of sums of number.

AVG, STDDEV and VARIANCE aggregates must be computed by an external program from the results of the above aggregation queries, after reconstruction. For instance, AVG(X) is computed from SUM(X) and COUNT(X). Final aggregates are correct because SUM and COUNT queries return correct results.

5.3.3 Monetary Cost

Since fVSS allows users to adjust share volume at each CSP's, we theoretically study two strategies of adjusting share volume. Let $n = 5$ and $t = 4$. In the first strategy named fVSS-I, share volume is the same at all CSPs', as in all SSSs (balanced share volume). Moreover, lVM (Table 5.1) is assigned to all CSPs. In the second strategy named fVSS-II, share volume at each CSP's is adjusted with respect to pricing policies (unbalanced share volume). We again use the prices from Table 5.1 to estimate monetary cost. Share volume assigned to CSP_1, CSP_2, CSP_3, CSP_4 and CSP_5 are 27%, 27%, 20%, 13% and 13% of the global share volume, respectively, so that cheaper CSPs are

more solicited. Moreover, VM sizes assigned to CSP_1, CSP_2, CSP_3, CSP_4 and CSP_5 are lVM, lVM, mVM, sVM and sVM, respectively.

5.3.3.1 Storage Cost

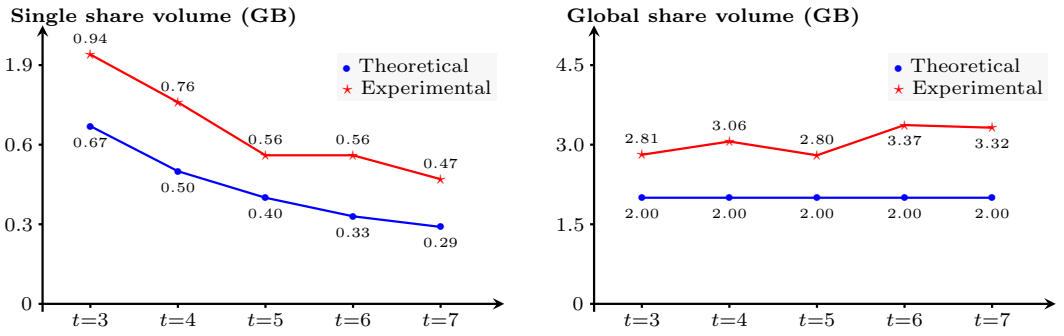
5.3.3.1.1 Share Volume

fVSS creates shares from a semi-random polynomial. Thus, share volume is not controlled. To estimate overall share volume, let us suppose the volume of a share is equal to that of its secret. Hence, global share volume is $n - t + 2$ times the secret's volume, because a secret is shared only $n - t + 2$ times.

In fVSS-I, share volume at each CSP's is $(n - t + 2)/n$ times the secret's volume. For example when $n = 5$ and $t = 4$, it is $(5-4+2)/5 = 0.6$ times the secret's volume. In fVSS-II, share volume at each CSP's ranges between 0 and 1 times the secret's volume. Let $n = 5$ and $t = 4$ again, we adjust share volumes at each CSP's as follows.

1. Share volume at CSP_1 is $0.27(n-t+2) = 0.27(5-4+2) = 0.81$ times the secret's volume.
2. Share volume at CSP_2 is $0.27(n-t+2) = 0.27(5-4+2) = 0.81$ times the secret's volume.
3. Share volume at CSP_3 is $0.20(n-t+2) = 0.20(5-4+2) = 0.60$ times the secret's volume.
4. Share volume at CSP_4 is $0.13(n-t+2) = 0.13(5-4+2) = 0.39$ times the secret's volume.
5. Share volume at CSP_5 is $0.13(n-t+2) = 0.13(5-4+2) = 0.39$ times the secret's volume.

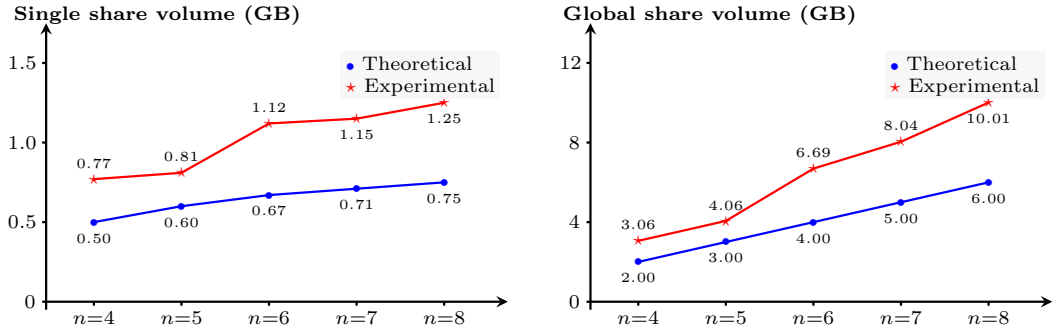
Figures 5.14 and 5.15 plot the theoretical (blue lines) and experimental (red lines) volumes of shared data constructed from 1 GB of random 32-bit random signed integers with fVSS-I. Figures over/below lines are ratios of share volume by original data volume. Experimental volumes are greater than theoretical volumes because the size of a share is equal to that of a secret in the theoretical assumption, but it is greater in reality. The size of a share is about 1.57 times that of the secret. In Figure 5.14(a), when $n = t$, the higher t is, the lower single share volume is. However, in Figure 5.14(b), t does not impact global share volume when $n = t$. Theoretical global share volume is twice the secret's volume. However, experimental global share volume varies between 2.8 and 3.37 times the secret's volume.



(a) Single share volume (b) Global share volume

FIGURE 5.14: Shared data volume with fVSS-I when $n = t$

In Figure 5.15, single share and global share volumes increase with n when $t = 4$. In the experiment, single share and global share volumes grow linearly.



(a) Single share volume (b) Global share volume

FIGURE 5.15: Shared data volume with fVSS-I when $t = 4$

To share table T_j with r_j records at n CSPs', each record is shared $n - t + 2$ times. Thus, the volume of all shared tables is also $n - t + 2$ times that of the original tables (cf. experiment in Section 6.3.1). In fVSS-I, the number er_{ij} of records in each shared table is $er_{ij} = (n - t + 2)r_j/n$. For example when $n = 5$ and $t = 4$, it is $er_{ij} = (n-t+2)r_j/n = (5-4+2)r_j/5 = 0.6r_j$ records. Thus, the volume of each shared table at any CSP is only $(n - t + 2)/n = (5 - 4 + 2)/5 = 0.6$ times that of the original table. In fVSS-II, the number er_{ij} of records in each shared table varies between 0 and the number r_j of original records ($0 \leq er_{ij} \leq r_j$). For example still with $n = 5$ and $t = 4$, let us adjust the numbers of records in shared tables at CSP_1, CSP_2, CSP_3, CSP_4 and CSP_5 to be $er_{ij} = 0.81r_j, 0.81r_j, 0.60r_j, 0.39r_j$ and $0.39r_j$, respectively. Thus, the volumes of each shared table at CSP_1, CSP_2, CSP_3, CSP_4 and CSP_5 are 0.81, 0.81, 0.60, 0.39 and 0.39 times that of the original table, respectively.

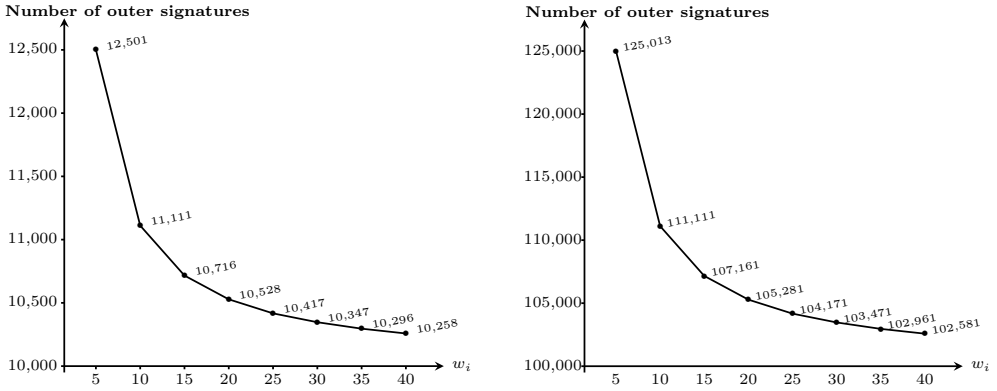
5.3.3.1.2 Signature Volume

As in bpVSS, inner signatures are encrypted with data and hidden in shares. However, outer signatures are stored separately from shared tables. They are stored in a w_i-ary tree at CSPs'. Each tree node stores the signature of a shared record, a group of shared records, a shared table, a group of shared tables or a shared DW. The size $\|s\|$ of signatures depends on user-defined incremental functions. The total number of signatures (nodes) for a shared table ET_{ij} (in the record-signature tree) and a shared DW (in the outer-signature tree) at CSP_i are defined in Equations 5.3 and 5.4, respectively, where er_{ij} is the number of records in shared table ET_{ij}, m is the number of shared tables and $\sum_{j=1}^{\lceil \log_{w_i} m \rceil} \lceil m/(w_i)^j \rceil$ is the total number of nodes in the table-signature tree. Thus, the volume of signatures for shared table ET_{ij} and a shared DW at CSP_i are $num_{(sig,ET_{ij})} \times \|s\|$ and $num_{(sig,DW_i)} \times \|s\|$, respectively.

$$num_{(sig,ET_{ij})} = er_{ij} + \sum_{l=1}^{\lceil \log_{w_i} er_{ij} \rceil} \lceil er_{ij}/(w_i)^l \rceil \tag{5.3}$$

$$num_{(sig,DW_i)} = \sum_{j=1}^{\lceil \log_{w_i} m \rceil} \lceil m/(w_i)^j \rceil + \sum_{j=1}^{m} \left(er_{ij} + \sum_{l=1}^{\lceil \log_{w_i} er_{ij} \rceil} \lceil er_{ij}/(w_i)^l \rceil \right) \tag{5.4}$$

Figure 5.16 plots the number of outer signatures at each CSP's, for 10 shared tables of 100,000 records. When w_i is high, the number of outer signatures decreases, and thus storage cost also decreases. The number of outer signatures sharply decreases when $w_i < 15$, and slowly decreases when $w_i > 15$.



(a) number of signatures for a shared table (b) number of signatures for a shared DW

FIGURE 5.16: Number of outer signatures in fVSS

5.3.3.1.3 Cloud Cube Volume

In fVSS, cloud cubes are managed as in bpVSS. However, outer signatures are separated from the cube. Each cube record is shared at all shared tables at n CSPs, whereas normal data records are shared in only $n-t+2$ shared tables. Thus, the volume of a cloud cube is n times that of the original cube.

5.3.3.1.4 Monetary Cost

Storage cost depends on storage volume and CSP pricing policies. It is estimated by Equation 5.5, where V_i is the volume of all shares and signatures of a DW and possible cloud cubes at CSP_i and $C_{s,i}$ is CSP_i's unitary storage price. Since CSPs storage prices are fixed, storage cost may be reduced by reducing global storage volume (Section 5.3.3.1.1) or unbalance share volume at each CSP with respect to pricing.

$$C_{s,all} = \sum_{i=1}^{n} (V_i \times C_{s,i}) \tag{5.5}$$

Let us illustrate how unbalancing share volume helps minimize monetary cost through an example. Table 5.3 shows the theoretical storage cost of sharing 1 GB of data at $n = 5$ CSPs, with $t = 4$. CSP storage prices are shown in Table 5.1. If signatures are not stored at CSPs', global share volume is $n - t + 2 = 5 - 4 + 2 = 3$ times the original data volume, or 3 GB (Section 5.3.3.1.1). In fVSS-I, share volume at each CSP is $(n - t + 2)/n = (5 - 4 + 2)/5 = 0.6$ times the original data volume, or 0.6 GB. Thus, storage cost is $C_{s,all} = (0.6 \times 0.030) + (0.6 \times 0.040) + (0.6 \times 0.053) + (0.6 \times 0.120) + (0.6 \times 0.325) = \$0.3408$. This cost is lower than [23]'s storage cost (\$0.5680) but greater than the maximum (\$0.3250) storage cost of unencrypted data. In fVSS-II, the share volumes at CSP_1, CSP_2, CSP_3, CSP_4 and CSP_5 are 0.81 GB, 0.81 GB, 0.60 GB, 0.39 GB and 0.39 GB, respectively. Storage cost is $(0.81 \times 0.03) + (0.81 \times 0.04) + (0.60 \times 0.053) + (0.39 \times 0.12) + (0.39 \times 0.325) = \$0.2621$. It is lower than fVSS-I's storage cost (\$0.3408) and lies between the costs to store unencrypted data at the cheapest CSP's (\$0.03) and the most expensive CSP's (\$0.325).

TABLE 5.3: Storage cost

Approach	**Share volume (GB)**						**Storage cost (\$)**
	CSP_1	CSP_2	CSP_3	CSP_4	CSP_5	**Global**	
Unencrypted data	1					1	0.0300 to 0.3250
Shamir [23]	1	1	1	1	1	5	0.5680
fVSS-I	0.60	0.60	0.60	0.60	0.60	3	0.3408
fVSS-II	0.81	0.81	0.60	0.39	0.39	3	0.2621

5.3.3.2 Computing Cost

Monetary cost for sharing data and accessing data depends on execution time and CSP pricing. It is estimated by Equation 5.6, where CT_i is the execution time at CSP_i and $C_{c,i}$ is CSP_i's computing cost per unit.

$$C_{c,all} = \sum_{i=1}^{n} (CT_i \times C_{c,i}) \tag{5.6}$$

5.3.3.2.1 Data Sharing

At CSP_i's, fVSS-I loading time CT_i is lower than that of loading unencrypted data and other approaches, because the number of shared records is lower. Thus, monetary computation cost bears the same behavior. Moreover, the benefit of unbalancing share volumes and VM sizes helps fVSS-II's computation cost being lower than that of fVSS-I's.

Let us illustrate this through an example. Table 5.4 shows the theoretical computing cost for sharing 10^{15} records at $n = 5$ CSPs', with $t = 4$. CSP pricing policies are depicted in Table 5.1. Let us assume that the size of a shared record is equal to the size of an unencrypted record, and that the computing powers of sVMs, mVMs and lVMs are 1×10^{10}, 2×10^{10} and 4×10^{10} records per second, respectively.

TABLE 5.4: Sharing cost comparison

Approach	**#records at each CSP's**	**VM type**	**Sharing time (h:mm)**	**CPU cost ($)**
Unencrypted data at 1 CSP's	10^{15}	lVM	6:57	0.36 to 1.94
Shamir [23]	10^{15}	lVM	6:57	6.40
fVSS-I	6×10^{14}	lVM	4:10	3.84
fVSS-II	8.1×10^{14}	lVM	5:38	3.23
	8.1×10^{14}	lVM	5:38	
	6.0×10^{14}	mVM	8:20	
	3.9×10^{14}	sVM	10:50	
	3.9×10^{14}	sVM	10:50	

To store $r_j = 10^{15}$ records, $(n-t+2)r_j = (5-4+2) \times 10^{15} = 3 \times 10^{15}$ new records are loaded at $n = 5$ CSPs', with $t = 4$ (Section 5.3.3.1.1). In fVSS-I, the number of new records at each CSP's is $er_{ij} = (n-t+2)r_j/n = (5-4+2) \times 10^{15}/5 = 6 \times 10^{14}$. Loading time on a lVM is $CT_i = 6 \times 10^{14}/(4 \times 10^{10}) = 1.5 \times 10^4$ s, i.e., 4h10min. Hence, computation cost is \$3.84. It is lower than [23]'s computing cost (\$6.40) but greater than the maximum (\$1.94) cost for loading unencrypted data.

With fVSS-II, the number of new records at CSP_1, CSP_2, CSP_3, CSP_4 and CSP_5 are 8.1×10^{14}, 8.1×10^{14}, 6.0×10^{14}, 3.9×10^{14} and 3.9×10^{14}, respectively. Loading times are estimated as follows.

1. At CSP_1, loading time on an lVM is
   $CT_1 = 8.1 \times 10^{14}/(4 \times 10^{10}) = 20,250$ s, i.e., 5h38min.

2. At CSP_2, loading time on an lVM is
   $CT_1 = 8.1 \times 10^{14}/(4 \times 10^{10}) = 20,250$ s, i.e., 5h38min.

3. At CSP_3, loading time on an mVM is
   $CT_3 = 6.0 \times 10^{14}/(2 \times 10^{10}) = 30,000$ s, i.e., 8h20min.

4. At CSP_4, loading time on an sVM is
   $CT_4 = 3.9 \times 10^{14}/(1 \times 10^{10}) = 39,000$ s, i.e., 10h50min.

5. At CSP_5, loading time on an sVM is
   $CT_4 = 3.9 \times 10^{14}/(1 \times 10^{10}) = 39,000$ s, i.e., 10h50min.

Hence, computation cost is $3.23 overall. It is lower than fVSS-I's computing cost ($3.84) but still greater than the maximum ($1.94) cost for loading unencrypted data.

5.3.3.2.2 Data Access

Data access cost directly depends on query response time CT_i, which depends on several factors. Let us suppose that query response time depends only on record volume, the number of matched records and computation power. Table 5.5 shows the theoretical data access cost when performing a query on a shared DW at $n = 5$ CSPs, with $t = 4$. Let us assume we run a query matching 10% of records, i.e., $\gamma = 10^{14}$, and $RG = \{CSP_1, CSP_2, CSP_4, CSP_5\}$. CSP pricing policies and VM power are the same as in Section 5.3.3.2.1.

TABLE 5.5: Data access cost comparison

Approach	**#records at each CSP's**	**VM type**	**Response time (h:mm)**	**CPU cost ($)**
Unencrypted data at 1 CSP's	10^{14}	lVM	0:42	0.04 to 0.19
Shamir [23]	10^{14}	lVM	0:42	0.48
fVSS-I	6×10^{13}	lVM	0:25	0.29
fVSS-II	8.1×10^{13}	lVM	0:34	0.26
	8.1×10^{13}	lVM	0:34	
	0	—	0:00	
	3.9×10^{13}	sVM	1:05	
	3.9×10^{13}	sVM	1:05	

With fVSS-I, query response time (25 minutes) is lower than on unencrypted data (42 minutes), since the number of fVSS-I's matched shared records is lower than that of matched unencrypted-data records. Hence, data access cost in fVSS-I ($0.29) is lower than [23]'s data access cost ($0.48) but greater than that in unencrypted data ($0.19). Moreover, unbalancing share volume makes data access cost in fVSS-II ($0.26) lower than in fVSS-I ($0.29).

5.3.3.3 Data Transfer Cost

Monetary cost of data transfer directly relates to the volume of query results. fVSS allows several aggregation (SUM, AVG, MAX, MIN, MEDIAN, MODE, COUNT, STDDEV and VARIANCE) queries running on shares. Moreover, Type-II indices also help exact match and range queries. Thus, only aggregate shares and indices are transferred to the user's. For SUM, global volume of fVSS' query results is about $2t$ times the volume of unencrypted query results (t times from t CSPs and t times from Type-I indices). For COUNT, only the results (one time the volume of unencrypted query results) discovered by Type-II indices are transferred to the user's. For MAX, MIN, MEDIAN and MODE, global volume of fVSS' query results is no greater than $n - t + 3$ times the volume of unencrypted query results (no greater than $n - t + 2$ times from t CSPs and 1 times from Type-II indices) because data are shared only $n - t + 2$ times. AVG, STDDEV and VARIANCE are computed from reconstructed results from the above operators. Thus, global data transfer volumes are the sum of the volumes output by the above operators. For instance, for AVG, global volume of fVSS' query results is about $2t + 1$ times the volume unencrypted query results ($2t$ for SUM and 1 for COUNT).

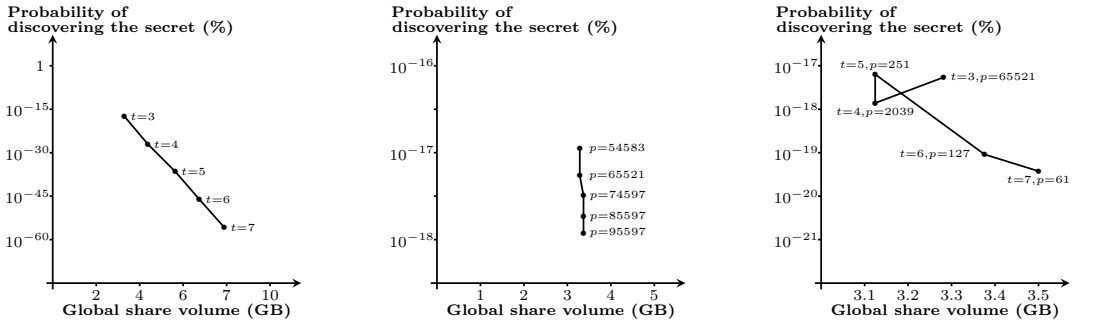
5.4 Discussion

5.4.1 Security vs. Performance

Security and performance of our SSSs depend on the values of parameters n and t and the sizes of inner and outer signatures $\|s_{in}\|$ and $\|s_{out}\|$. We discuss their impact and show an example of sharing 1 GB of 32-bit signed integers. Note that p is the base of transformed integers in bpVSS and its size $\|p\|$ is assigned with respect to t and the size of data (32 bits). Similarly, w_i is the maximum number of child nodes in the outer signature tree in fVSS. It is assigned with respect to CSP pricing policies.

Data privacy is the main security issue we focus on. To achieve higher privacy, t and $\|p\|$ (in bpVSS) should be big integers. When t is big, our SSSs produce small shares, which helps minimizing memory consumption and execution time when loading and accessing data. However, t cannot be too big, because the number of CSPs is limited in practice. Similarly, $\|p\|$ should not be greater than the maximum size of a data piece in bpVSS. Moreover, when t and $\|p\|$ are assigned to big integers, global share volume is very large, and thus data storage cost is high. In contrast, t does not impact fVSS' global share volume, which is fixed at twice the original data volume when t increase and $n = t$. However, when t is big, the efficiency of data sharing and reconstruction is negatively impacted.

With bpVSS, privacy must balance with storage cost. For example, Figures 5.17(a), 5.17(b), and 5.17(c) plot the probability of discovering the secret *vs.* global share volume in bpVSS, with respect to t, p and both t and p, respectively. The probability of breaking the secret (when one share is stolen) decreases, but global share volume increases with t and $\|p\|$. To achieve the highest security with the lowest possible storage cost (Figure 5.17(c)), t should be a big integer and $\|p\|$ should equal to $\|d_{max}\|/(t-1)$, where $\|d_{max}\|$ is the size of greatest secret data, because we can only share data in rang $[-p^{t-1}, p^{t-1}]$ and $\|d_{max}\| = \log_2 p^{t-1} = (t-1)\log_2 p = (t-1)\|p\| \Leftrightarrow \|p\| = \|d_{max}\|/(t-1)$.



(a) p=65521 ($\|p\|$ is 16 bits) (b) t=3 (c) p is assigned with respect to t and the size of data

FIGURE 5.17: Probability of discovering the secret in bpVSS

With fVSS, no tradoff between privacy and storage cost since global share volume is fixed at twice the original data volume.

Our SSSs guarantee data availability when $n > t$. However, global share volume increases with n when t is fixed. Moreover, sharing time at the user's also increases with n. Thus, to achieve data availability while minimizing global share volume and sharing time, n should be close to t. Moreover, n and t should meet the condition $n < 2 \times t - 2$ and $t > 2$ in fVSS to achieve higher privacy (no CSP group can break the secret) and guarantee new data can still be shared if some CSPs fail.

Data integrity in our SSSs is achieved with inner and outer signatures. The efficiency of inner and outer signatures is higher when their sizes are big. Since $\|s_{in}\| = \|p\|$ and $\|p\|$ relates to t in bpVSS, $\|s_{in}\|$ should meet the condition $\|s_{in}\| = \|p\| = \|d_{max}\|/(t-1)$ to achieve the best efficiency and the lowest possible storage cost. $\|s_{in}\|$ does not impact share volume in fVSS, but $\|s_{in}\|$ should be a big integer and greater than a half the size of data to detect all incorrect data pieces (Section 5.3.2.3.1).

Unlike inner signature volume, outer signature volume increases with $\|s_{out}\|$ when t and n are fixed. Hence, the efficiency and volume of outer signatures must be balanced. Since inner and outer signatures work together during data reconstruction with bpVSS, we plot the ratio of false positives achieved with inner and outer signatures *vs.* global

outer-signature volume when $t = n = 4$ and $p = 2039$ (11 bits) in Figure 5.18. To detect all incorrect data pieces, $\|s\|$ should be at least 6 bits. This is achieved when global signature volume is 0.75 GB (original data is 1 GB and the maximum global share volume is 3.1250 GB).

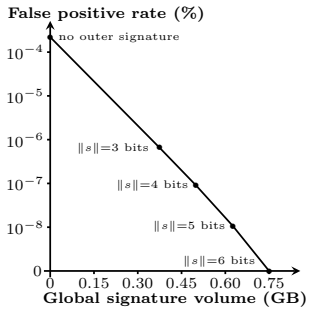


FIGURE 5.18: Ratio of false positives achieved with inner and outer signatures vs. global signature volume in bpVSS

Since outer signatures in fVSS are verified on-demand and separate from inner signature, Figure 5.19 plots the ratio of false positives achieved with only outer signatures *vs.* global outer-signature volume when $t = n = 4$. The efficiency and volume of outer signatures increase with $\|s_{out}\|$. Thus, $\|s_{out}\|$ should be a big integer. Since outer signatures can be verified on-demand on several levels (i.e., shared records, groups of shared records, shared tables, groups of shared tables, and shared DWs) of the w_i-ary signature tree, all incorrect shares can be detected if signatures are verified at at least three levels (Section 5.3.2.3.1). Moreover, w_i impacts signature volume. Thus, it should be a big integer to reduce signature volume, and thus storage cost.

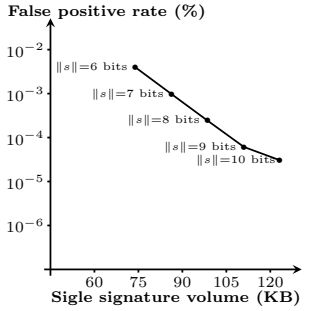


FIGURE 5.19: Ratio of false positives achieved with only outer signatures vs. global signature volume with fVSS

Cloud cubes improve query response time on shares. However, $n/(t-1)$ and n times the original cube volume are required to store a cloud cube. Since the number of records in a cloud cube is equal to the number of combinations of all dimension values, it can be huge. Hence, cloud cubes should store only the records that are frequently accessed to reduce storage volume, which turns down to a materialized view selection problem. Other records can be directly retrieved from the shared DW.

Finally, not only parameter setting, but also unbalancing share volume and VM size help reduce storage and computation monetary cost in fVSS. Thus, the biggest share volume should be stored at the cheapest CSP's. Moreover, the maximum size of VM should be assigned to CSPs that store the largest share volumes, to reduce the gap between the lowest and highest execution times at CSPs'. Since data sharing or access runs parallely on CSPs, total execution time is indeed the highest individual execution.

5.4.2 Comparison w.r.t. State of the Art SSSs

In this section, we compare our SSSs to the related approaches presented in Section 2.2, with respect to security, database features and complexity. Table 5.6 synthesizes the features and complexity of all approaches, which we discuss below.

5.4.2.1 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. However, a coalition or the compromise of at least t CSPs breaks the secret. Only fVSS guarantees no CSP *group* can hold enough shares to reconstruct the original data if $n < 2 \times t - 2$.

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, only fVSS allows updating shares in case at most $t - 2$ CSP fail, simply by sharing new data at other $n - t + 2$ available CSPs'.

To enforce integrity, only our SSSs verify both the correctness of shares and the honesty of CSPs, with the help of both outer and inner code verification, respectively. Only three other approaches [21, 58, 60] use inner code verification alone.

5.4.2.2 Database Features

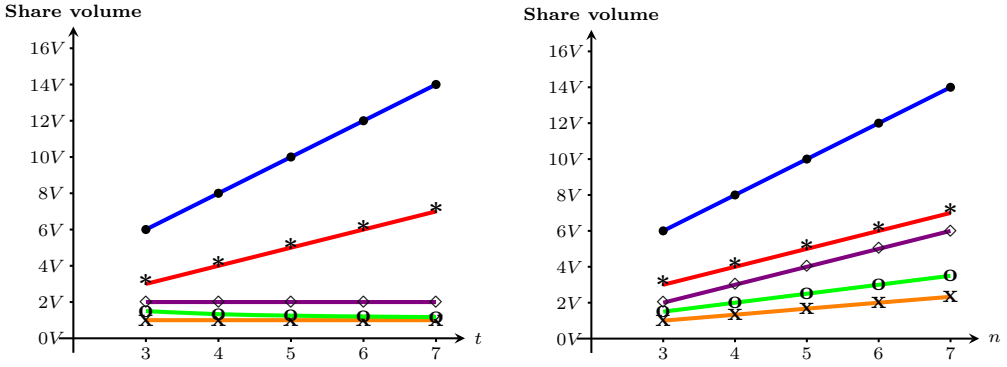
All secret sharing-based approaches focus on sharing (positive) integers on DBs or DWs. Only our SSSs and [56] further share nonnumerical data such as characters and strings. To access shared data, all approaches allow at least one query type (exact match, range, aggregate and grouping queries) on shares. Only SSSs allow all these queries. Moreover, only our SSSs allow straight computations on shares by creating shared data cubes.

TABLE 5.6: Comparison of database sharing approaches

Features and costs	**[56]**	**[54]**	**[55]**	**[57]**	**[59]**	**[60]**	**[58]**	**[21]**	**bpVSS**	**fVSS**
Data privacy	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Data availability	Yes	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	Yes
- Update shares	No	No	No	No	No	No	No	No	Yes	Yes
Data integrity										
- Inner code verifying	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes
- Outer code verifying	No	No	No	No	No	No	No	No	Yes	Yes
Target	DBs	DWs	DBs	DBs	DBs	DBs	DBs	DBs	DWs	DWs
Data sources	Single	Multi	Multi	Single	Single	Single	Single	Single	Single	Single
Data types	Positive integers, Characters, Strings	Positive integers	Integers	Integers	Positive integers	Positive integers	Positive integers	Positive integers	Integers, Reals, Characters, Strings, Dates, Booleans	Integers, Reals, Characters, Strings, Dates, Booleans
Shared data access										
- Updates	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
- Exact match queries	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
- Range queries	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
- Aggregation queries on one attribute	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
- Aggregation queries on two attributes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
- Grouping queries	No	No	No	No	No	No	No	No	Yes	Yes
Complexity										
- Data storage w.r.t. original data volume	$\geq n$	$\geq 2n$	$\geq 2n$	$\geq n$	$\geq n$ +1 (B++ tree)	$\geq n$ +1 (B++ tree)	$\geq 2n$ +1 (hash tree)	$\geq n/t$ $+n/t$ (B++ tree) + signatures	$\geq n/(t-1)$ + signatures	$\geq n-t+2$ +1 (B++ tree) + signatures
- Sharing time	$O(nt)$	$O(nt)$	$O(nt)$	$O(nt)$	$O(nt)$	$O(nt)$	$O(nt)$	$O(n)$	$O(nt)$	$O(nt)$
- Reconstruction time	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t^2)$	$O(t)$	$O(t^2)$	$O(t^2)$

5.4.2.3 Complexity

Storage volume (Table 5.6) of all approaches depends on n and t. Figures 5.20(a) and 5.20(b) plot the volume of shared data for all studied approaches, expressed as a multiple of original data volume V, with respect to t when $n = t$ and n when $t = 3$, respectively. Figure 5.20 shows that our SSSs help control shared data volume better than most existing approaches, and is close to the best approach [21] in this respect. However, storage cost depends not only on the global volume of shares, but also on CSP pricing policies. Only fVSS allows selecting the data volume shared at each CSP's, which can be differentiated to benefit from different pricing policies.



(a) $n = t$ (b) $t = 3$

• *[54, 55, 58]* ∗ *[56, 57, 59, 60]* x *[21]* o *bpVSS* ◇ *fVSS*

FIGURE 5.20: Storage complexity comparison

All secret sharing approaches bear the same sharing and reconstruction time complexity, except [21], which bears the lowest time complexity because it encrypts many records at once. Thus, we expect the execution time of our SSSs to be the same as other approaches. This is experimentally verified in the next chapter.

Chapter 6

Comparative Study

6.1 Introduction

In this chapter, we experimentally compare our SSSs with two state of the art algorithms (Thompson et al.'s [58] and Hadavi et al.'s [60] approaches; Section 2.2, which we label Thompson and Hadavi in the remainder of this chapter, for the sake of brevity) that address all three security issues: data privacy, availability and integrity, and also allow aggregation queries. The performance of all approaches is measured with respect to share volume, data sharing/reconstruction time, workload response time and transferred data volume. Finally, we discuss our SSSs' performance and compare monetary costs with these existing, related approaches.

6.2 Experimental Setup

6.2.1 Hardware and Software

Our experiments are conducted with PHP 5.2.6 on a PC with an Intel(R) Core(TM) i5 2.76 GHz processor with 3 GB of RAM on Microsoft Windows 7 as the machine at the user's side. Moreover, we simulate three VMs on three physical machines.

1. The first VM (VM_1) is simulated on a laptop with an Intel(R) Core(TM) i5 1.80 GHz processor with 4 GB of RAM with Microsoft Windows 8, PHP 5.2.6 and MySQL 5.0.51b.
2. The second VM (VM_2) is simulated on a laptop with an Intel(R) Core(TM) i5 2.50 GHz processor with 6 GB of RAM with Microsoft Windows 8, PHP 5.2.6 and MySQL 5.0.51b.

3. The third VM (VM_3) is simulated on a PC with an Intel(R) Core(TM) i5 2.76 GHz processor with 3 GB of RAM with Microsoft Windows 7, PHP 5.2.6 and MySQL 5.0.51b.

6.2.2 Benchmark

In our experiments, we use the Star Schema Benchmark (SSB), which is designed to measure the performance of database products in support of classical DW applications [94]. SSB remodels the TPC-H [95] benchmark's DB as a star schema. We selected SSB because it is much simpler than the new standard TPC-DS [96], and thus simpler to implement in a distributed environment, while still being relevant for our experiments. Figure 6.1 shows SSB's DW schema. SSB's DB composes of one fact table (LINEORDER) and four dimension tables (CUSTOMER, SUPPLIER, PART and DATE). Its global size is 757 MB (Table 6.1).

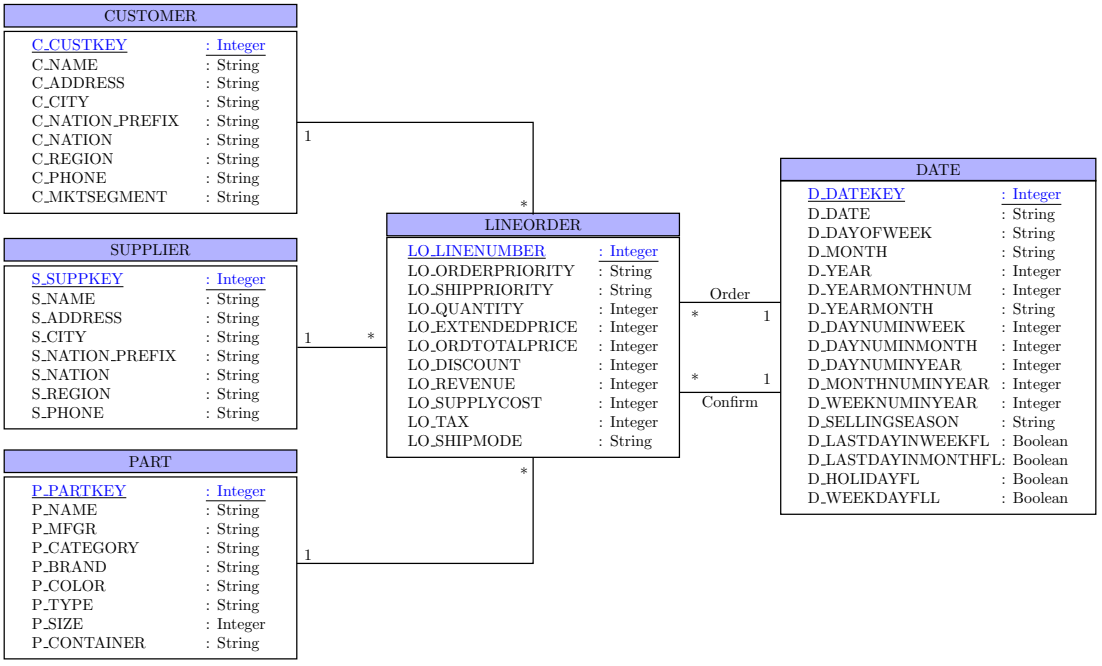


FIGURE 6.1: SSB data warehouse schema

TABLE 6.1: SSB data warehouse's volume

Fact table	**Dimension tables**				**Total**
LINEORDER	**CUSTOMER**	**SUPPLIER**	**PART**	**DATE**	
735 MB	3 MB	956 KB	18 MB	207 KB	757 MB

SSB's query workload is made of four series of queries Q1 to Q4. Q1 bears restrictions on the fact table and dimension Date. Its queries are meant to quantify the revenue increase that would have resulted from eliminating certain company-wide discounts in

a given percentage range for products shipped in a given year. Q1 is subdivided into three subqueries: Q1.1, Q1.2 and Q1.3 (Table 6.2).

TABLE 6.2: SSB query series Q1

Q1.1	SELECT	SUM(lo_extendedprice * lo_discount) AS revenue
	FROM	lineorder, date
	WHERE	lo_orderdate = d_datekey AND d_year = 1993 AND lo_discount between 1 and 3 AND lo_quantity < 25
Q1.2	SELECT	SUM(lo_extendedprice * lo_discount) AS revenue
	FROM	lineorder, date
	WHERE	lo_orderdate = d_datekey AND d_yearmonthnum = 199401 AND lo_discount between 4 and 6 AND lo_quantity between 26 and 35
Q1.3	SELECT	SUM(lo_extendedprice * lo_discount) AS revenue
	FROM	lineorder, date
	WHERE	lo_orderdate = d_datekey AND d_weeknuminyear = 6 AND d_year = 1994 AND lo_discount between 5 and 7 AND lo_quantity between 26 and 35

Q2 operates on dimensions Date, Part and Supplier, and compares revenue for some product classes, for suppliers in a certain region, grouped by more restrictive product classes and all years. Q2 is subdivided into three subqueries: Q2.1, Q2.2 and Q2.3 (Table 6.3).

TABLE 6.3: SSB query series Q2

Q2.1	SELECT	SUM(lo_revenue) AS revenue, d_year, p_brand
	FROM	lineorder, date, part, supplier
	WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND p_category = 'MFGR#12' AND s_region = 'AMERICA'
	GROUP BY	d_year, p_brand
	ORDER BY	d_year, p_brand
Q2.2	SELECT	SUM(lo_revenue) AS revenue, d_year, p_brand
	FROM	lineorder, date, part, supplier
	WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND s_region = 'ASIA' AND p_brand between 'MFGR#2221' and 'MFGR#2228'
	GROUP BY	d_year, p_brand
	ORDER BY	d_year, p_brand
Q2.3	SELECT	SUM(lo_revenue) AS revenue, d_year, p_brand
	FROM	lineorder, date, part, supplier
	WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND s_region = 'EUROPE' AND p_brand = 'MFGR#2221'
	GROUP BY	d_year, p_brand
	ORDER BY	d_year, p_brand

Q3 operates on all dimensions. It provides revenue volume for LINEORDER transactions by customer nation, supplier nation and year within a given region, in a certain time period. Q3 is subdivided into four subqueries: Q3.1, Q3.2, Q3.3 and Q3.4 (Table 6.4).

Finally, Q4 is a "What-If" OLAP sequence that measures profit, measured as (LO_REVENUE - LO_SUPPLYCOST). Q4 is subdivided into three subqueries: Q4.1, Q4.2, and Q4.3 (Table 6.5).

TABLE 6.4: SSB query series Q3

Q3.1	SELECT	SUM(lo_revenue) AS revenue, c_nation, s_nation, d_year
	FROM	customer, lineorder, supplier, date
	WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
		lo_orderdate = d_datekey AND c_region = 'ASIA' AND
		s_region = 'ASIA' AND d_year >= 1992 AND d_year <= 1997
	GROUP BY	c_nation, s_nation, d_year
	ORDER BY	d_year asc, revenue desc
Q3.2	SELECT	SUM(lo_revenue) AS revenue, c_city, s_city, d_year
	FROM	customer, lineorder, supplier, date
	WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
		lo_orderdate = d_datekey AND c_nation = 'UNITED STATES' AND
		s_nation = 'UNITED STATES' AND d_year >= 1992 AND d_year <= 1997
	GROUP BY	c_city, s_city, d_year
	ORDER BY	d_year asc, revenue desc
Q3.3	SELECT	SUM(lo_revenue) AS revenue, c_city, s_city, d_year
	FROM	customer, lineorder, supplier, date
	WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
		lo_orderdate = d_datekey AND d_year >= 1992 AND d_year <= 1997 AND
		(c_city='UNITED KI1' OR c_city='UNITED KI5') AND
		(s_city='UNITED KI1' OR s_city='UNITED KI5')
	GROUP BY	c_city, s_city, d_year
	ORDER BY	d_year asc, revenue desc
Q3.4	SELECT	SUM(lo_revenue) AS revenue, c_city, s_city, d_year
	FROM	customer, lineorder, supplier, date
	WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
		lo_orderdate = d_datekey AND d_yearmonth = 'Dec1997' AND
		(c_city='UNITED KI1' OR c_city='UNITED KI5') AND
		(s_city='UNITED KI1' OR s_city='UNITED KI5')
	GROUP BY	c_city, s_city, d_year
	ORDER BY	d_year asc, revenue desc

TABLE 6.5: SSB query series Q4

Q4.1	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year, c_nation
	FROM	lineorder, date, customer, supplier, part
	WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
		lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND
		c_region = 'AMERICA' AND s_region = 'AMERICA' AND
		(p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
	GROUP BY	d_year, c_nation
	ORDER BY	d_year, c_nation
Q4.2	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year, s_nation, p_category
	FROM	lineorder, date, customer, supplier, part
	WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
		lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND
		c_region = 'AMERICA' AND s_region = 'AMERICA' AND
		(d_year = 1997 OR d_year = 1998) AND
		(p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
	GROUP BY	d_year, s_nation, p_category
	ORDER BY	d_year, s_nation, p_category
Q4.3	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year, s_city, p_brand
	FROM	lineorder, date, customer, supplier, part
	WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
		lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND
		c_region = 'AMERICA' AND s_nation = 'UNITED STATES' AND
		(d_year = 1997 OR d_year = 1998) AND p_category = 'MFGR#14'
	GROUP BY	d_year, c_nation
	ORDER BY	d_year, c_nation

SSB query workload results' sizes are shown in Table 6.6. Finally, SSB's only metric is query response time.

TABLE 6.6: SSB workload results' size

Query set	**Qx.1**	**Qx.2**	**Qx.3**	**Qx.4**	**All queries**
Q.1	12 bytes	11 bytes	11 bytes		34 bytes
Q.2	5.93 KB	1.20 KB	153 bytes		7.28 KB
Q.3	3.87 KB	18.74 KB	767 bytes	74 bytes	23.45 KB
Q.4	789 bytes	2.71 KB	3.46 KB		6.94 KB

6.2.3 Parameter Settings

Table 6.7 shows the parameter settings of all the SSSs we compare this chapter. * means several values can be assigned to the parameter.

TABLE 6.7: Parameter setting

Parameters	**Thompson**	**Hadavi**	**bpVSS**	**fVSS**
n	5	5	5	5
t	4	4	4	4
p	-	-	*	-
$\|\|s_{in}\|\|$	8 bits	*	*	*
$\|\|s_{out}\|\|$	-	-	*	6 bits
w_i	-	-	-	100

Values of n and t impact the security, global volume and time complexity of all approaches. We fix $n = 5$ and $t = 4$, which we empirically found are the "best values" for the following reasons.

1. Since n and t relate to a number of real CSPs, they cannot be too big for economic reasons.
2. t should be big to achieve high privacy, but it should be small to minimize storage, computation and data transfer costs.
3. n must be greater than t to guarantee data availability, but t must be close to n to minimize global share volume.
4. bpVSS guarantees global share volume is lower than t times that of original data when $t > 3$ and data are 32-bit integers.
5. fVSS guarantees no CSP group can break the secret and data can be shared even if some CSPs fail when $n < 2 \times t - 2$ and $t > 2$.

Moreover, in bpVSS, we assign p with different values. Values of p are assigned with respect to types and sizes of SSB's data attributes to construct the smallest possible share volume. Moreover, we assign the same value of p to attributes LO_REVENUE and

LO_SUPPLYCOST in fact table LINEORDER to allow aggregating SUM(LO_REVENUE - LO_SUPPLYCOST) in Q4 directly on shares. Table A.1 in Appendix A provides all values of p by attributes.

To enforce data integrity, the studied SSS approaches construct inner and outer signatures with different functions. The size of signatures impacts verification performance and signature volume. Hence, we must assign different signature sizes to the different approaches to enforce the same verification performance (Table A.1). In Hadavi and fVSS, inner signature sizes $\|s_{in}\|$ must be about half of the data size to detect all incorrect data pieces. In bpVSS, $\|s_{in}\|$ must be fixed to the size of p to meet this condition. The verification performance of Thompson's inner signatures depends on three more parameters: p, g and h (a prime and parameters of a discrete logarithm, respectively). We set them up to $p = 251$, $g = 61$ and $h = 59$ to have inner signatures sizing about half the greatest attribute size, i.e., about 8 bits, because the smallest size of inner signatures that can detect all incorrect data pieces is about half of data size (Section 5.3.2.3.1).

Moreover, the outer signature sizes $\|s_{out}\|$ in bpVSS are fixed at $\|p\|/2$ to help detect all incorrect data pieces (Table A.1). In fVSS, $\|s_{out}\|$ and w_i (the maximum number of child nodes in the outer signature tree) are fixed at 6 bits and 100, respectively, to construct a signature tree allowing all incorrect share pieces to be detected.

Since the numbers of shared records is variable from CSP to CSP in fVSS, we assign the number of shared records as in Section 5.3.3. Table A.2 in Appendix A summarizes the number of shared records stored at CSPs by all compared approaches.

In fVSS-II, we assign different machine sizes to each CSP with respect to the number of shared records and CSP pricing policies (Section 5.3.3). In all other approaches, the most powerful machine VM_3 is assigned to all CSPs. To achieve the best performance, we assign the most powerful (and expensive) VM to the index server in all approaches but bpVSS, which does not use an index server. Table 6.8 summarizes the VMs used by each CSP and the index server.

Table 6.8: Virtual machines

	CSP_1	**CSP_2**	**CSP_3**	**CSP_4**	**CSP_5**	**Index server**
Thompson Hadavi fVSS-I	VM_3	VM_3	VM_3	VM_3	VM_3	VM_3
bpVSS	VM_3	VM_3	VM_3	VM_3	VM_3	-
fVSS-II	VM_3	VM_3	VM_2	VM_1	VM_1	VM_3

Finally, only four CSPs are used in the reconstruction process (CSP_2, CSP_3, CSP_4 and CSP_5). In fVSS-II, we use the least powerful VMs to measure worst case performance.

6.2.4 Experimental protocol

6.2.4.1 Data Sharing

SSB's DW is shared, and then data volume and data sharing time are measured. However, Thompson and Hadavi can share integers, but not strings. Hence, strings are first transformed into integers. Moreover, to avoid bias in the measurements of share volume, Thompson and Hadavi shares are compressed with Huffman code as in fVSS. bpVSS' shares cannot be compressed because they are used in exact match and range queries.

6.2.4.2 Data Reconstruction

We measure global reconstruction time to understand data access performance well, because no SSS executes SSB's workload queries directly (details in Section 6.2.4.3). Moreover, global reconstruction time also helps measure data recovering time for CSPs with erroneous data. To assess global reconstruction time, SSB's whole DW is reconstructed three times and data reconstruction time is averaged. To avoid bias the measurements of data reconstruction time, we execute the queries without caching. Moreover, they are run only three times because data reconstruction time only slightly varies for a given SSS.

6.2.4.3 Data Access

In this series of experiments, each SSb query is run three times without caching, and then transferred data size and mean workload response time are measured. Since the queries are executed without caching, data access time again hardly varies. Moreover, since no SSS allows composite queries on shares, SSB queries must be rewritten with respect to each SSS. Moreover, Thompson and Hadavi do not support GROUP BY clauses on shares either. Hence, in both these SSSs, most shares must be transferred back to the user and decrypted to run GROUP BY queries. To avoid this issue and compare all approaches on a similar basis, GROUP BY queries are run on unencrypted keys, which are constructed as in our SSSs.

For example, Table 6.9 features the rewriting of queries Q1.1 and Q2.1, respectively, where E(d) stands for a share of d. All other rewritten queries are provided in Appendix A.

TABLE 6.9: SSB Q1.1 rewriting

Approaches	subqueries		
Thompson	Q1.1.1	SELECT	d_datekey, d_year
		FROM	date
	Q1.1.2	SELECT	lo_orderkey, lo_linenumber, lo_extendedprice, lo_discount, lo_quantity
		FROM	lineorder
		WHERE	{results from queries Q1.1.1 after filtering at the user's}
Hadavi	Q1.1.1	SELECT	d_datekey
		FROM	date
		WHERE	d_year = 1993
	Q1.1.2	SELECT	lo_orderkey, lo_linenumber
		FROM	lineorder
		WHERE	lo_discount between 1 and 3 AND lo_quantity < 25
	Q1.1.3	SELECT	lo_orderkey, lo_linenumber, lo_extendedprice, lo_discount,
		FROM	lineorder
		WHERE	{results from queries Q1.1.1 and Q1.1.2 }
bpVSS	Q1.1.1	SELECT	d_datekey
		FROM	date
		WHERE	d_year = E(1993)
	Q1.1.2	SELECT	lo_orderkey, lo_linenumber, lo_extendedprice, lo_discount, lo_quantity
		FROM	lineorder
		WHERE	lo_discount IN {E(1),E(2),E(3)} AND lo_quantity IN {E(0),...,E(25)} AND {results from queries Q1.1.1 after filtering at the user's}
fVSS	Q1.1.1	As query Q1.1.1 in Hadavi	
	Q1.1.2	As query Q1.1.2 in Hadavi	
	Q1.1.3	SELECT	SUM(lo_extendedprice * lo_discount) AS revenue
		FROM	lineorder
		WHERE	{results from queries Q1.1.1 and Q1.1.2 }

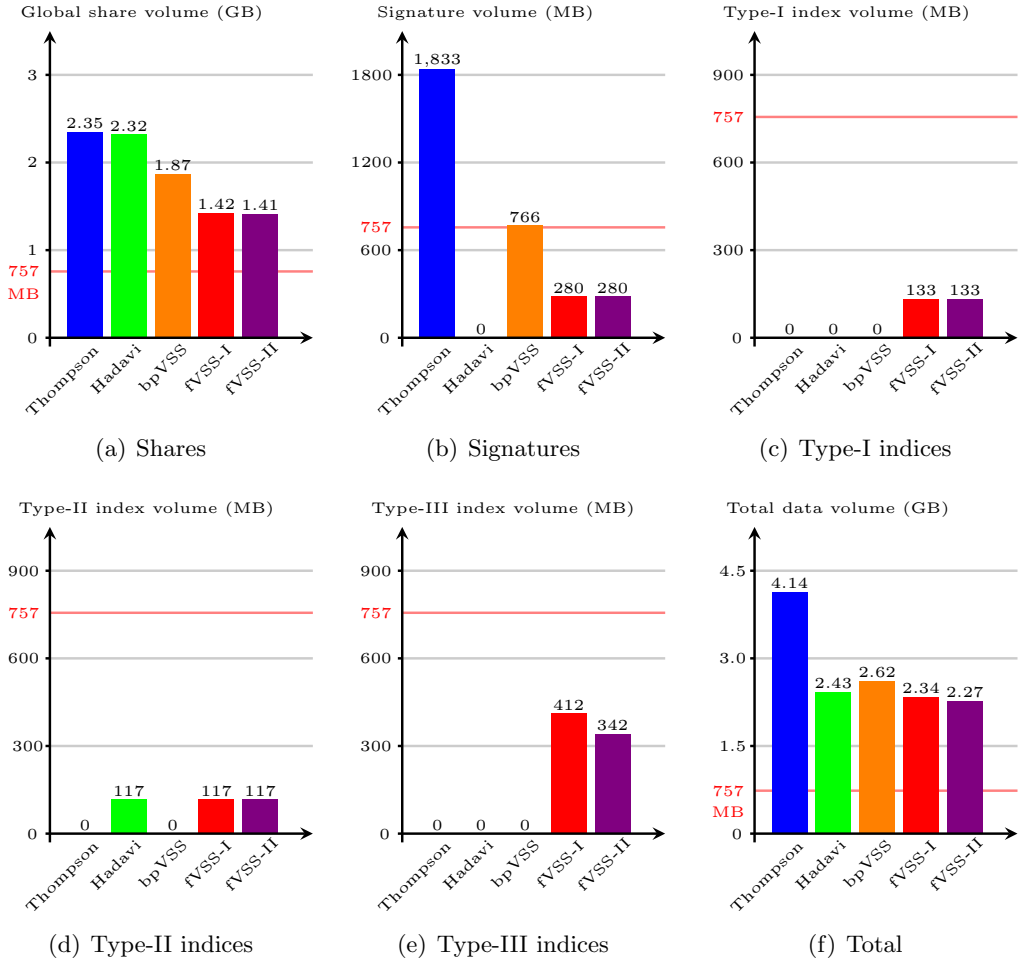
TABLE 6.10: SSB Q2.1 rewriting

Approaches	subqueries		
Thompson	Q2.1.1	SELECT	p_partkey, p_category
		FROM	part
	Q2.1.2	SELECT	s_suppkey, s_region
		FROM	supplier
	Q2.1.3	SELECT	SUM(lo_revenue) AS revenue, d_year_key, p_brand_key
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND {results from queries Q2.1.1 and Q2.1.2 after filtering }
		GROUP BY	d_year_key, p_brand_key
	Q2.1.4	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q2.1.3 }
	Q2.1.5	SELECT	p_brand_key, p_brand
		FROM	part
		WHERE	{results from queries Q2.1.3 }
Hadavi and fVSS	Q2.1.1	SELECT	p_partkey
		FROM	part
		WHERE	p_category = 'MFGR#12'
	Q2.1.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = 'AMERICA'
	Q2.1.3	SELECT	SUM(lo_revenue) AS revenue, d_year_key, p_brand_key
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND {results from queries Q2.1.1 and Q2.1.2 }
		GROUP BY	d_year_key, p_brand_key
	Q2.1.4	As query Q2.1.4 in Thompson	
	Q2.1.5	As query Q2.1.5 in Thompson	
bpVSS	Q2.1.1	SELECT	p_partkey
		FROM	part
		WHERE	p_category = E('MFGR#12')
	Q2.1.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = E('AMERICA')
	Q2.1.3	SELECT	SUM(lo_revenue) AS revenue, d_year_key, p_brand_key, d_year, p_brand
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND {results from queries Q2.1.1 and Q2.1.2 after filtering }
		GROUP BY	d_year_key, p_brand_key

6.3 Experiment Results

6.3.1 Share Volume

Figure 6.2 plots the data volumes featured by all compared approaches, i.e., global share volume (Figure 6.2(a)), signature volume (Figure 6.2(b)), index volume (Figures 6.2(c), 6.2(d) and 6.2(e)) and total volume (Figure 6.2(f)). Detailed tables are provided in Appendix B.



(a) Shares (b) Signatures (c) Type-I indices

(d) Type-II indices (e) Type-III indices (f) Total

FIGURE 6.2: Data volume comparison

Figure 6.2(a) clearly shows that our SSSs bear a lower share volume than that of Thompson and Hadavi. Although these SSSs compress shares of strings with Huffman code, the size of shares is much smaller in bpVSS because data is encrypted into base-p integers, and the number of shared records is much smaller in fVSS (Table A.2) since records are not shared at all CSPs'. These features help our fVSS achieve a global share volume that is lower than twice that of SSB's DW.

Figure 6.2(b) highlights the significant cost of enforcing data integrity with signatures, and shows that our SSSs are again less greedy than state-of-the-art SSSs, although they include outer signatures in addition to inner signatures. Thompson indeed stores both encrypted and unencrypted inner signatures at all CSPs' and the index server's, while other approaches hide inner signatures in shares. Moreover, bpVSS uses smaller outer signatures and fVSS constructs record, group of records, table, group of tables and DB signatures instead of share signatures.

Only Hadavi and our SSSs exploit indices. As for signatures, Figures 6.2(c), 6.2(d) and 6.2(e) show that index volume is significant, but it is the price to pay to allow exact match, range and aggregation queries on shares, which Hadavi only partially allows and Thompson not at all.

Figure 6.2(f) plots the total data volume consumed by all approaches, including shares, signatures and indices. We can see that our SSSs achieve comparable performance with the best existing approach (Hadavi), while featuring better data privacy and integrity and a wider range of possible queries over shares. Figure 6.2(f) also illustrates the advantage of fVSS' signature management over bpVSS'.

Finally, since share record size impacts data sharing/reconstruction time and workload response time (Sections 6.3.2 and 6.3.3, respectively), Figure 6.3 plots share record size. Although our SSSs bear a lower global share volume than that of Thompson and Hadavi, the mean sizes of shared records in our SSSs lie between that of Hadavi and Thompson. This is because signatures are also stored in shared records in bpVSS (as in Thompson), and shares are reals in fVSS, while shares are integers in other approaches and the size of reals are greater than that of integers.

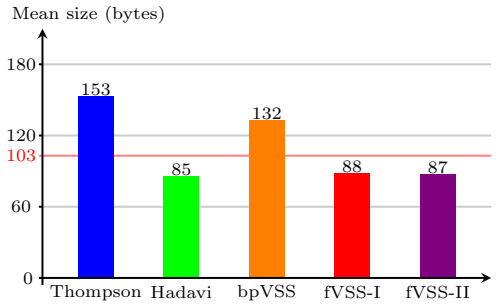
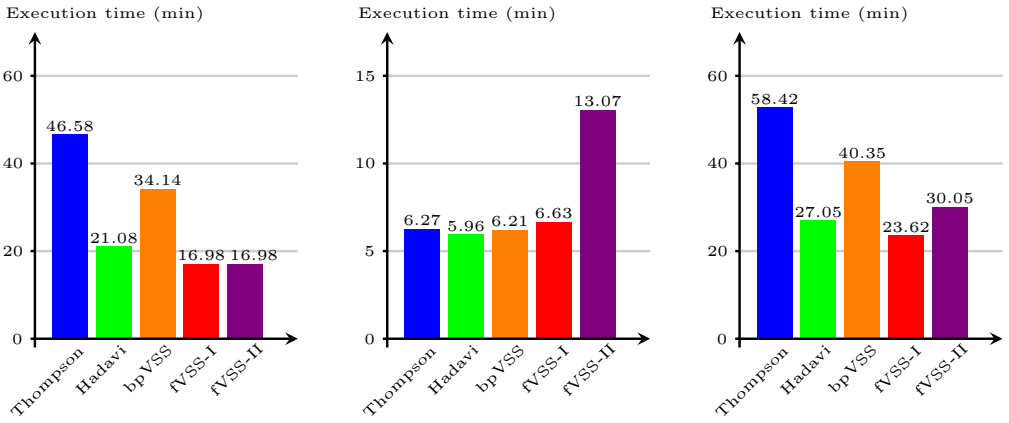


FIGURE 6.3: Mean size of shared records

6.3.2 Data Sharing Time

Figure 6.4 plots data sharing time i.e., execution time at the user's (Figure 6.4(a)), execution time in the cloud (Figure 6.4(b)) and total execution time (Figure 6.4(c)).

Detailed tables are provided in Appendix B.



(a) User (b) Cloud (c) Total

FIGURE 6.4: Data sharing time

Theoretically, all experimental approaches bear the same data encryption time complexity: $O(nt)$ (Section 5.4.2). However, in practice, data encryption time is clearly different in each approach (Figure 6.4(a)). Thompson performs the worst, because it separately encrypts both data and inner signatures, while other approaches encrypt them at once. In contrast, fVSS performs the best, because pseudo shares are computed only one time for all attribute values in the same record and shares are constructed only $n - t + 2 = 3$ times, while shares are constructed 5 times in other approaches. bpVSS' data sharing time is higher, because bpVSS transforms decimal integers into base p integers and constructs outer signatures at the user's side.

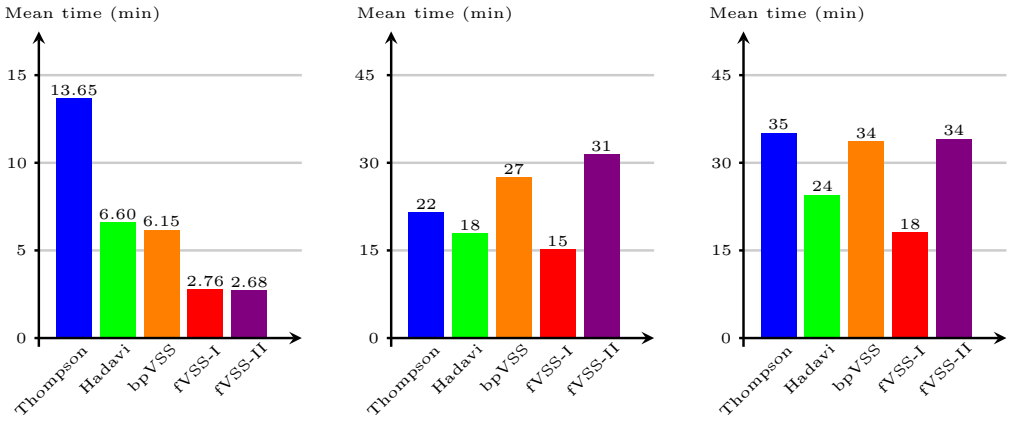
Figure 6.4(b) shows execution time in the cloud, including loading shares and signatures (Figure B.1(b)), building signature trees (Figure B.1(c)), building indices (Figures B.1(d) and B.1(e)). bpVSS' loading time lies between that of Hadavi and Thompson, because the number of shared records is the same than that of Hadavi and Thompson (Table A.2), but the size of records lies between that of Hadavi and Thompson (Figure 6.3). Although Thompson and Hadavi must build signature trees and Type-II indices, respectively, both processes run in parallel with share loading. fVSS-I loads shares faster because the number of shared records is lower. However, its execution time in the cloud is not the lowest, because other processes (i.e., building signature trees and indices) run, and building signature trees are sequentially run after loading shares in fVSS-I. Moreover, fVSS-II's execution time is higher than fVSS-I's, because total execution time of fVSS-II is the execution time of the slowest machines, i.e., at CSP_4 and CSP_5.

Finally, Figure 6.4(c) plots total execution time. We can see that fVSS achieves comparable performance with the best existing, related approach (Hadavi), while allowing more flexibly in sharing data, i.e., allowing to share data even though some CSPs fail. Moreover, Figure 6.4(c) also illustrates that VM power management impacts fVSS' data sharing time. fVSS-II's total execution time is indeed higher than fVSS-I's because the volume of shares does not match with machine power. Finally, although bpVSS shares data slower than Hadavi, it features better data integrity.

6.3.3 Workload Response Time

6.3.3.1 Data Reconstruction Time

Figure 6.5 shows full database reconstruction time in all compared approaches, i.e., execution time at the user's (Figure 6.5(a)), execution time in the cloud (Figure 6.5(b)) and total execution time (Figure 6.5(c)). Detailed tables are provided in Appendix B.



(a) User (b) Cloud (c) Total

FIGURE 6.5: Full database reconstruction time

All approaches bear the same data decryption time complexity: $O(t^2)$ (Section 5.4.2). In practice, Figure 6.5(a) shows that our SSSs bear a lower data decryption time than that of Thompson and Hadavi. This is because bpVSS does not need to extract shares compressed with Huffman code as other approaches, and fVSS' execution time to decrypt each record with Lagrange interpolation is optimized by the use of the same pseudo shares for all data pieces in the same record.

Figure 6.5(b) plots the mean execution time in the cloud, i.e., for querying data and verifying outer signatures. bpVSS' execution time is higher than that of Thompson and Hadavi, because only bpVSS verifies the correctness of shares. Moreover, fVSS-I's execution time is the lowest, because the number of shared records is the lowest

(Table A.2). fVSS-II's execution time is higher than fVSS-I's, because it is tied to the execution time of the slowest VM.

Finally, Figure 6.5(c) plots the total reconstruction time of all approaches. We can see that fVSS-I is the most efficient approach for accessing the whole database. As for the data sharing process, fVSS-II's total reconstruction time is higher than fVSS-I's. Moreover, although bpVSS reconstructs data slower than Hadavi, it guarantees that no erroneous share is transferred back to the user.

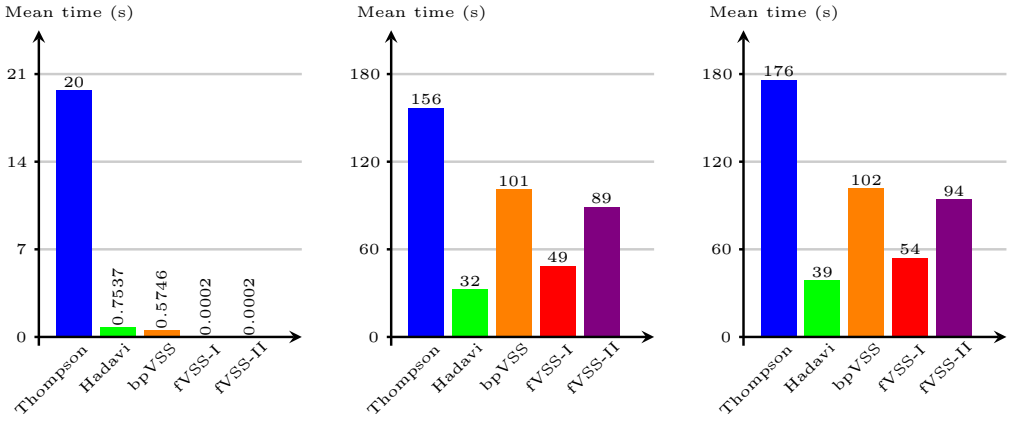
6.3.3.2 Data Access Time

Figure 6.6 shows Q1's response time with all compared approaches, i.e., mean response time at the user's (Figure 6.6(a)), mean response time in the cloud (Figure 6.6(b)) and mean total execution time (Figure 6.6(c)). Similarly to Figure 6.6, Figures 6.7, 6.8, 6.9 and 6.10 show Q2's, Q3's, Q4's and the whole workload's response time, respectively. Detailed tables are provided in Appendix B.

At the user's, our SSSs bear a lower mean execution time than that of Thompson and Hadavi. In Q1, fVSS runs the fastest because only one aggregate piece is reconstructed, since fVSS can perform aggregation SUM(LO_EXTENDEDPRICE * LO_DISCOUNT). In contrast, bpVSS and other approaches cannot compute this sum directly on shares. Hence, the aggregate must be computed from a large volume of reconstructed data. bpVSS' execution time is lower than that of Thompson and Hadavi, because bpVSS decrypts data faster (Section 6.3.3.1). In other queries, all approaches can perform aggregate operations, e.g., SUM(lo_revenue), SUM(LO_REVENUE - LO_SUPPLYCOST) on shares. Yet, our SSSs still run faster because they decrypt data faster.

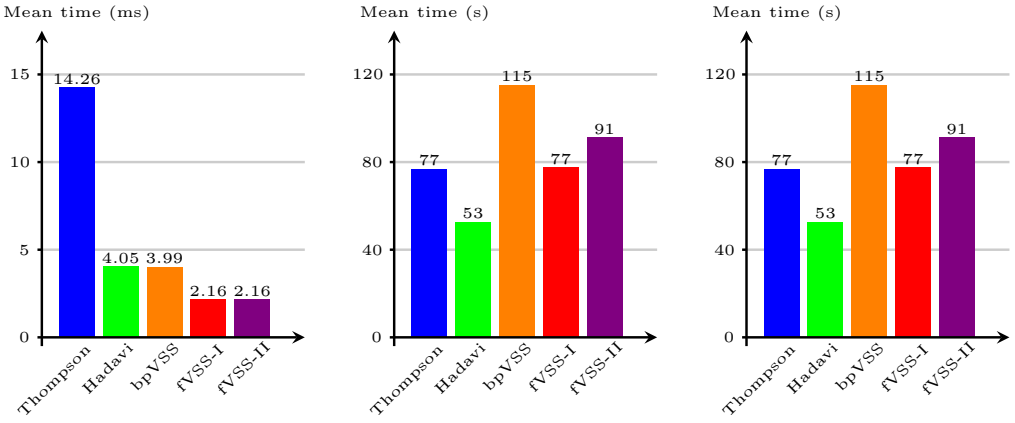
In the cloud, bpVSS' mean execution time is almost always the highest, because only bpVSS verifies the correctness of shares. However, Thompson runs Q1 the slowest, because only Thompson allows neither aggregation nor exact match operations on shares. fVSS run Q1 slower than Hadavi, because aggregate queries in fVSS runs slower than simple SELECT/FROM queries in other approaches. Similarly, fVSS also runs other queries slower than Hadavi, because aggregation on reals (shares) in fVSS is slower than on integers in other approaches.

Total mean execution time shows that our SSSs achieve comparable performance with the best existing approach (Hadavi), while guaranteeing no erroneous share are transferred back to the user (bpVSS) and allowing more numerous aggregation operators on shares (fVSS).



(a) User (b) Cloud (c) Total

FIGURE 6.6: Q1 execution time



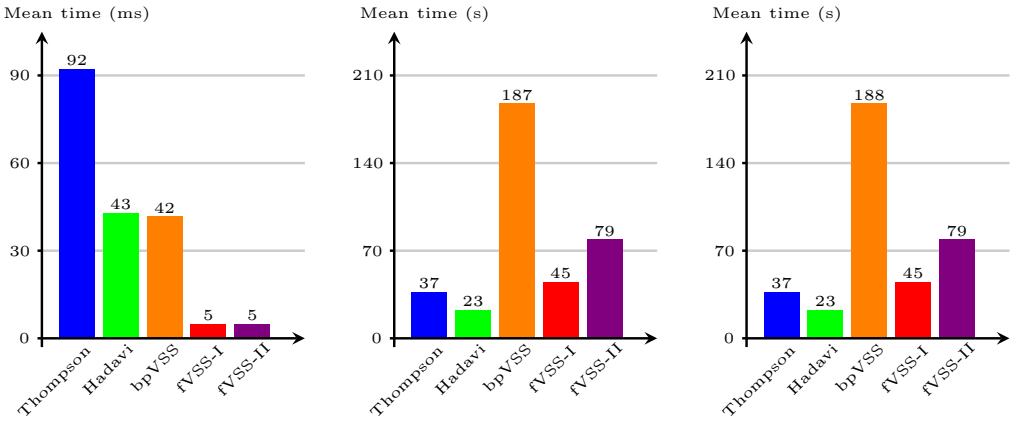
(a) User (b) Cloud (c) Total

FIGURE 6.7: Q2 execution time

6.3.4 Transferred data volume

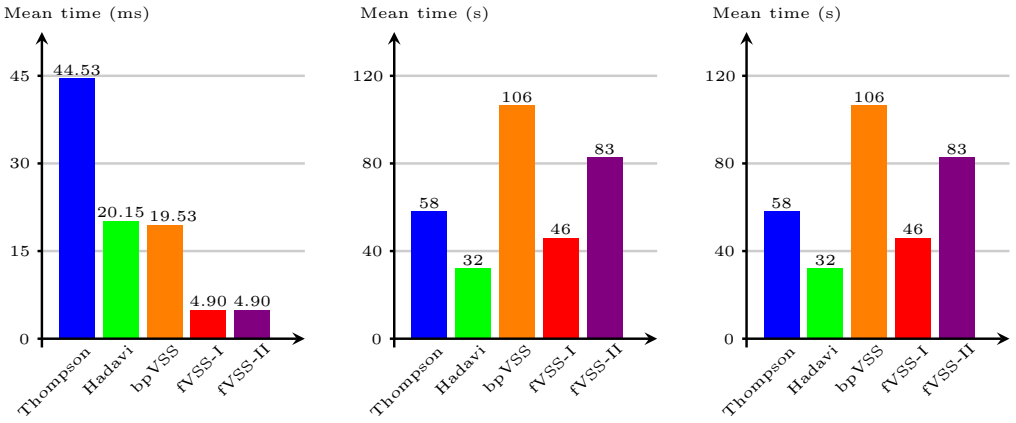
Figure 6.11 plots the data volumes that are transferred back to the user after executing queries, i.e., Q1 result size (Figure 6.11(a)), Q2 result size (Figure 6.11(b)), Q3 result size (Figure 6.11(c)), Q4 result size (Figure 6.11(d)) and whole workload result size (Figure 6.11(e)). SSB query result sizes are featured with red lines. Detailed tables and figures are provided in Appendix B.

Figure 6.11(a) shows that our SSSs bear a lower workload result size than that of Thompson and Hadavi. The size of transferred data is indeed much smaller in our SSSs, because only fVSS can run both aggregation and exact match operators on shares in Q1, and the size of bpVSS' shares is smaller. In contrast, Thompson transfers the largest data volume, because it cannot run aggregation nor exact match operators on shares in Q1.



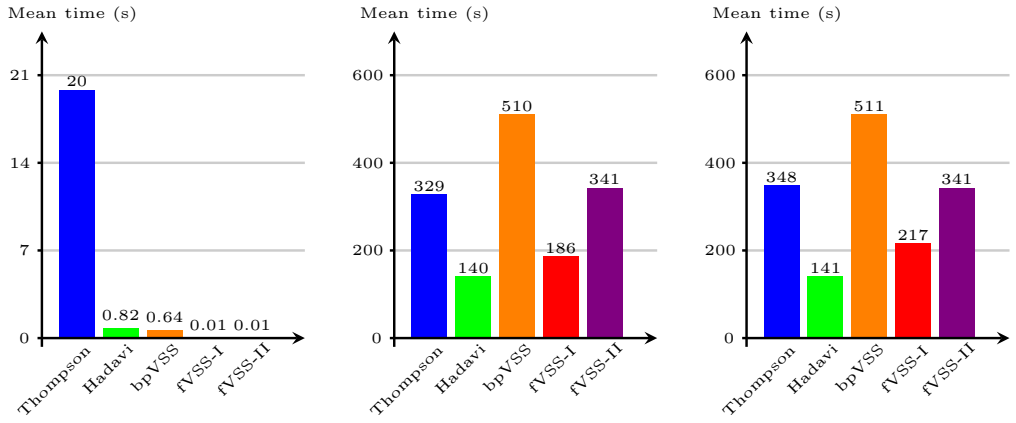
(a) User (b) Cloud (c) Total

FIGURE 6.8: Q3 execution time



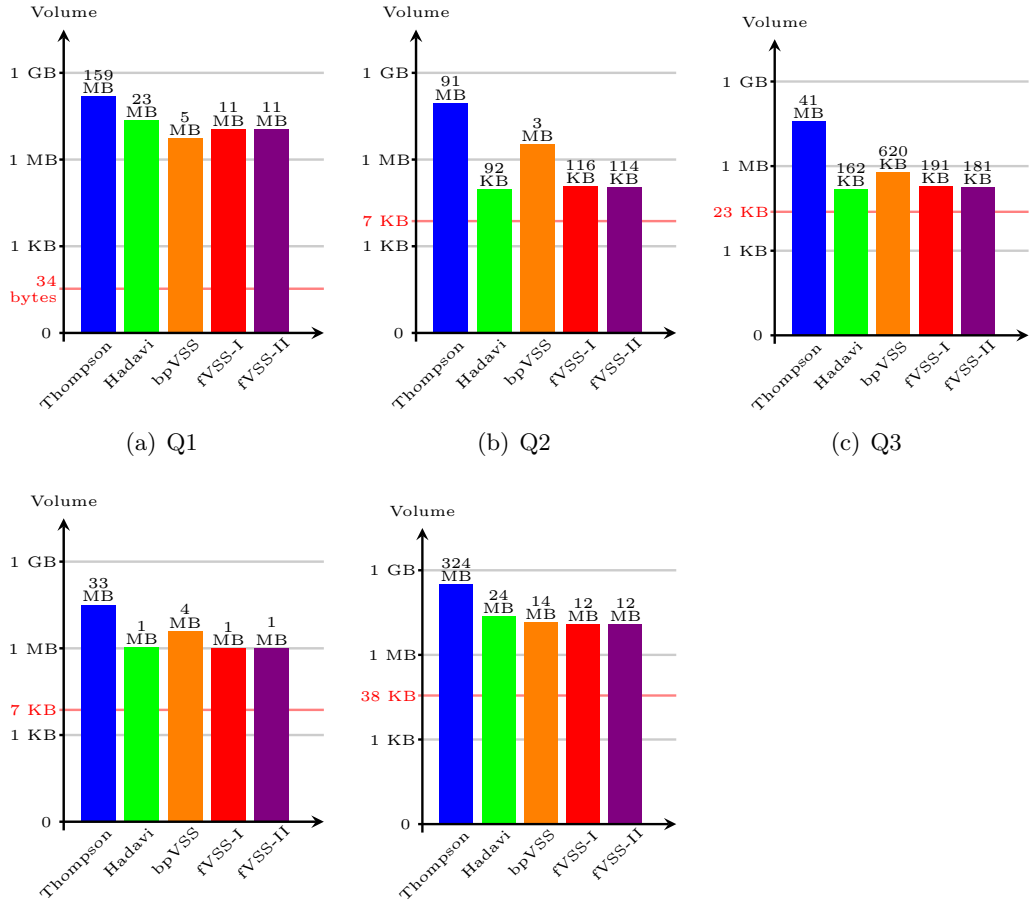
(a) User (b) Cloud (c) Total

FIGURE 6.9: Q4 execution time



(a) User (b) Cloud (c) Total

FIGURE 6.10: Whole workload execution time



(a) Q1 (b) Q2 (c) Q3

(d) Q4 (e) Whole workload

FIGURE 6.11: Transferred data volume

In other queries (Figures 6.11(b), 6.11(c) and 6.11(d)), our SSSs transfer the lowest volume of true positive shares. However, the global size of our SSSs' transferred data volume is greater than that of Hadavi. This is because bpVSS also transfers a large number of false positive shares (resulting from exact match queries), and fVSS also transfers several types of indices to help reconstruct aggregation and exact match results. Hadavi transfers the smallest data volume since only shares and Type-II indices are transferred. In contrast, Thompson transfers the largest data volume since it cannot run exact match operation on shares. Moreover, it also transfers signatures to help verify data correctness at the user's.

Globally, Figure 6.10 clearly shows that our SSSs bear the lowest workload result size. Workload sizes are only half that of the best related, existing approach (Hadavi). Moreover, fVSS features more aggregation operators on shares.

6.4 Discussion

Table 6.11 recapitulates data volume, execution time, transferred data volume of all studied approaches. In addition, it computes monetary costs with respect to CSP pricing policies depicted in Table 5.1 (Chapter 5). We assume that the index server uses the most expensive VM. Details of monetary costs are depicted in Appendix B.5.

TABLE 6.11: Cost comparison of database sharing approaches

	Thompson	**Hadavi**	**bpVSS**	**fVSS-I**	**fVSS-II**
Storage volume (GB)	4.14	2.43	2.62	2.34	**2.27**
Data transfer volume (MB)	323.88	23.90	13.51	12.36	**12.34**
Data sharing time (min)	58.42	27.05	40.35	**23.62**	30.05
Data reconstruction time (min)	35.16	24.50	33.59	**17.95**	34.09
Data access time (min)	5.81	**2.34**	8.51	3.61	5.69
Storage cost ($/month)	$0.5584	$0.3023	$0.2973	$0.3168	**$0.2651**
Data transfer cost ($)	$0.0302	$0.0025	**$0.0012**	$0.0014	$0.0014
Data sharing cost ($)	$0.1214	$0.0935	**$0.0933**	$0.1242	$0.0996
Data reconstruction cost ($)	$0.4092	$0.2599	$0.3964	$0.2641	**$0.2071**
Data access cost ($)	$0.0887	**$0.0314**	$0.1204	$0.0520	$0.0372

Table 6.11 shows that fVSS consumes a lower storage volume, and bpVSS a greater storage volume, respectively, than that of the best existing, related approach (Hadavi). However, storage cost does not only depend on storage volume, but also on CSP pricing policies. bpVSS' storage cost is lower than that of Hadavi, because bpVSS does not use an index server, which is the most expensive VM. Similarly, since fVSS-I's storage volume at the index server's is greater than that of Hadavi, fVSS-I's storage cost is higher than that of Hadavi. However, since fVSS-II's storage volume is the smallest and storage volumes at CSPs' are adjusted with respect to CSP pricing policies, fVSS-II's storage cost is clearly the cheapest.

Thanks to a smaller share size in bpVSS and a wider range of possible queries over shares in fVSS, transferred data volumes of our SSSs are the lowest. Thus, their data transfer costs are also the lowest, i.e., about half that of the most efficient related approach (Hadavi).

bpVSS and fVSS-I share data the slowest and the fastest, respectively. Yet, bpVSS' data sharing cost turns to be the cheapest, because bpVSS does not use an index server, which is the most expensive VM. Similarly, fVSS-I's data sharing cost is the most expensive, because several processes (i.e., loading Type-I indices and building Type-II indices) are executed by the index server. By assigning VM sizes to CSPs with respect to CSP pricing policies, fVSS-II bears a lower data sharing cost than fVSS-I. Moreover, only fVSS allows sharing data even though some CSPs fail.

Unlike the data sharing process, all approaches mainly execute queries on CSPs. Only small processes are run at the index server's in fVSS and Thompson. Hence, data

reconstruction cost is correlated to data reconstruction time. However, fVSS-II's data reconstruction cost turns to be the cheapest because of the benefit of assigning VM sizes at CSPs with respect to CSP pricing policies. Note that although bpVSS's data reconstruction cost is higher than that of the most efficient related approach (Hadavi), only bpVSS guarantees that no erroneous share is transferred back to the user.

Finally, bpVSS' and fVSS' data access cost is higher than that of Hadavi. The benefit of assigning VM size with respect to CSP pricing policies makes fVSS-II's data access cost cheaper than fVSS-I's. However, it cannot turn into the lowest data access cost, because several processes (i.e., accessing Type-I and Type-II indices) are executed at the index server's to run aggregation and exact match queries on shares. To reduce this cost, we aim at extending fVSS to support aggregation and exact match operations on shares without the help of indices in future work.

Chapter 7

Conclusion and Perspectives

In this chapter, we summarize the main issues addressed in this dissertation and the contributions we achieved. We further identify some open research directions as perspectives.

7.1 Conclusion

In a business intelligence context, one of the top concerns of cloud users and would-be users is 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., CSP or subcontractor espionage, cost-effective defense of availability and uncontrolled mashups. Moreover, cloud data warehouses must not only be highly protected, but also effectively refreshed and analyzed through on-line analysis processing. Hence, we focus on data security (data privacy, availability and integrity) and performance.

To handle all concerns, we propose two new approaches for securing cloud DWs, which simultaneously support data privacy, availability, integrity and OLAP. Base-p verifiable secret sharing (bpVSS) and flexible verifiable secret sharing (fVSS) are novel types of verifiable secret sharing. Both approaches construct small shares to minimize global share volume. Moreover, fVSS allows users adjusting share volume with respect to CSP pricing polices. Our experiments show that small shares and unbalancing share volume allows significantly minimizing storage and computing costs in the pay-as-you-go paradigm. Privacy and availability in our approaches are achieved by design with secret sharing. Moreover, fVSS achieves higher privacy (i.e., no CSP group can break the secret) and allows DW and cloud cube refreshing even when some CSPs fail. Finally,

data integrity is reinforced with both inner and outer signatures to help detect errors in query results and shares, respectively.

Several secure DB approaches use secret sharing to handle privacy and availability. Only a few approaches also handle integrity as ours. Our experiments with the Star Schema Benchmark show that global share volume in our approaches is lower than n times the original SSB data volume, and is also lower than global share volume in existing, comparable approaches. Moreover, at data access time, data transfer volumes in our approaches are also lowest. However, our approaches are not the most efficient in sharing and accessing data, since the sharing and reconstruction processes in our approaches are more complex (to minimize share volume and construct/verify outer signatures), and thus cause longer execution timees. Moreover, fVSS' shares are reals, while other approaches' shares are integers, and reals process slower than integers.

7.2 Perspectives

In this section, we discuss some open issues that should be addressed in our approaches to improve their security, performance, cost and usage.

First, we plan to improve data privacy, since in bpVSS, table names, attribute names and primary and foreign keys are unencrypted. Thus, encrypted data may be attacked with, e.g., banburismus, dictionary or frequency methods (with the help of a knowledge base). Thus, to achieve higher security, table names, attribute names (including in fVSS) and keys should be encrypted to hide the data schema. No current SSS can achieve this because each table name (after its name is encrypted) must be different from other table names if they are stored in the same CSP/node. Similarly, encrypted attribute names and encrypted primary keys in each shared table must be different. Moreover, encrypted keys in a foreign-primary keys couple should be different to hide the relationship between encrypted tables. Hence, we plan to use an injective and one-way function to encrypt table names, attribute names and keys in each table at each node.

Second, we seek to minimize the risk of bottleneck and optimize query response time. Although our approaches allow exact match, range and aggregation queries running on shares, they cannot run composite queries, i.e., queries involving both exact match and aggregation operators. Hence, when composite queries are run, large data volumes are transferred between the user and CSPs (Chapter 6). In bpVSS, this problem may happen because both true and false positive results of exact match queries are transferred to the user's for filtering. Then, only true positives are transferred back to CSPs to process

the aggregation. Thus, we plan to eliminate false positives of exact match and range queries to help composite queries run on shares. In fVSS, all exact match and range queries and some aggregate queries must run on indices (bitmaps and B+ trees) stored at the index server. Moreover, share placement is random. Thus, we should seek for a solution to organize share placement without the help of indices.

Third, we plan to further improve the cost of our solution in the cloud pay-as-you-go paradigm. fVSS' experiments (Chapter 6) show high costs (storage, computation and data transfer costs) are paid for the index server. Moreover, execution time is high if data volume does not balance with machine power, because data sharing or access runs parallely on CSPs and total execution time is thus the highest individual execution time. Hence, we should eliminate the index server and design 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. This is possible if share placement is organized as in the previous paragraph. However, estimating share volume at each CSP's with respect to cost and machine power is still a challenge.

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 in fVSS, with the lowest possible update costs, while still preserving data integrity. In all SSSs including our approaches, any CSP can be immediately removed without any impact even if $n \geq t$. In contrast, when a new CSP is added, shares of all already existing data must be shared at the new CSP's to handle data consistency. The shares stored at the new CSP's must be constructed from t existing CSPs and thus, the construction process is costly, both in execution time and bandwidth. To avoid this, we can still exploit pseudo shares in fVSS, but functions to construct pseudo shares in new and existing CSPs must be redefined dynamically to link all CSPs, which is a challenge.

Finally, our approaches can be applied to store and analyze big data. Since big data are huge (volume), come quickly (velocity) and are heterogeneous (variety), we discuss how our approaches may handle these issues.

First, our approaches are distributed database approaches, where the volume of individual shares decreases when the number of CSPs increases. However, the number n of CSPs is limited in practice, and share volume at each CSP may not be small enough to fit in a database. Thus, to store a big data volume, we could change our design strategy from "one participant per CSP" to "many participants per CSP".

Second, in our approaches, data access runs parallely on participants (one or more participants per one CSP), and only necessary data pieces are reconstructed at the user's. When the number of participants increases, access time at each participants

decreases, because the volume of shares decreases and the number of shared records may also decrease (in fVSS). Thus, our approaches, especially fVSS, can query stream data. However, sharing stream data may be a problem, because all data pieces must be encrypted at the user's and the polynomial encryption time increases with the number of participants. Moreover, the number of participants must be big enough to store a big data volume. Thus, encryption complexity must fall under polynomial time to allow sharing stream data. We expect this will be solved by some related researches from (e.g., the numerical data or the cryptography fields) in the future.

Third, our approaches can already share any type of data if data items are identified by a key and their type can convert to integers, reals, characters, strings or binary strings. For example, text files can be shared with our approaches by only generating the identity key of each files and then encrypting strings in text files. Hence, we can share data types such as textual or XML documents, images, videos or graphs.

Appendix A

Experimental Setup Details

TABLE A.1: SSS parameter setting

Tables	**Attributes**	**bpVSS**			**fVSS and Hadavi's**
		p	$\|\|s_{in}\|\|$	$\|\|s_{out}\|\|$	$\|\|s_{in}\|\|$
All tables	string attributes	11	4 bits	2 bits	4 bits
LINEORDER	LO_QUANTITY	5	3 bits	2 bits	3 bits
	LO_EXTENDEDPRICE	41	6 bits	3 bits	8 bits
	LO_ORDTOTALPRICE	73	7 bits	4 bits	10 bits
	LO_DISCOUNT	3	2 bits	1 bits	2 bits
	LO_REVENUE	61	6 bits	3 bits	8 bits
	LO_SUPPLYCOST	61	6 bits	3 bits	8 bits
	LO_TAX	3	2 bits	1 bits	2 bits
PART	P_SIZE	5	3 bits	2 bits	3 bits
DATE	D_YEAR	13	4 bits	2 bits	6 bits
	D_YEARMONTHNUM	101	7 bits	4 bits	9 bits
	D_DAYNUMINWEEK	2	1 bits	1 bits	2 bits
	D_DAYNUMINMONTH	5	3 bits	2 bits	3 bits
	D_DAYNUMINYEAR	11	4 bits	2 bits	5 bits
	D_MONTHNUMINYEAR	3	2 bits	1 bits	2 bits
	D_WEEKNUMINYEAR	5	3 bits	2 bits	3 bits

TABLE A.2: Number of shared records

		Fact table	**Dimension tables**				
		LINEORDER	**CUSTOMER**	**SUPPLIER**	**PART**	**DATE**	**Total**
Thompson, Hadavi and bpVSS	each CSP	6,001,171	30,000	10,000	200,000	2,556	6,243,727
	Total	30,005,855	150,000	50,000	1,000,000	12,780	31,218,635
fVSS-I	each CSP	3,600,702	18,000	6,000	120,000	1,533	3,746,235
	Total	18,003,510	90,000	30,000	600,000	7,665	18,731,175
fVSS-II	CSP_1	4,799,099	24,000	8,000	160,000	2,057	4,993,156
	CSP_2	4,800,875	24,000	8,000	160,000	2,052	4,994,927
	CSP_3	3,600,576	18,000	6,000	120,000	1,553	3,746,129
	CSP_4	2,400,595	12,000	4,000	80,000	1,003	2,497,598
	CSP_5	2,402,368	12,000	4,000	80,000	1,003	2,499,371
	Total	18,003,510	90,000	30,000	600,000	7,665	18,731,175

TABLE A.3: SSB Q1.2 rewriting

Approaches	subqueries		
Thompson	Q1.2.1	SELECT	d_datekey, d_year
		FROM	date
	Q1.2.2	SELECT	lo_orderkey, lo_linenumber,
			lo_extendedprice, lo_discount, lo_quantity
		FROM	lineorder
		WHERE	{results from queries Q1.2.1 after filtering at the user's}
Hadavi	Q1.2.1	SELECT	d_datekey
		FROM	date
		WHERE	d_yearmonthnum = 199401
	Q1.2.2	SELECT	lo_orderkey, lo_linenumber
		FROM	lineorder
		WHERE	lo_discount between 4 and 6 AND lo_quantity between 26 and 35
	Q1.2.3	SELECT	lo_orderkey, lo_linenumber, lo_extendedprice, lo_discount,
		FROM	lineorder
		WHERE	{results from queries Q1.2.1 and Q1.2.2}
bpVSS	Q1.2.1	SELECT	d_datekey
		FROM	date
		WHERE	d_yearmonthnum = E(199401)
	Q1.2.2	SELECT	lo_orderkey, lo_linenumber,
			lo_extendedprice, lo_discount, lo_quantity
		FROM	lineorder
		WHERE	lo_discount IN {E(4),E(5),E(6)} AND lo_quantity IN {E(26),...,E(35)}
			AND {results from queries Q1.2.1 after filtering at the user's}
fVSS	Q1.2.1	As query Q1.2.1 in Hadavi	
	Q1.2.2	As query Q1.2.2 in Hadavi	
	Q1.2.3	SELECT	SUM(lo_extendedprice * lo_discount) AS revenue
		FROM	lineorder
		WHERE	{results from queries Q1.2.1 and Q1.2.2}

TABLE A.4: SSB Q1.3 rewriting

Approaches	subqueries		
Thompson	Q1.3.1	SELECT	d_datekey, d_year
		FROM	date
	Q1.3.2	SELECT	lo_orderkey, lo_linenumber,
			lo_extendedprice, lo_discount, lo_quantity
		FROM	lineorder
		WHERE	{results from queries Q1.3.1 after filtering at the user's}
Hadavi	Q1.3.1	SELECT	d_datekey
		FROM	date
		WHERE	d_weeknuminyear = 6 AND d_year = 1994
	Q1.3.2	SELECT	lo_orderkey, lo_linenumber
		FROM	lineorder
		WHERE	lo_discount between 5 and 7 AND lo_quantity between 26 and 35
	Q1.3.3	SELECT	lo_orderkey, lo_linenumber, lo_extendedprice, lo_discount,
		FROM	lineorder
		WHERE	{results from queries Q1.3.1 and Q1.3.2}
bpVSS	Q1.3.1	SELECT	d_datekey
		FROM	date
		WHERE	d_weeknuminyear = E(6) AND d_year = E(1994)
	Q1.3.2	SELECT	lo_orderkey, lo_linenumber,
			lo_extendedprice, lo_discount, lo_quantity
		FROM	lineorder
		WHERE	lo_discount IN {E(5),E(6),E(7)} AND lo_quantity IN {E(26),...,E(35)}
			AND {results from queries Q1.3.1 after filtering at the user's}
fVSS	Q1.3.1	As query Q1.3.1 in Hadavi	
	Q1.3.2	As query Q1.3.2 in Hadavi	
	Q1.3.3	SELECT	SUM(lo_extendedprice * lo_discount) AS revenue
		FROM	lineorder
		WHERE	{results from queries Q1.3.1 and Q1.3.2}

TABLE A.5: SSB Q2.2 rewriting

Approaches	subqueries		
Thompson	Q2.2.1	SELECT	p_partkey, p_brand
		FROM	part
	Q2.2.2	SELECT	s_suppkey, s_region
		FROM	supplier
	Q2.2.3	SELECT	SUM(lo_revenue) AS revenue, d_year_key, p_brand_key
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			{results from queries Q2.2.1 and Q2.2.2 after filtering}
		GROUP BY	d_year_key, p_brand_key
	Q2.2.4	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q2.2.3}
	Q2.2.5	SELECT	p_brand_key, p_brand
		FROM	part
		WHERE	{results from queries Q2.2.3}
Hadavi and fVSS	Q2.2.1	SELECT	p_partkey
		FROM	part
		WHERE	p_brand between 'MFGR#2221' and 'MFGR#2228'
	Q2.2.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = 'ASIA'
	Q2.2.3	SELECT	SUM(lo_revenue) AS revenue, d_year_key, p_brand_key
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			{results from queries Q2.2.1 and Q2.2.2}
		GROUP BY	d_year_key, p_brand_key
	Q2.2.4	As query Q2.2.4 in Thompson	
	Q2.2.5	As query Q2.2.5 in Thompson	
bpVSS	Q2.2.1	SELECT	p_partkey
		FROM	part
		WHERE	p_brand IN {E('MFGR#2221'),$\cdots$,E('MFGR#2228')}
	Q2.2.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = E('ASIA')
	Q2.2.3	SELECT	SUM(lo_revenue) AS revenue,
			d_year_key, p_brand_key, d_year, p_brand
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			{results from queries Q2.2.1 and Q2.2.2 after filtering}
		GROUP BY	d_year_key, p_brand_key

TABLE A.6: SSB Q2.3 rewriting

Approaches	subqueries		
Thompson	Q2.3.1	SELECT	p_partkey, p_brand
		FROM	part
	Q2.3.2	SELECT	s_suppkey, s_region
		FROM	supplier
	Q2.3.3	SELECT	SUM(lo_revenue) AS revenue, d_year_key, p_brand_key
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			{results from queries Q2.3.1 and Q2.3.2 after filtering}
		GROUP BY	d_year_key, p_brand_key
	Q2.3.4	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q2.3.3}
	Q2.3.5	SELECT	p_brand_key, p_brand
		FROM	part
		WHERE	{results from queries Q2.3.3}
Hadavi and fVSS	Q2.3.1	SELECT	p_partkey
		FROM	part
		WHERE	p_brand = 'MFGR#2221'
	Q2.3.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = 'EUROPE'
	Q2.3.3	SELECT	SUM(lo_revenue) AS revenue, d_year_key, p_brand_key
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			{results from queries Q2.3.1 and Q2.3.2}
		GROUP BY	d_year_key, p_brand_key
	Q2.3.4	As query Q2.3.4 in Thompson	
	Q2.3.5	As query Q2.3.5 in Thompson	
bpVSS	Q2.3.1	SELECT	p_partkey
		FROM	part
		WHERE	p_brand = E('MFGR#2221')
	Q2.3.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = E('EUROPE')
	Q2.3.3	SELECT	SUM(lo_revenue) AS revenue,
			d_year_key, p_brand_key, d_year, p_brand
		FROM	lineorder, date, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			{results from queries Q2.3.1 and Q2.3.2 after filtering}
		GROUP BY	d_year_key, p_brand_key

TABLE A.7: SSB Q3.1 rewriting

Approaches	subqueries		
Thompson	Q3.1.1	SELECT	c_custkey, c_region
		FROM	customer
	Q3.1.2	SELECT	s_suppkey, s_region
		FROM	supplier
	Q3.1.3	SELECT	d_datekey, d_year
		FROM	date
	Q3.1.4	SELECT	SUM(lo_revenue) AS revenue,
			c_nation_key, s_nation_key, d_year_key
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.1.1, Q3.1.2 and Q3.1.3 after filtering}
		GROUP BY	c_nation_key, s_nation_key, d_year_key
	Q3.1.5	SELECT	c_nation_key, c_nation
		FROM	customer
		WHERE	{results from queries Q3.1.4}
	Q3.1.6	SELECT	s_nation_key, s_nation
		FROM	supplier
		WHERE	{results from queries Q3.1.4}
	Q3.1.7	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q3.1.4}
Hadavi and fVSS	Q3.1.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_region = 'ASIA'
	Q3.1.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = 'ASIA'
	Q3.1.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year >= 1992 AND d_year <= 1997
	Q3.1.4	SELECT	SUM(lo_revenue) AS revenue,
			c_nation_key, s_nation_key, d_year_key
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.1.1, Q3.1.2 and Q3.1.3}
		GROUP BY	c_nation_key, s_nation_key, d_year_key
	Q3.1.5	As query Q3.1.5 in Thompson	
	Q3.1.6	As query Q3.1.6 in Thompson	
	Q3.1.7	As query Q3.1.7 in Thompson	
bpVSS	Q3.1.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_region = E('ASIA')
	Q3.1.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = E('ASIA')
	Q3.1.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year IN {E(1992),$\cdots$,E(1997)}
	Q3.1.4	SELECT	SUM(lo_revenue) AS revenue,
			c_nation_key, c_nation, s_nation_key, s_nation, d_year_key, d_year
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.1.1, Q3.1.2 and Q3.1.3 after filtering}
		GROUP BY	c_nation_key, s_nation_key, d_year_key

TABLE A.8: SSB Q3.2 rewriting

Approaches	subqueries		
Thompson	Q3.2.1	SELECT	c_custkey, c_nation
		FROM	customer
	Q3.2.2	SELECT	s_suppkey, s_nation
		FROM	supplier
	Q3.2.3	SELECT	d_datekey, d_year
		FROM	date
	Q3.2.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, s_city_key, d_year_key
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.2.1, Q3.2.2 and Q3.2.3 after filtering}
		GROUP BY	c_city_key, s_city_key, d_year_key
	Q3.2.5	SELECT	c_city_key, c_city
		FROM	customer
		WHERE	{results from queries Q3.2.4}
	Q3.2.6	SELECT	s_city_key, s_city
		FROM	supplier
		WHERE	{results from queries Q3.2.4}
	Q3.2.7	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q3.2.4}
Hadavi and fVSS	Q3.2.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_nation = 'UNITED STATES'
	Q3.2.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_nation = 'UNITED STATES'
	Q3.2.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year >= 1992 AND d_year <= 1997
	Q3.2.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, s_city_key, d_year_key
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.2.1, Q3.2.2 and Q3.2.3}
		GROUP BY	c_city_key, s_city_key, d_year_key
	Q3.2.5	As query Q3.2.5 in Thompson	
	Q3.2.6	As query Q3.2.6 in Thompson	
	Q3.2.7	As query Q3.2.7 in Thompson	
bpVSS	Q3.2.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_nation = E('UNITED STATES')
	Q3.2.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_nation = E('UNITED STATES')
	Q3.2.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year IN {E(1992),$\cdots$,E(1997)}
	Q3.2.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, c_city s_city_key, s_city, d_year_key, d_year
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.2.1, Q3.2.2 and Q3.2.3 after filtering}
		GROUP BY	c_city_key, s_city_key, d_year_key

TABLE A.9: SSB Q3.3 rewriting

Approaches	subqueries		
Thompson	Q3.3.1	SELECT	c_custkey, c_city
		FROM	customer
	Q3.3.2	SELECT	s_suppkey, s_city
		FROM	supplier
	Q3.3.3	SELECT	d_datekey, d_year
		FROM	date
	Q3.3.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, s_city_key, d_year_key
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.3.1, Q3.3.2 and Q3.3.3 after filtering}
		GROUP BY	c_city_key, s_city_key, d_year_key
	Q3.3.5	SELECT	c_city_key, c_city
		FROM	customer
		WHERE	{results from queries Q3.3.4}
	Q3.3.6	SELECT	s_city_key, s_city
		FROM	supplier
		WHERE	{results from queries Q3.3.4}
	Q3.3.7	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q3.3.4}
Hadavi and fVSS	Q3.3.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_city='UNITED KI1' OR c_city='UNITED KI5'
	Q3.3.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_city='UNITED KI1' OR s_city='UNITED KI5'
	Q3.3.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year >= 1992 AND d_year <= 1997
	Q3.3.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, s_city_key, d_year_key
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.3.1, Q3.3.2 and Q3.3.3}
		GROUP BY	c_city_key, s_city_key, d_year_key
	Q3.3.5	As query Q3.3.5 in Thompson	
	Q3.3.6	As query Q3.3.6 in Thompson	
	Q3.3.7	As query Q3.3.7 in Thompson	
bpVSS	Q3.3.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_city=E('UNITED KI1') OR c_city=E('UNITED KI5')
	Q3.3.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_city=E('UNITED KI1') OR s_city=E('UNITED KI5')
	Q3.3.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year IN {E(1992),$\cdots$,E(1997)}
	Q3.3.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, c_city s_city_key, s_city, d_year_key, d_year
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.3.1, Q3.3.2 and Q3.3.3 after filtering}
		GROUP BY	c_city_key, s_city_key, d_year_key

TABLE A.10: SSB Q3.4 rewriting

Approaches	subqueries		
Thompson	Q3.4.1	SELECT	c_custkey, c_city
		FROM	customer
	Q3.4.2	SELECT	s_suppkey, s_city
		FROM	supplier
	Q3.4.3	SELECT	d_datekey, d_yearmonth
		FROM	date
	Q3.4.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, s_city_key, d_year_key
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.4.1, Q3.4.2 and Q3.4.3 after filtering}
		GROUP BY	c_city_key, s_city_key, d_year_key
	Q3.4.5	SELECT	c_city_key, c_city
		FROM	customer
		WHERE	{results from queries Q3.4.4}
	Q3.4.6	SELECT	s_city_key, s_city
		FROM	supplier
		WHERE	{results from queries Q3.4.4}
	Q3.4.7	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q3.4.4}
Hadavi and fVSS	Q3.4.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_city='UNITED KI1' OR c_city='UNITED KI5'
	Q3.4.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_city='UNITED KI1' OR s_city='UNITED KI5'
	Q3.4.3	SELECT	d_datekey
		FROM	date
		WHERE	d_yearmonth = 'Dec1997'
	Q3.4.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, s_city_key, d_year_key
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.4.1, Q3.4.2 and Q3.4.3}
		GROUP BY	c_city_key, s_city_key, d_year_key
	Q3.4.5	As query Q3.4.5 in Thompson	
	Q3.4.6	As query Q3.4.6 in Thompson	
	Q3.4.7	As query Q3.4.7 in Thompson	
bpVSS	Q3.4.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_city=E('UNITED KI1') OR c_city=E('UNITED KI5')
	Q3.4.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_city=E('UNITED KI1') OR s_city=E('UNITED KI5')
	Q3.4.3	SELECT	d_datekey
		FROM	date
		WHERE	d_yearmonth = E('Dec1997')
	Q3.4.4	SELECT	SUM(lo_revenue) AS revenue,
			c_city_key, c_city s_city_key, s_city, d_year_key, d_year
		FROM	customer, lineorder, supplier, date
		WHERE	lo_custkey = c_custkey AND lo_suppkey = s_suppkey AND
			lo_orderdate = d_datekey AND
			{results from queries Q3.4.1, Q3.4.2 and Q3.4.3 after filtering}
		GROUP BY	c_city_key, s_city_key, d_year_key

TABLE A.11: SSB Q4.1 rewriting

Approaches	subqueries		
Thompson	Q4.1.1	SELECT	c_custkey, c_region
		FROM	customer
	Q4.1.2	SELECT	s_suppkey, s_region
		FROM	supplier
	Q4.1.3	SELECT	p_partkey, p_mfgr
		FROM	part
	Q4.1.4	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year_key, c_nation_key
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND
			{results from queries Q4.1.1, Q4.1.2 and Q4.1.3 after filtering}
		GROUP BY	d_year_key, c_nation_key
	Q4.1.5	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q4.1.4}
	Q4.1.6	SELECT	c_nation_key, c_nation
		FROM	customer
		WHERE	{results from queries Q4.1.4}
Hadavi and fVSS	Q4.1.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_region = 'AMERICA'
	Q4.1.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = 'AMERICA'
	Q4.1.3	SELECT	p_partkey
		FROM	part
		WHERE	p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2'
	Q4.1.4	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year_key, c_nation_key
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND
			{results from queries Q4.1.1, Q4.1.2 and Q4.1.3}
		GROUP BY	d_year_key, c_nation_key
	Q4.1.5	As query Q4.1.5 in Thompson	
	Q4.1.6	As query Q4.1.6 in Thompson	
bpVSS	Q4.1.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_region = E('AMERICA')
	Q4.1.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = E('AMERICA')
	Q4.1.3	SELECT	p_partkey
		FROM	part
		WHERE	p_mfgr = E('MFGR#1') OR p_mfgr = E('MFGR#2')
	Q4.1.4	SELECT	SUM(lo_revenue - lo_supplycost) as profit,
			d_year_key, _year, c_nation_key, c_nation
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND
			lo_suppkey = s_suppkey AND lo_custkey = c_custkey ANDD
			{results from queries Q4.1.1, Q4.1.2 and Q4.1.3 after filtering}
		GROUP BY	d_year_key, c_nation_key

TABLE A.12: SSB Q4.2 rewriting

Approaches	subqueries		
Thompson	Q4.2.1	SELECT	c_custkey, c_region
		FROM	customer
	Q4.2.2	SELECT	s_suppkey, s_region
		FROM	supplier
	Q3.2.3	SELECT	d_datekey, d_year
		FROM	date
	Q4.2.4	SELECT	p_partkey, p_mfgr
		FROM	part
	Q4.2.5	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year_key, s_nation_key, p_category_key
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND {results from queries Q4.2.1, Q4.2.2, Q4.2.3 and Q4.2.4 after filtering}
		GROUP BY	d_year_key, s_nation_key, p_category_key
	Q4.2.6	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q4.2.5}
	Q4.2.7	SELECT	s_nation_key, s_nation
		FROM	supplier
		WHERE	{results from queries Q4.2.5}
	Q4.2.8	SELECT	p_category_key, p_category
		FROM	part
		WHERE	{results from queries Q4.2.5}
Hadavi and fVSS	Q4.2.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_region = 'AMERICA'
	Q4.2.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = 'AMERICA'
	Q3.2.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year = 1997 OR d_year = 1998
	Q4.2.4	SELECT	p_partkey
		FROM	part
		WHERE	p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2'
	Q4.2.5	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year_key, s_nation_key, p_category_key
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND {results from queries Q4.2.1, Q4.2.2, Q4.2.3 and Q4.2.4}
		GROUP BY	d_year_key, s_nation_key, p_category_key
	Q4.2.6	As query Q4.2.6 in Thompson	
	Q4.2.7	As query Q4.2.7 in Thompson	
	Q4.2.8	As query Q4.2.8 in Thompson	
bpVSS	Q4.2.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_region = E('AMERICA')
	Q4.2.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_region = E('AMERICA')
	Q3.2.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year = E(1997) OR d_year = E(1998)
	Q4.2.4	SELECT	p_partkey
		FROM	part
		WHERE	p_mfgr = E('MFGR#1') OR p_mfgr = E('MFGR#2')
	Q4.2.5	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year_key, d_year, s_nation_key, s_nation, p_category_key, p_category
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND lo_custkey = c_custkey ANDD {results from queries Q4.2.1, Q4.2.2, Q4.2.3 and Q4.2.3 after filtering}
		GROUP BY	d_year_key, s_nation_key, p_category_key

TABLE A.13: SSB Q4.3 rewriting

Approaches	subqueries		
Thompson	Q4.3.1	SELECT	c_custkey, c_region
		FROM	customer
	Q4.3.2	SELECT	s_suppkey, s_nation
		FROM	supplier
	Q3.2.3	SELECT	d_datekey, d_year
		FROM	date
	Q4.3.4	SELECT	p_partkey, p_category
		FROM	part
	Q4.3.5	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year_key, s_city_key, p_brand_key
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND {results from queries Q4.3.1, Q4.3.2, Q4.3.3 and Q4.3.4 after filtering}
		GROUP BY	d_year_key, s_city_key, p_brand_key
	Q4.3.6	SELECT	d_year_key, d_year
		FROM	date
		WHERE	{results from queries Q4.3.5 }
	Q4.3.7	SELECT	s_city_key, s_city
		FROM	supplier
		WHERE	{results from queries Q4.3.5 }
	Q4.3.8	SELECT	p_brand_key, p_brand
		FROM	part
		WHERE	{results from queries Q4.3.5 }
Hadavi and fVSS	Q4.3.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_region = 'AMERICA'
	Q4.3.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_nation = 'UNITED STATES'
	Q3.2.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year = 1997 OR d_year = 1998
	Q4.3.4	SELECT	p_partkey
		FROM	part
		WHERE	p_category = 'MFGR#14'
	Q4.3.5	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year_key, s_city_key, p_brand_key
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND lo_custkey = c_custkey AND {results from queries Q4.3.1, Q4.3.2, Q4.3.3 and Q4.3.4 }
		GROUP BY	d_year_key, s_city_key, p_brand_key
	Q4.3.6	As query Q4.3.5 in Thompson	
	Q4.3.7	As query Q4.3.6 in Thompson	
	Q4.3.8	As query Q4.3.6 in Thompson	
bpVSS	Q4.3.1	SELECT	c_custkey
		FROM	customer
		WHERE	c_region = E('AMERICA')
	Q4.3.2	SELECT	s_suppkey
		FROM	supplier
		WHERE	s_nation = E('UNITED STATES')
	Q3.2.3	SELECT	d_datekey
		FROM	date
		WHERE	d_year = E(1997) OR d_year = E(1998)
	Q4.3.4	SELECT	p_partkey
		FROM	part
		WHERE	p_category = E('MFGR#14')
	Q4.3.5	SELECT	SUM(lo_revenue - lo_supplycost) as profit, d_year_key, d_year, s_city_key, s_city, p_brand_key, p_brand
		FROM	lineorder, date, customer, supplier, part
		WHERE	lo_orderdate = d_datekey AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey AND lo_custkey = c_custkey ANDD {results from queries Q4.3.1, Q4.3.2, Q4.3.3 and Q4.3.3 after filtering }
		GROUP BY	d_year_key, s_city_key, p_brand_key

Appendix B

Experimental Result Details

B.1 Share Volume

Table B.1: Thompson data volume

Locations	Types	Fact table	Dimension tables				Total (KB)
		Lineorder (KB)	Customer (KB)	Supplier (KB)	Part (KB)	Date (KB)	
CSP_1	Shares	472,802	2,606	770	17,006	246	493,431
	Signatures	235,114	2,097	600	12,664	199	250,673
	Total	707,916	4,703	1,370	29,670	445	744,104
CSP_2	Shares	472,802	2,606	770	17,006	246	493,431
	Signatures	286,596	2,161	620	13,366	198	302,941
	Total	759,398	4,767	1,390	30,372	444	796,371
CSP_3	Shares	472,802	2,606	770	17,006	246	493,431
	Signatures	286,559	2,160	619	13,360	198	302,896
	Total	759,361	4,766	1,389	30,366	444	796,326
CSP_4	Shares	472,802	2,606	770	17,006	246	493,431
	Signatures	286,705	2,163	620	13,357	198	303,043
	Total	759,507	4,769	1,390	30,363	444	796,473
CSP_5	Shares	472,802	2,606	770	17,006	246	493,431
	Signatures	285,938	2,128	611	13,237	198	302,112
	Total	758,740	4,734	1,381	30,243	444	795,542
Index server	Signatures	395,373	2,593	765	16,555	233	415,519
Total	Shares	2,364,010	13,030	3,852	85,030	1,230	2,467,153
	Signatures	1,776,285	13,302	3,833	82,539	1,225	1,877,183
	Total	4,140,295	26,332	7,685	167,569	2,455	4,344,336

Table B.2: Hadavi data volume

Locations	Types	Fact table	Dimension tables				Total (KB)
		Lineorder (KB)	Customer (KB)	Supplier (KB)	Part (KB)	Date (KB)	
CSP_1	Shares	449,264	2,545	768	16,882	244	469,704
CSP_2	Shares	470,822	2,545	753	16,611	241	490,972
CSP_3	Shares	470,826	2,545	752	16,609	241	490,974
CSP_4	Shares	470,832	2,545	753	16,610	241	490,981
CSP_5	Shares	470,829	2,545	753	16,609	241	490,977
Index server	Type-II indices	115,039	333	98	3,787	91	119,348
Total	Shares	2,332,573	12,725	83,321	1,209	101,034	2,433,607
	Type-II indices	115,039	333	98	3,787	91	119,348
	Total	2,447,612	13,058	3,877	87,108	1,301	2,552,955

Table B.3: bpVSS data volume

Locations	Types	Fact table	Dimension tables				Total (KB)
		Lineorder (KB)	Customer (KB)	Supplier (KB)	Part (KB)	Date (KB)	
CSP_1	Shares	373,600	2,660	790	14,734	202	391,986
	Signatures	143,237	2,063	590	10,769	149	156,808
	Total	516,837	4,723	1,380	25,503	351	548,794
CSP_2	Shares	373,600	2,660	790	14,734	202	391,986
	Signatures	143,237	2,063	590	10,769	149	156,808
	Total	516,837	4,723	1,380	25,503	351	548,794
CSP_3	Shares	373,600	2,660	790	14,734	202	391,986
	Signatures	143,237	2,063	590	10,769	149	156,808
	Total	516,837	4,723	1,380	25,503	351	548,794
CSP_4	Shares	373,600	2,660	790	14,734	202	391,986
	Signatures	143,237	2,063	590	10,769	149	156,808
	Total	516,837	4,723	1,380	25,503	351	548,794
CSP_5	Shares	373,600	2,660	790	14,734	202	391,986
	Signatures	143,237	2,063	590	10,769	149	156,808
	Total	516,837	4,723	1,380	25,503	351	548,794
Total	Shares	1,868,000	13,300	3,951	73,670	1,010	1,959,931
	Signatures	716,185	10,315	2,949	53,845	746	784,040
	Total	2,584,185	23,615	6,900	127,515	1,756	2,743,971

Table B.4: fVSS-I data volume

Locations	Types	Fact table	Dimension tables				Total (KB)
		Lineorder (KB)	Customer (KB)	Supplier (KB)	Part (KB)	Date (KB)	
CSP_1	Shares	285,134	1,578	468	10,284	149	297,613
	Signatures	55,323	236	75	1,654	18	57,306
	Type-III indices	84,392	0	0	0	0	84,392
	Total	424,849	1,814	543	11,938	167	439,310
CSP_2	Shares	285,104	1,800	467	10,284	149	297,804
	Signatures	55,422	236	75	1,654	18	57,406
	Type-III indices	84,391	0	0	0	0	84,391
	Total	424,917	2,036	543	11,938	167	439,601
CSP_3	Shares	283,680	1,574	466	10,286	150	296,156
	Signatures	55,401	236	75	1,654	18	57,385
	Type-III indices	84,134	0	0	0	0	84,134
	Total	423,215	1,810	541	11,940	169	437,675
CSP_4	Shares	285,094	1,578	467	10,286	149	297,574
	Signatures	55,454	236	75	1,654	18	57,438
	Type-III indices	84,391	0	0	0	0	84,391
	Total	424,939	1,814	542	11,940	167	439,402
CSP_5	Shares	284,799	1,583	469	10,330	149	297,329
	Signatures	55,345	236	75	1,654	18	57,328
	Type-III indices	84,071	0	0	0	0	84,071
	Total	424,215	1,819	544	11,984	167	438,729
Index server	Type-I indices	131,251	477	159	3,958	53	135,898
	Type-II indices	115,039	333	98	3,787	91	119,348
	Total	246,290	810	258	7,745	144	255,247
Total	Shares	1,423,811	8,113	2,337	51,470	746	1,486,477
	Signatures	276,945	1,178	376	8,270	92	286,862
	Type-III indices	131,251	477	159	3,958	53	135,898
	Type-I indices	115,039	333	98	3,787	91	119,348
	Type-II indices	421,379	0	0	0	0	421,379
	Total	2,368,425	10,102	2,970	67,485	983	2,449,964

Table B.5: fVSS-II data volume

Locations	Types	Fact table	Dimension tables				Total (KB)
		Lineorder (KB)	Customer (KB)	Supplier (KB)	Part (KB)	Date (KB)	
CSP_1	Shares	379,015	2,092	618	13,609	198	395,532
	Signatures	55,323	236	75	1,654	18	57,306
	Type-III indices	93,732	0	0	0	0	93,732
	Total	528,070	2,328	693	15,263	216	546,570
CSP_2	Shares	379,160	2,091	618	13,609	198	395,675
	Signatures	55,422	236	75	1,654	18	57,406
	Type-III indices	93,767	0	0	0	0	93,767
	Total	528,349	2,327	693	15,263	216	546,848
CSP_3	Shares	284,357	1,569	465	10,208	150	296,749
	Signatures	55,401	236	75	1,654	18	57,385
	Type-III indices	70,323	0	0	0	0	70,323
	Total	410,081	1,805	540	11,862	169	424,457
CSP_4	Shares	188,937	1,049	310	6,803	97	197,196
	Signatures	55,454	236	75	1,654	18	57,438
	Type-III indices	45,597	0	0	0	0	45,597
	Total	289,988	1,285	385	8,457	116	300,231
CSP_5	Shares	189,728	1,048	311	6,808	97	197,992
	Signatures	55,345	236	75	1,654	18	57,328
	Type-III indices	46,922	0	0	0	0	46,922
	Total	291,995	1,284	386	8,462	116	302,242
Index server	Type-I indices	131,251	477	159	3,958	53	135,898
	Type-II indices	115,039	333	98	3,787	91	119,348
	Total	246,290	810	258	7,745	144	255,247
Total	Shares	1,421,197	7,849	2,321	51,037	740	1,483,144
	Signatures	276,945	1,178	376	8,270	92	286,862
	Type-III indices	131,251	477	159	3,958	53	135,898
	Type-I indices	115,039	333	98	3,787	91	119,348
	Type-II indices	350,341	0	0	0	0	350,341
	Total	2,294,773	9,838	2,955	67,052	976	2,375,594

B.2 Data Sharing Time

Table B.6: Thompson sharing time

Locations	Processes	Fact table	Dimension tables				Total (s)
		Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	
User	Encrypt data	2,576.07	32.75	8.97	174.35	2.38	2,794.52
CSP_1	Load shares & signatures	348.37	1.72	0.52	11.08	0.14	361.83
CSP_2	Load shares & signatures	361.85	1.82	0.51	10.81	0.14	375.13
CSP_3	Load shares & signatures	362.39	1.79	0.51	10.88	0.14	375.71
CSP_4	Load shares & signatures	357.44	1.65	0.51	10.84	0.14	370.58
CSP_5	Load shares & signatures	356.99	1.62	0.54	10.64	0.16	369.95
Index server	Build signature trees	322.68	1.52	0.50	10.14	0.14	334.98
Total	Encrypt data	2,576.07	32.75	8.97	174.35	2.38	2,794.52
	Load shares & signatures	362.39	1.82	0.54	11.08	0.16	375.98
	Build signature trees	322.68	1.52	0.50	10.14	0.14	334.98
	Total	2,938.46	34.57	9.51	185.43	2.54	3,170.51

Table B.7: Hadavi sharing time

Locations	Processes	Fact table	Dimension tables				Total
		Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	(s)
User	Encrypt data	1,159.31	15.58	4.51	84.62	1.06	1,265.08
CSP_1	Load shares	311.29	1.51	0.50	10.53	0.13	323.97
CSP_2	Load shares	329.94	1.50	0.50	10.32	0.13	342.38
CSP_3	Load shares	344.91	1.51	0.53	10.55	0.13	357.64
CSP_4	Load shares	313.20	1.48	0.50	10.35	0.15	325.68
CSP_5	Load shares	313.21	1.54	0.50	10.35	0.15	325.76
Index server	Build Type-II indices	36.51	1.98	0.64	56.66	0.20	95.99
Total	Encrypt data	1,159.31	15.58	4.51	84.62	1.06	1,265.08
	Load shares	36.51	1.98	0.64	56.66	0.20	95.99
	Build Type-II indices	344.91	1.54	0.53	10.55	0.15	357.69
	Total	1,504.23	17.56	5.16	141.28	1.26	1,669.47

Table B.8: bpVSS sharing time

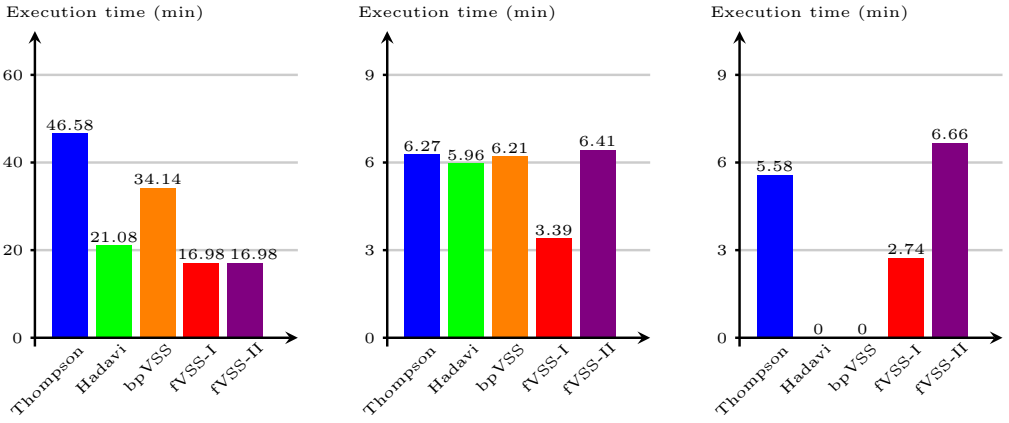
Locations	Processes	Fact table	Dimension tables				Total
		Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	(s)
User	Encrypt data	1,861.90	27.83	7.78	149.08	1.72	2,048.31
CSP_1	Load shares & signatures	339.83	1.60	0.55	10.56	0.14	352.67
CSP_2	Load shares & signatures	334.93	1.63	0.52	10.50	0.16	347.73
CSP_3	Load shares & signatures	358.76	1.69	0.55	10.72	0.15	371.87
CSP_4	Load shares & signatures	359.22	1.58	0.55	10.54	0.14	372.03
CSP_5	Load shares & signatures	345.87	1.57	0.52	11.00	0.14	359.11
Total	Encrypt data	1,861.90	27.83	7.78	149.08	1.72	2,048.31
	Load shares & signatures	359.22	1.69	0.55	11.00	0.16	372.62
	Total	2,221.13	29.51	8.34	160.08	1.87	2,420.93

Table B.9: fVSS-I sharing time

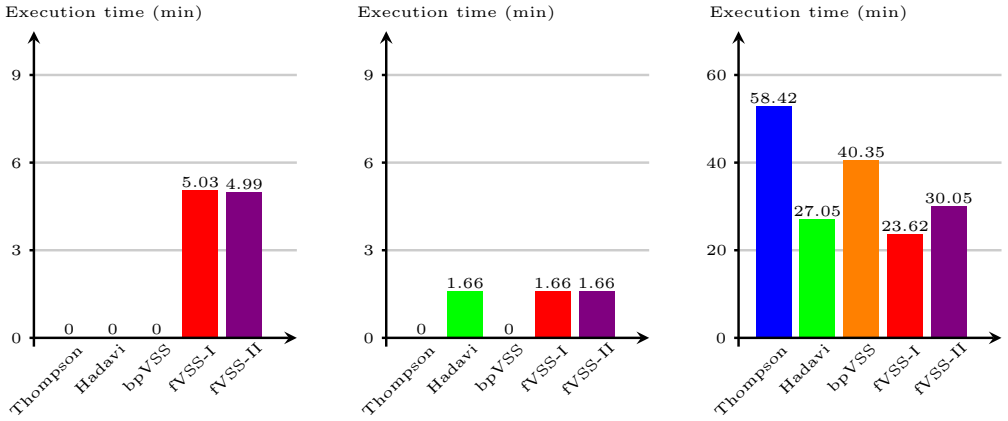
Locations	Processes	Fact table	Dimension tables				Total
		Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	(s)
User	Encrypt data	949.44	9.95	2.88	56.12	0.70	1,019.08
CSP_1	Load shares & type-III indices	193.20	1.01	0.33	6.25	0.08	200.86
	Build signature trees	153.29	0.46	0.14	3.46	0.07	157.41
	Total	346.49	1.46	0.47	9.71	0.14	358.28
CSP_2	Load shares & type-III indices	194.86	1.01	0.29	6.11	0.10	202.37
	Build signature trees	157.12	0.44	0.12	3.49	0.06	161.23
	Total	351.98	1.45	0.42	9.60	0.16	363.60
CSP_3	Load shares & type-III indices	195.39	1.02	0.33	6.12	0.08	202.94
	Build signature trees	156.37	0.58	0.15	3.62	0.06	160.78
	Total	351.76	1.60	0.48	9.74	0.14	363.72
CSP_4	Load shares & type-III indices	192.55	0.95	0.34	6.41	0.08	200.33
	Build signature trees	159.88	0.45	0.14	3.53	0.06	164.06
	Total	352.43	1.40	0.48	9.94	0.14	364.39
CSP_5	Load shares & type-III indices	194.57	1.00	0.30	6.10	0.08	202.05
	Build signature trees	158.08	0.44	0.14	3.52	0.06	162.25
	Total	352.65	1.44	0.44	9.62	0.14	364.30
Index server	Build Type-I indices	290.17	1.44	0.50	9.64	0.12	301.87
	Build Type-II indices	36.51	1.98	0.64	56.66	0.20	95.99
	Total	326.68	3.42	1.14	66.30	0.32	397.86
Total	Encryption	949.44	9.95	2.88	56.12	0.70	1,019.08
	Load shares & type-III indices	195.39	1.02	0.34	6.41	0.10	203.27
	Build signature trees	159.88	0.58	0.15	3.62	0.07	164.30
	Build Type-I indices	290.17	1.44	0.50	9.64	0.12	301.87
	Build Type-II indices	36.51	1.98	0.64	56.66	0.20	95.99
	Total	1,302.08	13.37	4.02	122.42	1.02	1,442.91

TABLE B.10: fVSS-II sharing time

Locations	Processes	Fact table	Dimension tables				Total (s)
		Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	
User	Encrypt data	949.44	9.95	2.88	56.12	0.70	1,019.08
CSP_1	Load shares & type-III indices	249.04	1.58	0.38	8.06	0.11	259.18
	Build signature trees	155.24	0.51	0.16	3.47	0.09	159.47
	Total	404.28	2.09	0.54	11.54	0.20	418.64
CSP_2	Load shares & type-III indices	258.37	1.61	0.42	7.91	0.10	268.42
	Build signature trees	154.27	0.48	0.18	3.58	0.11	158.62
	Total	412.64	2.10	0.60	11.50	0.21	427.04
CSP_3	Load shares & type-III indices	369.98	2.29	0.61	11.34	0.15	384.36
	Build signature trees	297.69	0.99	0.35	6.82	0.21	306.05
	Total	667.67	3.28	0.95	18.16	0.36	690.42
CSP_4	Load shares & type-III indices	322.71	2.08	0.47	9.80	0.13	335.20
	Build signature trees	384.55	1.22	0.33	8.71	0.16	394.96
	Total	707.25	3.31	0.80	18.51	0.30	730.17
CSP_5	Load shares & type-III indices	317.21	2.04	0.48	10.25	0.13	330.11
	Build signature trees	389.12	1.21	0.32	8.62	0.15	399.42
	Total	706.33	3.25	0.80	18.87	0.28	729.53
Index server	Build Type-I indices	287.63	1.88	0.47	9.23	11.70	299.33
	Build Type-II indices	36.51	1.98	0.64	56.66	0.20	95.99
	Total	324.14	3.86	1.11	65.88	0.32	395.32
Total	Encryption	949.44	9.95	2.88	56.12	0.70	1,019.08
	Load shares & type-III indices	369.98	2.29	0.61	11.34	0.15	384.36
	Build signature trees	389.12	1.22	0.35	8.71	0.21	399.60
	Build Type-I indices	287.63	1.88	0.47	9.23	0.12	299.33
	Build Type-II indices	36.51	1.98	0.64	56.66	0.20	95.99
	Total	1,656.69	13.81	3.99	122.00	1.06	1,797.55



(a) Encryption (b) Loading shares and/or signatures (c) Building signature trees



(d) Building Type-I indices (e) Building Type-II indices (f) Total

FIGURE B.1: Data sharing time

B.3 Workload Response Time

B.3.1 Data Reconstruction Time

Table B.11: Thompson reconstruction time

Locations	Fact table	Dimension tables				Total
	Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	(s)
User	733.71	15.65	3.52	65.54	0.48	818.89
CSP_2	1,247.97	0.60	0.33	35.76	0.02	1,284.68
CSP_3	1,254.07	0.59	0.33	35.65	0.02	1,290.65
CSP_4	1,247.42	0.61	0.26	38.31	0.03	1,286.62
CSP_5	1,247.53	0.61	0.26	35.44	0.03	1,283.87
Index server	1,232.23	0.54	0.29	30.59	0.06	1,263.71
Total	1,987.77	16.26	3.84	103.85	0.54	2,109.55

Table B.12: Hadavi reconstruction time

Locations	Fact table	Dimension tables				Total
	Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	(s)
User	356.84	6.08	1.75	31.17	0.37	396.22
CSP_2	1,055.18	0.50	0.07	21.71	0.02	1,077.48
CSP_3	1,052.21	0.52	0.08	21.42	0.02	1,074.25
CSP_4	1,055.53	0.50	0.07	21.53	0.02	1,077.65
CSP_5	1,051.87	0.46	0.07	21.32	0.02	1,073.75
Total	1,412.38	12.16	3.51	62.33	0.74	1,473.87

Table B.13: bpVSS reconstruction time

Locations	Fact table	Dimension tables				Total
	Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	(s)
User	329.90	7.37	1.57	29.67	0.40	368.90
CSP_2	1,616.28	0.73	0.09	29.56	0.02	1,646.69
CSP_3	1,602.02	0.75	0.09	31.51	0.02	1,634.39
CSP_4	1,614.81	0.74	0.09	29.68	0.02	1,645.34
CSP_5	1,607.27	0.73	0.10	29.70	0.02	1,637.82
Total	1,946.18	14.74	3.13	61.18	0.79	2,015.59

Table B.14: fVSS-I reconstruction time

Locations	Fact table	Dimension tables				Total
	Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	(s)
User	161.07	3.93	0.14	0.18	0.10	165.42
CSP_2	904.82	0.24	0.07	0.06	0.03	905.22
CSP_3	903.20	0.26	0.07	0.06	0.03	903.62
CSP_4	911.17	0.29	0.09	0.06	0.03	911.65
CSP_5	907.13	0.29	0.09	0.06	0.03	907.60
Index server	575.83	0.19	0.01	0.01	0.00	576.05
Total	1,072.24	4.22	0.23	0.24	0.13	1,077.06

Table B.15: fVSS-II reconstruction time

Locations	Fact table	Dimension tables				Total
	Lineorder (s)	Customer (s)	Supplier (s)	Part (s)	Date (s)	(s)
User	153.52	2.75	0.19	4.52	0.08	161.05
CSP_2	1,291.81	0.37	0.14	3.20	0.07	1,295.59
CSP_3	1,878.50	0.54	0.22	4.76	0.10	1,884.12
CSP_4	1,639.80	0.56	0.31	4.56	0.16	1,645.39
CSP_5	1,648.35	0.56	0.20	5.25	0.18	1,654.55
Index server	566.17	0.19	0.03	0.69	0.00	567.07
Total	2,032.02	3.31	0.50	9.76	0.26	2,045.17

B.3.2 Data Access Time

Table B.16: Thompson data access time

Queries		User	CSP_2	CSP_3	CSP_4	CSP_5	Index server	Total
		Decrypt data (s)	Access shares (s)	Access shares (s)	Access shares (s)	Access shares (s)	Access signature trees (s)	(s)
Q1	Q1.1	17.2801	127.8989	113.1091	146.9611	138.6160	95.5987	180.1920
	Q1.2	1.7156	4.6593	5.0529	4.6353	4.7524	3.1173	8.3521
	Q1.3	0.6936	4.4746	5.6195	4.7684	5.2761	3.8801	6.9532
	Total	19.6892	137.0328	123.7815	156.3649	148.6444	102.5961	194.2287
Q2	Q2.1	0.0120	25.9832	23.8958	21.8973	23.0974	15.5139	25.9863
	Q2.2	0.0016	25.0925	29.0487	24.4765	27.9571	17.7817	29.0498
	Q2.3	0.0007	24.1906	23.8897	23.7014	25.3541	17.5396	25.3544
	Total	0.0143	75.2663	76.8342	70.0751	76.4086	50.8352	76.8386
Q3	Q3.1	0.0176	11.6086	11.3254	14.1927	10.9761	7.1984	14.1966
	Q3.2	0.0676	7.3828	7.3106	8.6992	8.4667	4.9499	8.6993
	Q3.3	0.0049	7.3480	6.1341	9.1597	5.3705	3.9169	9.1601
	Q3.4	0.0021	4.6921	4.8412	5.2181	4.4099	3.7205	5.2183
	Total	0.0922	31.0315	29.6114	37.2698	29.2233	19.7858	37.2743
Q4	Q4.1	0.0038	10.4326	10.0083	12.6090	9.2610	6.4161	12.6114
	Q4.2	0.0082	21.8089	21.8978	22.5899	22.9805	12.2171	22.9843
	Q4.3	0.0325	25.8927	18.4484	17.4291	19.3233	11.3032	25.9352
	Total	0.0445	58.1342	50.3545	52.6280	51.5649	29.9364	58.1828
Total		19.7305	602.9296	561.1632	632.6757	611.6823	406.3070	741.2174

Table B.17: Hadavi data access time

Queries		User	CSP_2	CSP_3	CSP_4	CSP_5	Index server	Total
		Decrypt data (s)	Access shares (s)	Access shares (s)	Access shares (s)	Access shares (s)	Access Type-II indices (s)	(s)
Q1	Q1.1	0.7340	19.7280	20.9339	21.2110	19.5524	2.1902	24.8127
	Q1.2	0.0157	4.5882	3.7783	4.3778	4.4286	1.8810	6.4994
	Q1.3	0.0040	5.0797	5.3273	6.7241	4.7854	1.3752	8.1069
	Total	0.7537	29.3959	30.0395	32.3128	28.7665	5.4464	39.2085
Q2	Q2.1	0.0030	13.0164	19.9966	12.7169	14.1037	0.0022	20.0006
	Q2.2	0.0008	15.7879	16.3418	15.6419	16.3237	0.0115	16.3538
	Q2.3	0.0002	15.7256	16.3776	15.8095	16.3404	0.0027	16.3805
	Total	0.0040	44.5300	52.7160	44.1683	46.7678	0.0164	52.7349
Q3	Q3.1	0.0071	6.8024	6.8001	6.7825	6.9291	0.0009	6.9337
	Q3.2	0.0335	5.3929	5.8403	5.5392	3.5696	0.0057	5.8804
	Q3.3	0.0023	4.1198	6.4723	4.2042	6.5502	0.0078	6.5584
	Q3.4	0.0001	2.8248	3.5519	3.3417	2.9176	0.0057	3.5577
	Total	0.0429	19.1399	22.6647	19.8676	19.9665	0.0202	22.7234
Q4	Q4.1	0.0014	5.0670	5.9965	5.8727	5.2558	0.0654	6.0636
	Q4.2	0.0039	13.3410	13.2136	13.3267	14.6934	0.0057	14.7016
	Q4.3	0.0149	10.3090	10.6649	10.0424	12.1207	0.0045	12.1489
	Total	0.0201	28.7170	29.8751	29.2419	32.0699	0.0756	32.1732
Total		0.7894	243.5656	270.5905	251.1811	255.1414	11.1170	294.8381

TABLE B.18: bpVSS data access time

Queries		User Decrypt data (s)	CSP_2 Access & verify shares (s)	CSP_3 Access & verify shares (s)	CSP_4 Access & verify shares (s)	CSP_5 Access & verify shares (s)	Total (s)
Q1	Q1.1	0.5173	56.7908	54.7920	54.6009	58.3906	59.3854
	Q1.2	0.0431	22.2140	25.1041	20.1321	15.3784	25.1870
	Q1.3	0.0142	20.1569	21.1005	21.1931	16.7587	21.2204
	Total	0.5746	99.1617	100.9966	95.9261	90.5277	102.1016
Q2	Q2.1	0.0031	36.6664	34.2652	35.4869	34.5972	36.6723
	Q2.2	0.0007	37.1857	38.6276	41.2461	39.3168	41.2475
	Q2.3	0.0002	37.5234	36.7418	38.3598	36.9040	38.3602
	Total	0.0040	111.3755	109.6346	115.0928	110.8180	115.1005
Q3	Q3.1	0.0062	62.2928	63.1110	61.0772	63.3041	63.3160
	Q3.2	0.0326	34.0925	34.5213	34.0910	34.5245	34.5873
	Q3.3	0.0019	31.9301	31.2374	31.9544	31.9040	31.9580
	Q3.4	0.0008	55.8238	53.0440	58.2558	57.7601	58.2574
	Total	0.0416	184.1393	181.9137	185.3784	187.4927	187.5727
Q4	Q4.1	0.0013	28.4288	28.4384	27.8212	28.2522	28.4410
	Q4.2	0.0038	42.0429	40.6533	40.6599	40.5496	42.0502
	Q4.3	0.0144	35.9292	35.3184	37.1975	37.3328	37.3605
	Total	0.0195	106.4009	104.4101	105.6787	106.1346	106.4385
Total		0.6397	1,002.1548	993.9099	1,004.1520	989.9458	1,029.2565

TABLE B.19: fVSS-I data access time

Queries		User Decrypt data (s)	CSP_2 Access shares (s)	CSP_3 Access shares (s)	CSP_4 Access shares (s)	CSP_5 Access shares (s)	Index server Access Type-I indices (s)	Index server Access Type-II indices (s)	Total (s)
Q1	Q1.1	0.0001	16.4278	16.4983	15.6418	14.9424	3.5073	2.1902	18.6886
	Q1.2	0.0001	14.2765	15.1625	13.9727	15.6676	2.2837	1.8810	17.5486
	Q1.3	0.0001	18.0176	12.8647	13.1031	13.6331	3.9304	1.3752	19.3929
	Total	0.0002	48.7219	44.5256	42.7176	44.2431	9.7213	5.4464	54.1686
Q2	Q2.1	0.0016	15.7137	15.3239	15.6721	14.5669	13.1101	0.0022	15.7176
	Q2.2	0.0004	15.4171	15.7343	15.3091	13.9725	21.7381	21.7381	43.4771
	Q2.3	0.0002	15.4217	15.8592	16.2880	14.7842	20.8768	0.0027	20.8798
	Total	0.0022	46.5525	46.9174	47.2692	43.3237	55.7249	21.7430	77.4708
Q3	Q3.1	0.0009	11.3659	11.3870	11.7426	11.4866	10.9713	0.0009	11.7452
	Q3.2	0.0034	10.8408	11.6850	11.7997	12.6255	3.3359	0.0057	12.6377
	Q3.3	0.0003	9.8745	10.0267	10.0551	10.1589	2.7382	0.0078	10.1673
	Q3.4	0.0001	9.8846	9.7007	10.0297	10.2373	2.6706	0.0057	10.2433
	Total	0.0048	41.9658	42.7994	43.6271	44.5084	19.7159	0.0202	44.5376
Q4	Q4.1	0.0003	10.3930	10.8948	11.3290	13.3271	2.0828	0.0654	13.3931
	Q4.2	0.0007	15.9068	15.9031	16.1770	16.5240	1.6850	0.0057	16.5310
	Q4.3	0.0040	16.6551	14.9444	15.3768	16.0783	1.5348	0.0045	16.6671
	Total	0.0049	42.9549	41.7423	42.8828	45.9294	5.3026	0.0756	46.0145
Total		0.0462	360.3903	351.9692	352.9933	356.0092	180.9296	54.5702	449.2809

TABLE B.20: fVSS-II data access time

Queries		User Decrypt data (s)	CSP_2 Access shares (s)	CSP_3 Access shares (s)	CSP_4 Access shares (s)	CSP_5 Access shares (s)	Index server Access Type-I indices (s)	Index server Access Type-II indices (s)	Total (s)
Q1	Q1.1	0.0001	20.7943	33.0436	31.3821	37.6631	3.7317	2.1902	39.8535
	Q1.2	0.0001	24.4422	26.8189	26.9480	24.8598	3.1949	1.8810	28.8291
	Q1.3	0.0001	17.0687	24.9111	24.5108	26.1218	2.7800	1.3752	27.4971
	Total	0.0004	62.3053	84.7735	82.8410	88.6447	9.7066	5.4464	94.0914
Q2	Q2.1	0.0016	20.8171	29.9384	21.6831	23.0507	13.3928	0.0022	29.9436
	Q2.2	0.0004	23.8626	29.3827	21.5281	21.6382	21.5350	0.0115	29.3951
	Q2.3	0.0002	23.5494	31.6956	21.5232	23.0938	21.2979	0.0027	31.6986
	Total	0.0022	68.2291	91.0168	64.7343	67.7828	56.2257	0.0164	91.0373
Q3	Q3.1	0.0009	17.8471	21.7171	17.7649	18.3581	11.1620	0.0009	21.7197
	Q3.2	0.0034	14.5372	19.6855	18.6867	18.6317	3.3634	0.0057	19.6978
	Q3.3	0.0003	14.5343	19.2998	16.7587	16.5423	2.8244	0.0078	19.3082
	Q3.4	0.0001	12.6461	18.1888	15.4892	16.5361	2.7288	0.0057	18.1948
	Total	0.0047	59.5647	78.8912	68.6994	70.0683	20.0786	0.0202	78.9206
Q4	Q4.1	0.0003	16.5596	21.8593	18.2043	17.9444	3.3230	0.0654	21.9251
	Q4.2	0.0007	21.1460	30.5313	25.4602	25.1506	1.8805	0.0057	30.5383
	Q4.3	0.0039	20.5343	30.3080	25.8061	25.4868	1.6070	0.0045	30.3202
	Total	0.0049	58.2400	82.6986	69.4706	68.5818	6.8105	0.0756	82.7836
Total		0.0120	496.6782	674.7603	571.4907	590.1550	185.6430	11.1170	695.7541

B.4 Transferred Data Volume

Table B.21: Thompson transferred data volume

Queries		CSP_2		CSP_3		CSP_4		CSP_5		Index server	Total
		Shares (KB)	Signatures (KB)	Shares (KB)	Signatures (KB)	Shares (KB)	Signatures (KB)	Shares (KB)	Signatures (KB)	Signatures (KB)	(KB)
Q1	Q1.1	17,643	17,643	17,194	17,194	15,895	15,895	15,068	15,068	15,375	146,976
	Q1.2	1,542	1,542	1,503	1,503	1,390	1,390	1,318	1,318	1,340	12,846
	Q1.3	370	370	361	361	334	334	318	318	329	3,093
	Total	19,555	19,555	19,058	19,058	17,619	17,619	16,704	16,704	17,044	162,915
Q2	Q2.1	3,273	3,273	3,205	3,205	2,952	2,952	2,912	2,912	3,678	28,362
	Q2.2	3,605	3,605	3,603	3,603	3,603	3,603	3,603	3,603	3,676	32,502
	Q2.3	3,604	3,604	3,602	3,602	3,602	3,602	3,602	3,602	3,675	32,497
	Total	10,483	10,483	10,409	10,409	10,157	10,157	10,116	10,116	11,030	93,361
Q3	Q3.1	1,016	1,016	1,016	1,016	1,021	1,021	1,007	1,007	1,014	9,133
	Q3.2	1,053	1,053	1,053	1,053	1,056	1,056	1,043	1,043	1,048	9,456
	Q3.3	1,286	1,286	1,286	1,286	1,286	1,286	1,286	1,286	1,286	11,576
	Q3.4	1,286	1,286	1,286	1,286	1,286	1,286	1,286	1,286	1,286	11,573
	Total	4,640	4,640	4,641	4,641	4,648	4,648	4,623	4,623	4,634	41,738
Q4	Q4.1	1,014	1,014	1,015	1,015	1,019	1,019	1,006	1,006	1,013	9,121
	Q4.2	1,354	1,354	1,354	1,354	1,360	1,360	1,342	1,342	1,352	12,172
	Q4.3	1,372	1,372	1,373	1,373	1,379	1,379	1,361	1,361	1,376	12,345
	Total	3,740	3,740	3,741	3,741	3,758	3,758	3,709	3,709	3,741	33,638
Total		76,837	76,837	75,700	75,700	72,364	72,364	70,303	70,303	72,897	663,305

Table B.22: Hadavi transferred data volume

Queries		CSP_2	CSP_3	CSP_4	CSP_5	Index server	Total
		Shares (bytes)	Shares (bytes)	Shares (bytes)	Shares (bytes)	Type-II indices (bytes)	(bytes)
Q1	Q1.1	3,109,261	3,034,678	2,909,321	2,716,608	7,474,205	19,244,073
	Q1.2	73,567	72,303	70,314	67,018	2,050,163	2,333,365
	Q1.3	15,722	15,470	15,021	14,328	2,048,063	2,108,604
	Total	3,198,550	3,122,451	2,994,656	2,797,954	11,572,431	23,686,042
Q2	Q2.1	3,631	3,600	3,577	3,528	50,855	65,191
	Q2.2	822	813	805	789	16,361	19,590
	Q2.3	144	141	136	128	8,827	9,376
	Total	4,597	4,554	4,518	4,445	76,043	94,157
Q3	Q3.1	2,574	2,573	2,569	2,561	53,330	63,607
	Q3.2	13,990	13,914	13,867	13,881	24,943	80,595
	Q3.3	400	399	395	388	18,809	20,391
	Q3.4	50	49	49	48	1,521	1,717
	Total	17,014	16,935	16,880	16,878	98,603	166,310
Q4	Q4.1	1,109	1,104	1,099	1,095	471,718	476,125
	Q4.2	2,739	2,726	2,703	2,701	477,550	488,419
	Q4.3	19,763	18,986	18,233	18,162	78,901	154,045
	Total	23,611	22,816	22,035	21,958	1,028,169	1,118,589
Total		6,487,544	6,333,512	6,076,178	5,682,470	25,550,492	50,130,196

TABLE B.23: bpVSS transferred data volume

Queries		CSP_2 Shares (KB)	CSP_3 Shares (KB)	CSP_4 Shares (KB)	CSP_5 Shares (KB)	Total (KB)
Q1	Q1.1	1,369	1,341	1,340	1,341	5,392
	Q1.2	42	39	37	38	157
	Q1.3	15	8	9	9	41
	Total	1,427	1,389	1,386	1,388	5,590
Q2	Q2.1	342	428	341	171	1,283
	Q2.2	17	642	370	709	1,738
	Q2.3	128	71	79	94	371
	Total	486	1,141	790	974	3,391
Q3	Q3.1	86	86	86	86	343
	Q3.2	44	44	44	44	175
	Q3.3	19	19	19	19	77
	Q3.4	16	3	3	3	26
	Total	165	152	152	152	620
Q4	Q4.1	462	462	462	462	1,847
	Q4.2	470	470	470	470	1,879
	Q4.3	127	127	127	127	508
	Total	1,059	1,059	1,058	1,059	4,235
Total		6,273	7,480	6,773	7,145	27,671

TABLE B.24: fVSS-I transferred data volume

Queries		CSP_2 Shares (bytes)	CSP_3 Shares (bytes)	CSP_4 Shares (bytes)	CSP_5 Shares (bytes)	Index server Type-I indices (bytes)	Index server Type-II indices (bytes)	Total (bytes)
Q1	Q1.1	15	15	16	15	36	7,474,205	7,474,302
	Q1.2	11	15	13	11	30	2,050,163	2,050,243
	Q1.3	13	13	14	12	29	2,048,063	2,048,144
	Total	39	43	43	38	95	11,572,431	11,572,689
Q2	Q2.1	5,522	5,557	5,142	5,032	13,368	50,855	85,476
	Q2.2	1,211	1,203	1,097	1,058	2,900	16,361	23,830
	Q2.3	193	204	132	137	360	8,827	9,853
	Total	6,926	6,964	6,371	6,227	16,628	76,043	119,159
Q3	Q3.1	3,517	3,525	3,182	3,078	6,585	53,330	73,217
	Q3.2	13,857	13,898	9,792	9,103	27,003	24,943	98,596
	Q3.3	504	429	353	310	1,092	18,809	21,497
	Q3.4	28	35	73	40	140	1,521	1,837
	Total	17,906	17,887	13,400	12,531	34,820	98,603	195,147
Q4	Q4.1	897	898	676	637	3	471,718	474,829
	Q4.2	1,922	1,961	2,408	2,313	4	477,550	486,158
	Q4.3	2,212	2,265	12,226	11,824	7	78,901	107,435
	Total	5,031	5,124	15,310	14,774	14	1,028,169	1,068,422
Total		59,804	60,036	70,248	67,140	103,114	25,550,492	25,910,834

TABLE B.25: fVSS-II transferred data volume

Queries		CSP_2 Shares (bytes)	CSP_3 Shares (bytes)	CSP_4 Shares (bytes)	CSP_5 Shares (bytes)	Index server Type-I indices (bytes)	Index server Type-II indices (bytes)	Total (bytes)
Q1	Q1.1	12	15	16	15	23	7,474,205	7,474,286
	Q1.2	15	15	13	11	20	2,050,163	2,050,237
	Q1.3	15	15	14	12	18	2,048,063	2,048,137
	Total	42	45	43	38	61	11,572,431	11,572,660
Q2	Q2.1	5,658	5,350	4,526	3,946	13,368	50,855	83,703
	Q2.2	1,234	1,173	974	850	2,900	16,361	23,492
	Q2.3	202	194	118	115	360	8,827	9,816
	Total	7,094	6,717	5,618	4,911	16,628	76,043	117,011
Q3	Q3.1	3,528	3,527	3,155	3,050	6,584	53,330	73,174
	Q3.2	17,098	13,605	3,197	3,245	27,003	24,943	89,091
	Q3.3	578	362	49	98	1,092	18,809	20,988
	Q3.4	164	73	0	0	140	1,521	1,898
	Total	21,368	17,567	6,401	6,393	34,819	98,603	185,151
Q4	Q4.1	894	896	675	639	2	471,718	474,824
	Q4.2	2,355	2,330	1,670	1,530	4	477,550	485,439
	Q4.3	16,707	11,911	416	453	7	78,901	108,395
	Total	19,956	15,137	2,761	2,622	13	1,028,169	1,068,658
Total		96,920	78,932	29,646	27,928	103,042	25,550,492	25,886,960

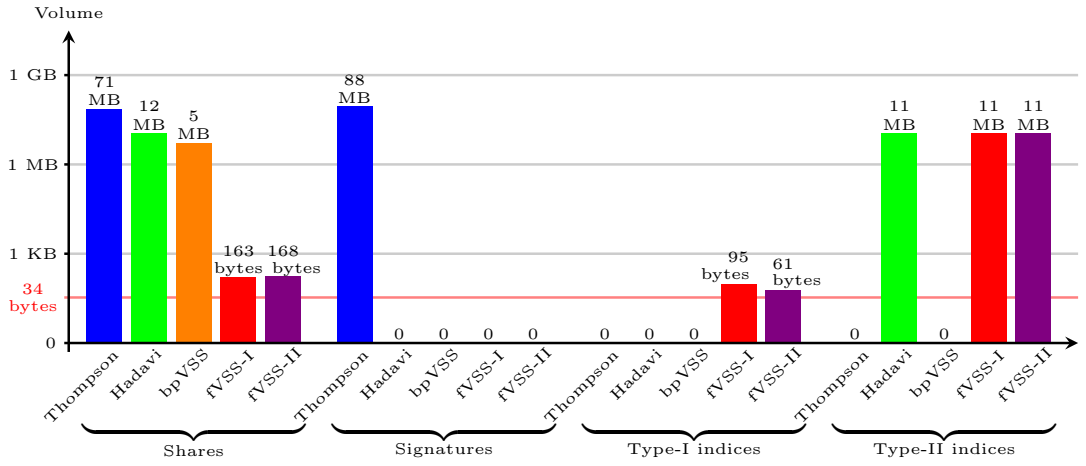


FIGURE B.2: Q1 result size

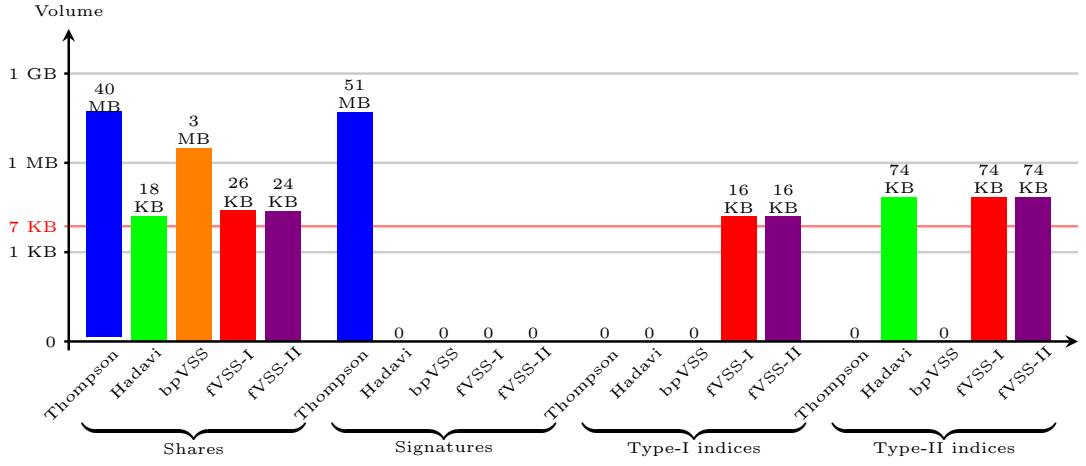


FIGURE B.3: Q2 result size

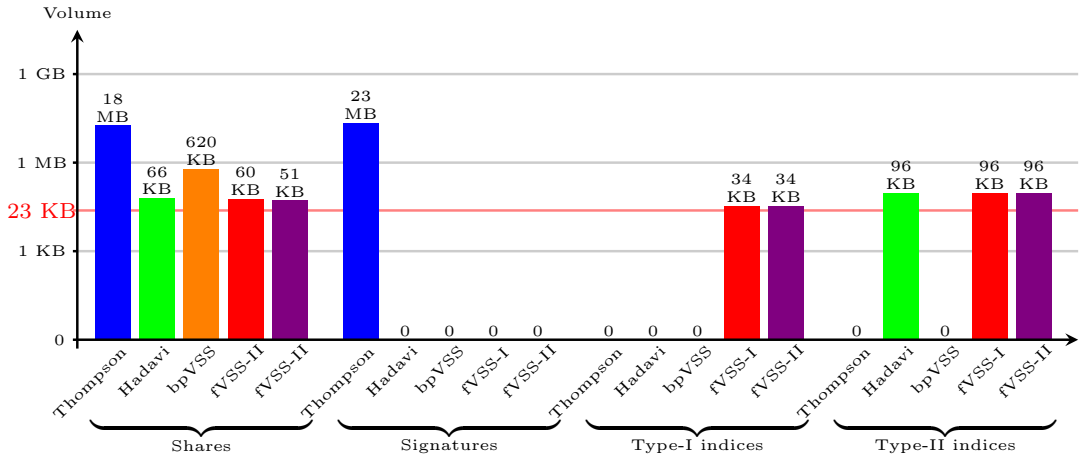


FIGURE B.4: Q3 result size

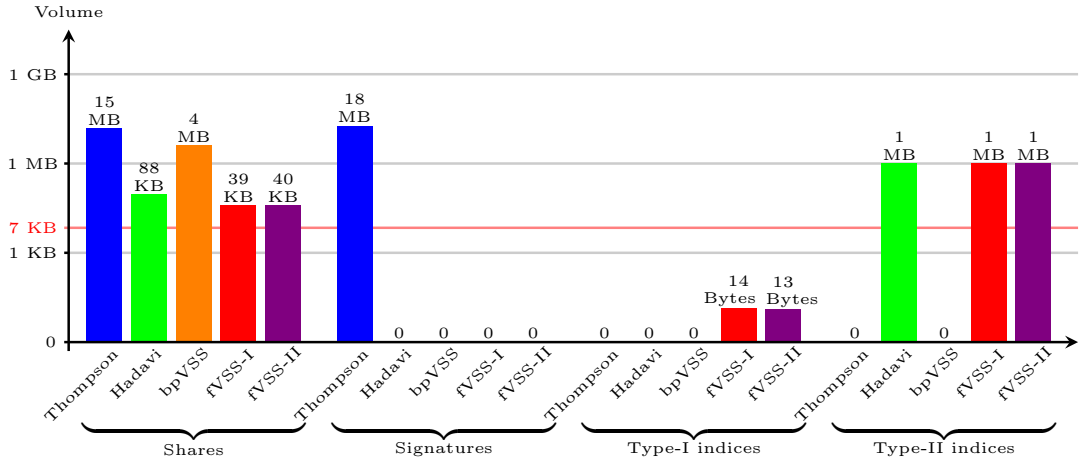


FIGURE B.5: Q4 result size

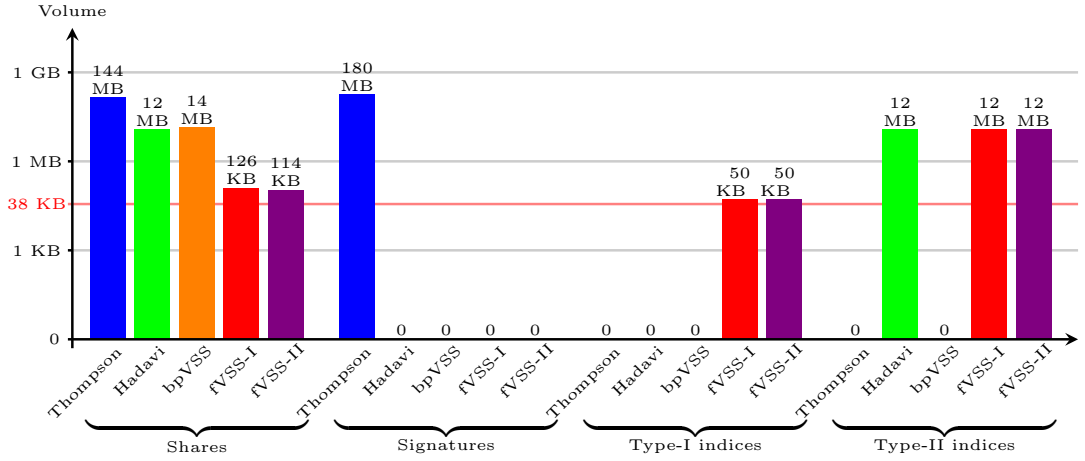


FIGURE B.6: Whole workload result size

B.5 Monetary Costs

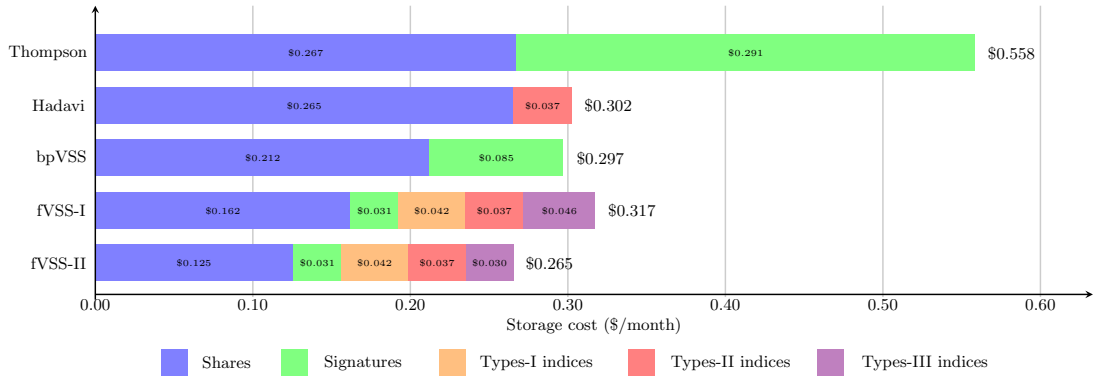


FIGURE B.7: Storage cost comparison

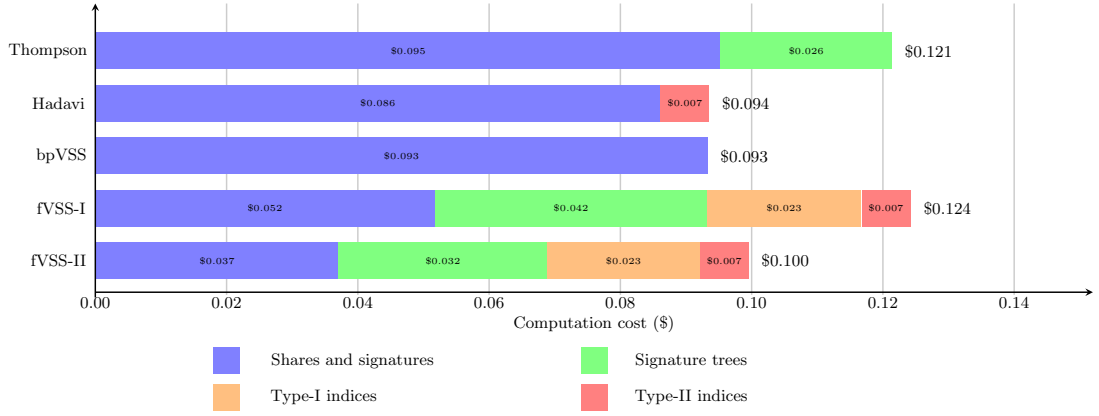
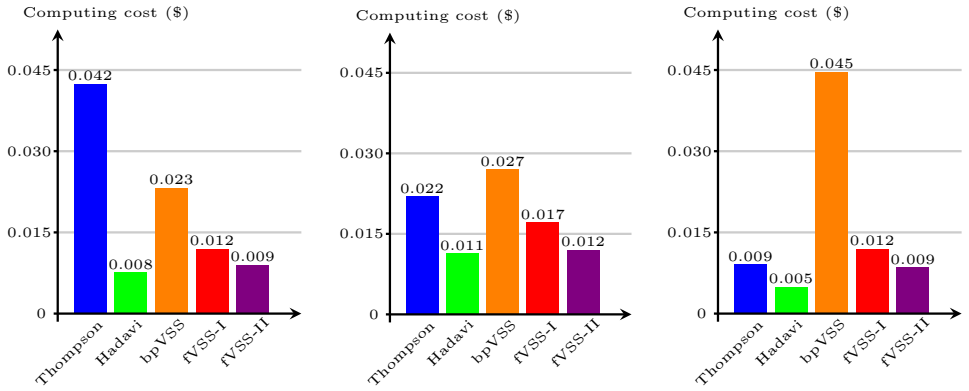


FIGURE B.8: Data sharing cost

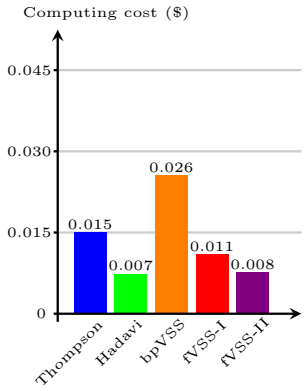
FIGURE B.9: Data reconstruction cost



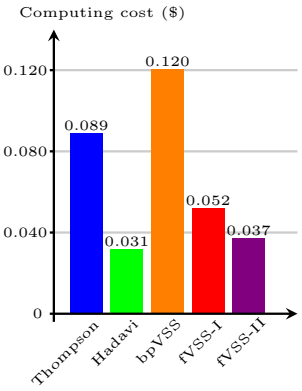
(a) Q1

(b) Q2

(c) Q3

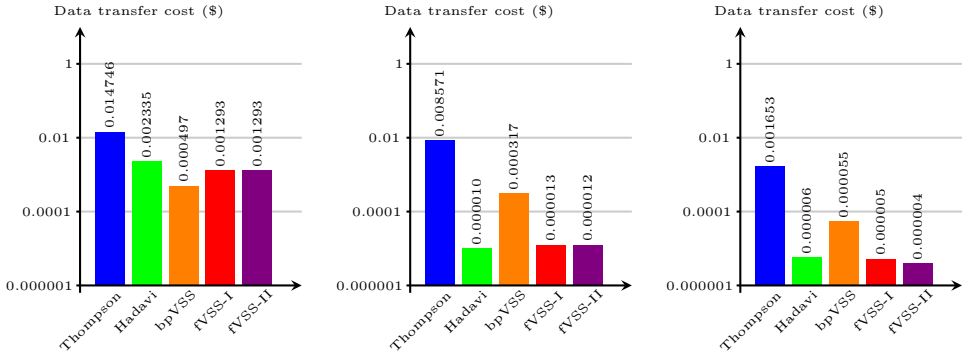


(d) Q4



(e) Total

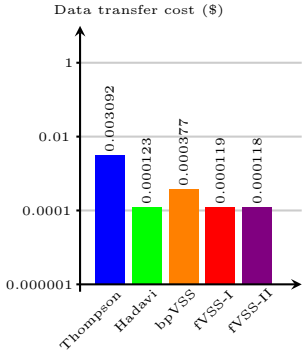
FIGURE B.10: Data access cost



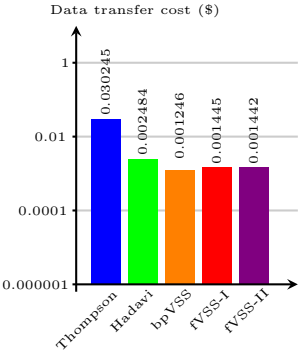
(a) Q1

(b) Q2

(c) Q3



(d) Q4



(e) Total

FIGURE B.11: Data transfer cost

Appendix C

Résumé détaillé

C.1 Contexte

L'informatique décisionnelle (BI) n'a cessé de croitre depuis plus de vingt ans, mais l'avènement récent de l'Infonuagique permet désormais de déployer des analyses de données encore plus facilement. Alors que la construction d'un système de BI traditionnelle nécessite généralement un investissement initial important avec l'Infonuagique et le modèle de paiement à la demande, les utilisateurs peuvent ponctuellement consacrer de petites quantités de ressources en échange d'un avantage en temps. Cette tendance est actuellement proposée par de nombreuses offres "de service BI" avec des enjeux économiques élevés.

Bien que l'Infonuagique soit actuellement en plein essor, la sécurité des données reste une des principales préoccupations des utilisateurs d'Infonuagique et des futurs utilisateurs. Certains aspects de la sécurité sont héritées des architectures distribués classiques, par exemple, l'authentification, les attaques de réseau et l'exploitation de certaines vulnérabilités, mais D'autres sont directement liés au nouvel environnement du Infonuagique, par exemple, la fiabilité d'un fournisseur de services de Infonuagique ou d'un sous-traitant, l'efficacité de la disponibilité et les mashups incontrôlées [2–4]. Dans le contexte particulier du cloud BI, la protection des données privées a une grande importance. Jusqu'à présent, les questions de sécurité ont été traitées par les fournisseurs de service (CSPs). Mais avec la multiplication des CSPs et des sous-traitants dans de nombreux pays, les questions juridiques complexes se posent, ainsi qu'une autre question fondamentale: la confiance. A savoir si la confiance doit être accordés aux CSPs ou finalement déplacer la prise en charge de la sécurité vers les utilisateurs finaux, avec les coûts générés.

Les risques de la sécurité des données stockées dans les nuages (surtout de type publics) sont représentés dans la figure C.1. Les données de l'utilisateur pourraient être supprimées, endommagées ou perdues pour plusieurs raisons. Premièrement, certains CSPs ont la politique de prendre le plus de profit. Par conséquent, les données non modifiées ou non utilisées peuvent être supprimées afin de servir d'autres clients. Deuxièmement, la perte de données peut aussi être causée par exemple, un incident involontaire, électrique ou réseau, ou intentionnel, par exemple, l'entretien ou la sauvegarde du système. En plus, les architectures du cloud basée sur la virtualisation possèdent des failles et ne sont pas suffisamment protégés contre les attaques. Enfin, tous les CSPs ne peuvent pas garantir à 100% la disponibilité des données, bien que certaines entreprises du cloud doivent fonctionner sur une base 7/24. Ainsi, la confidentialité des données, la disponibilité et l'intégrité sont les principaux enjeux en matière de sécurité des données dans les nuages.

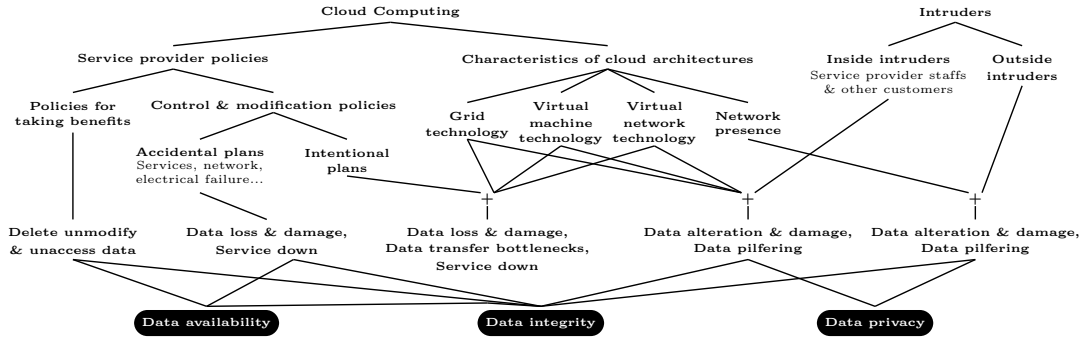


FIGURE C.1: Les risques (ou failles) de la sécurité des données entreposées dans les nuages

Dans le contexte du cloud BI, les entrepôts de données (DWs) dans les nuages ne doivent pas seulement être fortement protégés, mais aussi efficacement actualisés et analysés par le traitement d'analyse en ligne (OLAP). De là, pendant que les CSPs doivent optimiser la qualité de service et le profit, les utilisateurs cherchent à réduire les coûts de stockage et d'accès dans le modèle de paiement à la demande. Ainsi, pour les entrepôts de données dans les nuages, le compromis entre la sécurité des données et l'analyse OLAP à grande échelle pose un grand défi [4, 5].

C.2 Motivation et contribution

Les recherches existantes pour résoudre la confidentialité, la disponibilité et l'intégrité des données proposent des solutions basées sur la cryptage, l'anonymisation, la réplication ou la vérification des données. Seule l'utilisation du partage de clés secrètes (mono ou multi) permet de résoudre simultanément tous les problèmes de sécurité (confidentialité

des données, de disponibilité et d'intégrité). Toutefois, le stockage/mise à jour/Accès aux données avec le partage de clé secrète peut être difficile à mettre en œuvre et plus coûteux, car ces approches répliquent les données n fois et ne permettent pas l'accès aux données cryptées.

Toutes les approches basées sur le partage de clés secrètes permettent de crypter les données qui ne peuvent pas être déchiffrées par un seul CSP, ni aucun intrus qui arriverait à pirater un CSP. Cependant, une coalition ou de la compromission d'au moins t CSPs rompt le secret. Pour la disponibilité des données, toutes les approches permettent l'accès aux partages de $t \leq n$ CSPs, à savoir, les données partagées sont encore disponibles lorsque jusqu'à $n - t$ de CSPs sont indisponibles, à cause de défaillances techniques ou même par la malveillance. Toutefois, aucune approche ne permet la reconstitution des données partagées même si un seul CSP est indisponible, entravant ainsi les capacités de mise à jour des DBs dans les nuages. Bien que les approches de partage de secrets disposent de tous les opérateurs d'interrogation de base DB, aucun ne gère OLAP. Cependant, quelques approches garantissent effectivement l'intégrité des données, grâce à la vérification de seulement le code interne (pour vérifier si les CSPs sont malveillants). Enfin, une seule approche apporte des solutions pour réduire le volume de stockage global de sorte qu'il tombe bien sous n fois que des données originales, et ainsi diminuer les coûts financiers de stockage dans le modèle de paiement à la demande. Toutefois, son coût d'accès aux données reste élevé parce que les données interrogées doivent être entièrement reconstruites à l'avance.

Pour répondre à toutes ces questions, nous proposons deux nouvelles approches qui reposent sur un partage de clé secrète de base-p (bpVSS) et sur un mécanisme flexible vérifiables de partage de clés secrète (fVSS). A notre connaissance, bpVSS et fVSS sont les premières approches à base de partage de clés secrètes qui permettent l'exécution d'opérateurs OLAP sur DW ou cubes partagés sans reconstruire toutes les données en premier, et tout en minimisant le volume global des données partagées à moins de n fois que des données originales. Elles disposent également tous les deux (pour détecter des données incorrectes avant décryptage) de signatures pour la vérification des données interne (pour vérifier si les CSPs sont malveillants) et externe. En plus, fVSS est la première approche qui garantit qu'aucun groupe de CSPs ne peut contenir suffisamment da données partagées pour reconstruire le secret. fVSS permet également le rafraîchissement du DW lorsque l'un ou plusieurs CSPs est indisponible, et permet aux utilisateurs de régler le volume de données partagées de chaque CSPs ainsi sont optimisés les coûts par rapport aux différentes politiques de tarification des CSPs.

C.3 bpVSS: Base-p partage vérifiable de clé secrète

bpVSS est un nouveau schema vérifiable de clé secrète de type (t, n) base-p programme. Comme toutes les approches fondées sur le partage de secret, bpVSS partage les données sur n CSPs (Figure C.2), t est le nombre nécessaire de données partagées pour reconstruire les données originales. Chaque CSP ne stocke qu'une partie des données partagées, qui ne sont pas exploitables, ni par le CSP, ni par tout intrus, parce qu'elles ont été transformées par une fonction mathématique. Bien qu'on améliore le coût de traitement à travers cette approche, les données doivent être décryptées. Les résultats sont mathématiquement transformés sans que l'utilisateur soit au courant et ainsi elles seront reconstruites en informations significatives. Les données individuelles cryptées et les résultats de traitement étant chiffrées, leur transfert à travers les réseaux des CSP est donc sûr. Ainsi, la confidentialité de l'utilisateur est préservée à chaque point d'accès externe (réseaux, fournisseurs). La disponibilité est également garantie parce que les données peuvent encore être reconstituées si $n - t$ CSPs disparaissent.

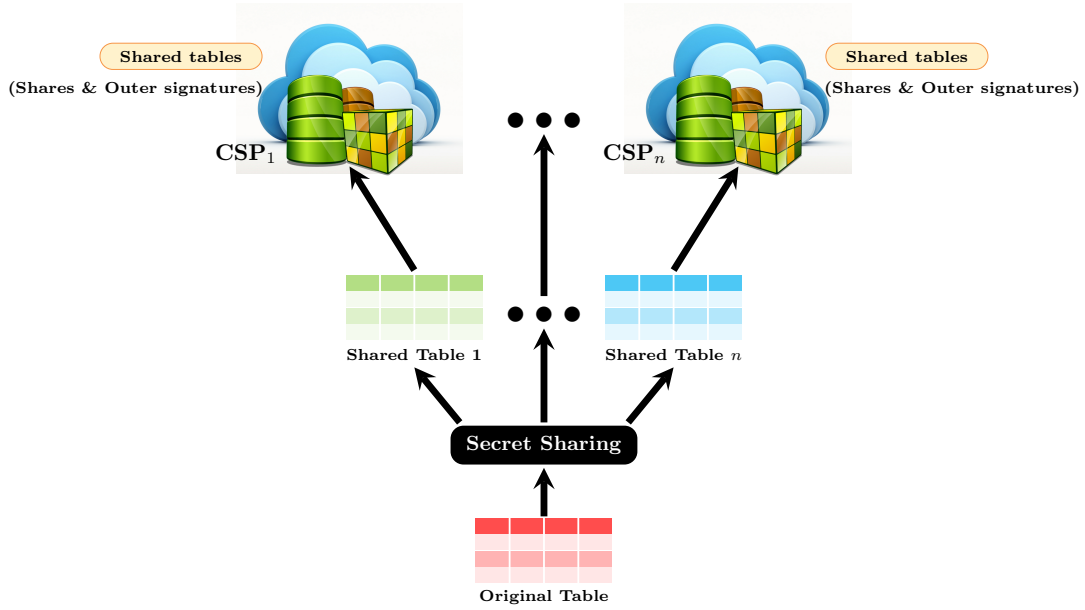


FIGURE C.2: Approche bpVSS

Dans bpVSS, un élément de données (un entier décimal) se transforme en un nombre entier de base-p tel que p est inférieur à la valeur de données. Ensuite, tous les chiffres d'un nombre entier de base-p sont cryptées à la fois par n distinctes t variables des équations linéaires f_i. Le volume des données partagées est beaucoup moins important que le volume de ses données, car les coefficients de f_i et base-p chiffres sont contrôlés à plus faible que défini par l'utilisateur paramètre p. Par conséquent, le volume totale des données partagées dans tous les n CSPs est inférieure n fois au volume de données.

Le volume de données partagées et donc le coût de stockage sont réduits au minimum par rapport aux autres approches.

Contrairement à toutes les approches, deux types de signatures (signatures internes et externes) sont incorporées dans bpVSS pour vérifier l'honnêteté des CSPs, l'exactitude des données et des données partagées. Les signatures internes créées à partir d'une fonction homomorphe pour aider à vérifier l'exactitude des données au cas où certains CSPs ne sont défaillants. Cependant, ils sont cachés dans des données partagées (ils font partie des données partagées), et donc pas de stockage supplémentaire pour garder les signatures internes. Les signatures extérieures créées à partir d'une fonction à sens unique permettent de vérifier les données partagées incorrectes ou erronées avant de reconstituer les données. Par conséquent, aucune données partagèes erronée n'est transférée à l'utilisateur pour la reconstruction. Cela permet de minimiser à la fois le coût de transfert de données et le coût de l'informatique du côté de l'utilisateur. Contrairement aux signatures intérieures, les signatures extérieures sont stockées dans des attributs supplémentaires dans des tables partagées.

Dans bpVSS, chaque table d'un DW partagée est stockée dans une base de données relationnelle chez un CSP, chaque valeur d'attribut est chiffrée indépendamment. Ainsi, bpVSS aide à mettre en œuvre un modèle logique DW, à savoir, en étoile, en flocon de neige ou en un schéma de constellation. Chaque DW partagé se base sur le même schéma que l'original de DW, mais le type et la taille de chaque attribut dans les tables partagées diffèrent des tables originales. Tous les types d'attributs sont en effet transformés en nombres entiers à l'exception des booléens qui ne sont pas cryptés pour préserver le coût de calcul et de stockage des données. Cependant, des tables partagées ont un plus grand nombre d'attributs que les tables d'origine, car les attributs de signatures extérieures sont également stockés dans des tables partagées. Toutes ne sont pas cryptées. Ils aident à détecter les enregistrements dans différentes tables partagées de chaque CSP. En plus, ils aident à regrouper les enregistrements partagés et les résultats dans le processus de restauration des données.

Pour gérer les types de données usuelles figurant dans les bases de données, nous chiffrons et traitons chaque valeur d'attribut de façon indépendante. Chaque valeur d'attribut (par exemple, réelles, caractères, chaînes de caractères, les chaînes binaires) est d'abord transformée en un ou plusieurs entiers en fonction du type de données. Ensuite, les nombres entiers sont chiffrés par bpVSS. Par exemple, un caractère est transformé en un nombre entier positif à travers son code ASCII. Comme certaines approches similaires, bpVSS permet l'analyse des données sur les données partagées. Pour optimiser le coût de l'informatique et le coût de transfert des données du côté de l'utilisateur lors de l'analyse des données, les requêtes de correspondance exacte et

les fonctions d'agrégation peuvent être directement effectuées sur les données partagées. Les résultats agrégés sont alors transférés à l'utilisateur pour la reconstruction. Toutes les opérations OLAP de base (roll-up, drill-down, some slice and dice, pivot and drill-across) peuvent également être appliquées directement sur les données partagées (cubes de nuages) des CSPs, avec des résultats en cours de reconstruction chez l'usager. Les cubes sont physiquement stockés dans des tables relationnelles qui sont partagées entre les CSPs, en conservant la même structure.

C.4 fVSS: Mécanisme de flexibilité de partage de secret vérifiables

fVSS est un schéma de (t, n) partage vérifiable flexible de clé secrète. Comme une approche basée sur le partage de données, fVSS découpe en n CSPs données partagées (Figure C.3), dont t sont nécessaires pour reconstruire les données originales. Par conséquent, ce qui garantit à la fois la confidentialité des données et la disponibilité de celles-ci.

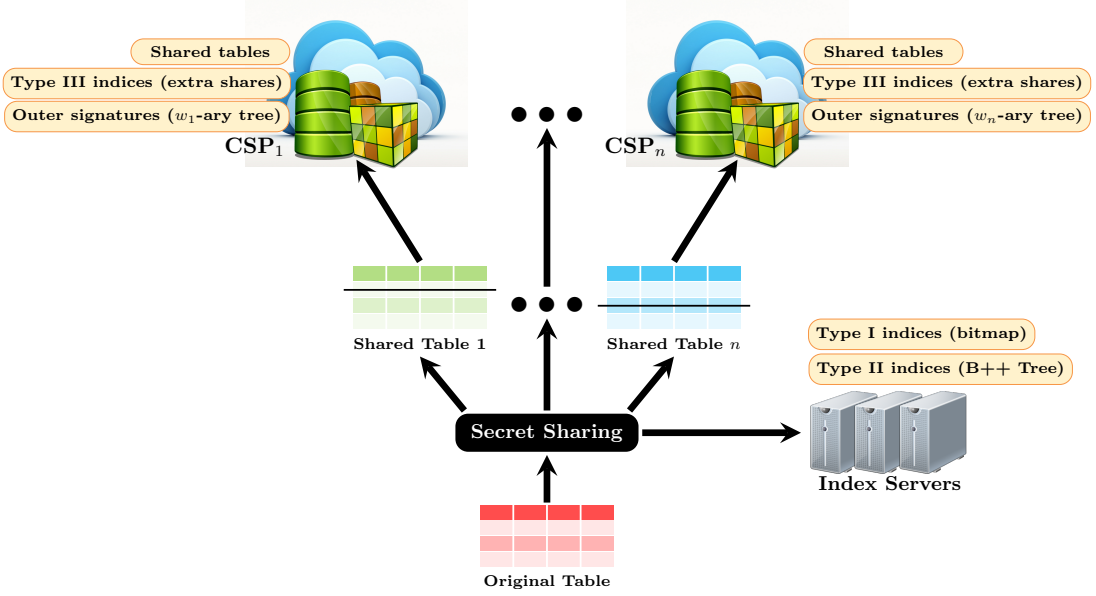


FIGURE C.3: Approche fVSS

Pour optimiser le volume de données partagées et donc le coût, nous partageons un morceau de données de moins que n fois. Seuls $n - t + 2$ parts d'une partie de données sont construits à partir d'un polynôme de degré t, qui est construit par un polynôme par interpolation de Lagrange en utilisant une signature intérieure et les valeurs semi-aléatoires $t - 2$ appelé donnée méta encryptée. Étant donné que chaque élément de la donnés est partagé à seulement quelques n CSP, fVSS est le premier partage de

secret flexible qui permet aux utilisateurs de régler le volume des données partagées en fonction des politiques de tarification CSP. Le volume déséquilibré des données partagées à chaque CSP aide en effet à minimiser le stockage et de calcul des coûts dans le modèle de paiement à la demande par la conception. En plus, contrairement à tous les autres systèmes de partage de secret, fVSS permet la mise à jour des données partagées en cas de défaillance de CSPs, tout simplement en ne sélectionnant pas les CSPs par défaut pour le partage de nouvelles données. fVSS atteint alors un niveau de sécurité plus élevé, car il peut protéger les données, même si toutes les CSPs sont malveillants ou indisponibles. Aucun groupe de CSPs peut contenir suffisamment de données partagées pour reconstruire le secret si $n < 2 \times t - 2$. En effet, $n < 2 \times t - 2 \Leftrightarrow nt + 2 < t$, soit le nombre de données partagées est inférieur au nombre de données partagées nécessaire à la reconstruction.

L'intégrité des données est renforcée avec des signatures à la fois internes et externes qui aident à détecter les erreurs dans les données partagées et les résultats de requête. Les signatures intérieures cachées dans des données partagées sont vérifiées après la reconstruction avec une fonction à sens unique, comme dans bpVSS. Les signatures extérieures sont stockées dans une structure de données arborescente. Ces dernières sont créées à l'aide d'une fonctions à sens unique, qui aident à réduire le volume de la signature extérieure et permettent une mise à jour à chaque fois qu'un DW partagé est rafraîchie. Par conséquent, la mise à jour des signatures extérieures accélère le stockage ainsi les coûts sont réduits au minimum. Plusieurs types de vérifications de signature externe nécessaires (vérification des enregistrements, vérification d'ensemble d'enregistrements, vérification de table, l'ensemble de tables, et de vérification DW) sont disponibles sur demande, parce que les signatures extérieures sont créées indépendamment des différentes combinaisons de enregistrements partagés ou des tables partagées avec plusieurs fonctions de cryptage. Par ailleurs, les signatures extérieures à différents niveaux dans l'arborescence de la signature extérieure peuvent être créées et vérifiées pour réduire le taux de données incorrectes non détectées, ainsi l'intégrité est améliorée.

Étant donné que chaque table d'un DW partagée est stockée dans une base de données relationnelle d'un CSP, chaque valeur d'attribut dans chaque document est chiffrée indépendamment, notre approche permet d'appliquer un modèle logique de DW, à savoir, étoile, flocon de neige et de schémas de constellation. Un DW partagé porte le même schéma que l'original de DW. Cependant, tous les types d'attributs sont transformés en réels par le processus de partage des données. Contrairement à bpVSS, les signatures extérieures sont stockées en dehors des tables partagées, à savoir, dans l'arborescence de la signature. Ainsi on a moins d'enregistrements partagés à crypter de n'importe quel enregistrement original et moins de tables partagés pour les stocker. Ainsi, le nombre d'enregistrements dans une table partagée est inférieur à celui de la

table d'origine, et diffère de celle des données partagées des autres CSPs. Pour améliorer les performances de la requête et réduire les coûts de stockage et de calcul, le nombre d'enregistrements partagés dans la table partagée doit être ajusté pour respecter les politiques de tarification des CSPs.

Comme d'autres approches de base de données sécurisées, fVSS permet l'analyse de données sur les données partagées. Pour analyser les données, des requêtes SQL traditionnelles peuvent être effectuées sur les données partagées des CSPs sans décryptage, puis seulement les résultats sont reconstruits pour l'utilisateur. Cela permet de réduire les coûts de communication et de calcul chez l'usager. Trois types d'indices permettent d'améliorer les performances des requêtes. Les indices de type I sont des bitmaps stockés dans le serveur d'index. Ils stockent des emplacements de données partagées et sont utilisés dans les requêtes SUM et AVG. Comme dans certaines approches existantes, les indices de type II sont des arbres B+ stockés sur le serveur de l'indice. Ils sont utilisés dans correspondance exacte, la portée et MAX, MIN, médiane, le mode et les requêtes COUNT. Les indices de type III sont cryptées et stockées chez CSP. Ils sont exploités pour les requêtes instance de stdDev et la variance. Comme bpVSS, fVSS se base sur le stockage des cubes de données qui permettent d'optimiser le temps de réponse et la bande passante lors de l'exécution des opérations ROLAP. Contrairement à bpVSS, fVSS crée un cube partagé d'enregistrements et indices sans reconstruire les données. Enfin, les cubes partagés peuvent être actualisés, même si certains CSPs disparaissent.

Comme bpVSS, fVSS se base sur le stockage des cubes de données qui permettent d'optimiser le temps de réponse et la bande passante lors de l'exécution des opérations ROLAP. En plus dans fVSS, les cubes sont créés directement dans le nuage et mis à jour par les données partagées et indices seulement. Cependant, les données partagées sont réelles et les signatures extérieures ne sont pas stockées dans des cubes du cloud. Depuis le Infonuagique, les cubes sont construits à partir des données partagées, ils sont physiquement stockés dans des tables qui doivent être partagées à tous les n CSPs, parce que les méta-données partagées ne sont pas disponibles. Cependant, les cubes de cloud peuvent être actualisés, même si certains CSP sont défaillants. En plus de références de dimensions usuelles et d'agrégats, ils peuvent inclure des attributs supplémentaires qui sont effectivement intégrés de type indices III.

C.5 Sécurité, performance et analyse des bpVSS fVSS

Dans cette section, nous discutons de la sécurité et la performance de bpVSS et fVSS. Par sécurité des données, nous entendons la confidentialité, la disponibilité et

l'intégrité des données. Le volume des données cryptées et le coût du temps de traitement sont attribuées à la performance.

La confidentialité des données est le principal problème de sécurité sur lequel nous nous concentrons. De par leur conception, nos approches renforcent la protection des données basées sur le partage des données et garantissant ainsi qu'elles ne peuvent pas être déchiffré par un seul CSP ou un intrus qui pirater un CSP. L'approche fVSS garantit qu'aucun groupe de CSPs ne peut avoir suffisamment de données partagées pour reconstruire les données d'origine si $n < 2 \times t - 2$. Toutefois, dans le cas où un intrus peut voler une partie à partir d'au moins CSP, la probabilité d'apparition du secret dépend de t et $\|p\|$ (en bpVSS). Pour atteindre une protection plus élevée, t et $\|p\|$ devraient être les grands entiers. Non seulement la sécurité mais aussi la performance dépendent de t et $\|p\|$. Lorsque t est grand, nos approches produisent de petites données partagées, ce qui permet de minimiser la consommation de mémoire et le temps d'exécution lors du chargement et de l'accès aux données. Cependant, t peut ne pas être trop grand, parce que le nombre des CSPs est limité dans la pratique. De même, $\|p\|$ ne devrait pas être supérieur à la taille maximale d'un élément de données dans bpVSS. Lorsque t et $\|p\|$ sont affectés à de grands entiers, le volume global des données partagées est très large, et donc le coût de stockage de données est élevé. Pour atteindre la plus grande sécurité avec le plus bas coût possible du stockage, t devrait être un grand entier et $\|p\|$ devrait égale à $\|d_{max}/(t-1)\|$, où $\|d_{max}\|$, est la taille de données secrètes la plus importante. En revanche, t n'a pas d'impact sur volume global de données partagées du fVSS, qui est fixé à deux fois, une première fois avec le volume de données d'origine et une deuxième fois lorsque $n = t$. Par conséquent, aucune tradoff entre la confidentialité et le coût de stockage avec fVSS. Toutefois, lorsque t est grand, l'efficacité du partage des données et de la reconstruction est négativement impacté.

En ce qui concerne la disponibilité, nos approches, toujours par conception, permettent de reconstruire les secrets, à savoir, les données partagées de la requête, lorsque $n - t$ CSPs est indisponible. En outre, fVSS permet également la mise à jour des données partagées dans le cas où plus $t - 2$ CSPs sont indisponibles, tout simplement en partageant de nouvelles données entre $n - t + 2$ CSPs disponibles. Toutefois, la part globale du volume augmente avec n lorsque t est fixé. Le temps de partage chez l'utilisateur augmente également avec n. Ainsi, pour atteindre la disponibilité des données tout en minimisant le volume global des données partagées et de temps de partage, n devrait être proche de t. Cependant, n et t doivent respecter le condition $n < 2 \times t - 2$ et $t > 2$ dans fVSS pour atteindre la protection la plus élevée (aucun groupe de CSP ne peut briser le secret) et pour garantir que de nouvelles données peuvent être partagées même si certains CSP sont défaillants.

Pour assurer l'intégrité, nos approches permettent à la fois de vérifier l'exactitude des données cryptées et l'intégrité des CSPs, avec l'aide de deux vérifications de code externe et interne. L'efficacité des signatures internes et externes est plus élevée lorsque leurs tailles sont grandes. Depuis $\|s_{in}\| = \|p\|$ et $\|p\|$ concerne t en bpVSS, $\|s_{in}\|$ doit satisfaire à la condition $\|s_{in}\| = \|p\| = \|d_{max}/(t-1)\|$ pour atteindre la meilleure efficacité et le plus bas coût de stockage possible. $\|s_{in}\|$ n'a pas d'impact sur la taille des données partagées en fVSS, mais $\|s_{in}\|$ devrait être un grand entier et supérieure à une moitié de la taille des données afin de détecter tous les morceaux de données incorrectes (Section 5.3.2.3.1).

Contrairement à la taille de la signature intérieure, la taille externe augmente avec la signature $\|s_{out}\|$ lorsque t et n sont fixés. Par conséquent, l'efficacité et le volume des signatures extérieures doivent être équilibrés. Les signatures internes et externes travaillent ensemble lors de la reconstruction de données avec bpVSS. Pour détecter tous les morceaux de données incorrectes, $\|s_{out}\|$ devrait être d'au moins 6 bits. Ce résultat est obtenu lorsque le volume global de la signature est de 0,75 Go (données d'origine est de 1 Go et le volume de l'action globale maximale est de 3,1250 Go). Dans fVSS, les signatures extérieures sont vérifiées sur demande et se séparent de la signature intérieure. L'efficacité et le volume des signatures extérieures augmentent avec $\|s_{out}\|$. Ainsi, $\|s_{out}\|$ devrait être un grand entier. A partir des signatures extérieures peuvent être vérifiés sur demande sur plusieurs niveaux (par exemple, enregistrements partagés, l'ensemble d'enregistrements partagés, tables partagées, l'ensemble de tables partagées, et le DW partagé) de l'arbre de la signature w_i-aire, toutes les données partagées incorrectes peuvent être détectées si les signatures sont vérifiées à au moins trois niveaux (Section 5.3.2.3.1). En outre, w_i impacts volume de signature. Ainsi, il devrait réduire le volume de la signature, et donc le coût de stockage.

Nos approches permettent de partager les données numériques (par exemple, les entiers et les réels) et non les non numériques (par exemple, les caractères et les chaînes de caractères) des données du DW. Pour accéder aux données partagées, ils permettent à tous les types de requêtes (correspondance exacte, la portée, l'agrégat et le regroupement des requêtes) sur les données partagées. En outre, ils permettent également le traitement directement sur les données partagées en créant des cubes de données partagées. Toutefois, $n/(t-1)$ et n fois le volume initial de cube sont nécessaires pour stocker une partie du nuage dans bpVSS et fVSS. Étant donné que le nombre d'enregistrements dans un cube de nuage est égal aux différentes combinaisons de toutes les valeurs de dimension, il peut être énorme. Par conséquent, les cubes dans le cloud doivent stocker uniquement les enregistrements qui sont souvent accessibles pour réduire le volume de stockage, qui devient un problème de sélection de vue matérialisée. D'autres enregistrements peuvent être directement extraits de la DW partagé.

Enfin, non seulement le paramétrage, mais aussi le volume déséquilibré des données partagées aide à réduire la taille du modèle ainsi que le stockage et le coût de traitement dans fVSS. Ainsi, le plus grand volume de données partagées doit être conservé auprès du CSP le moins cher. Cependant, la taille maximale de la machine virtuelle doit être attribué aux CSPs qui stockent les plus gros volumes de données partagées, à réduire l'écart entre les temps d'exécution les plus bas et les plus élevés du CSPs. Le partage de données ou l'accès fonctionne parallèlement sur les CSPs, le temps total de traitement est en effet représenté par le traitement individuel le plus important.

C.6 Étude comparative

Dans cette section, nous expérimentons nos approches proposées dans les chapitres 3 et 4. En effet, nous les comparons avec les deux états de l'algorithme de l'état de l'art (Thompson et al approche [58] et Hadavi et al approche [21] présentée dans la chapitre 2), car seuls ces approches se concentrent sur les trois aspects de la sécurité (confidentialité, disponibilité et intégrité des données) et respectent aussi la performance lorsque les requêtes agrégées sont exécutées. La performance de toutes les approches expérimentales est mesurée avec un volume de données partagées, le partage de données/temps de la reconstruction, le temps de réponse de la charge de travail et le volume de données transféré pour confirmer l'étudier théoriquement dans la chapitre 5. Dans les expériences, nous utilisons l'indice de référence pour un schéma en étoile et un paramètre p qui varie (en bpVSS) avec $t = 3$, $n = 4$ et $w = 100$ (en fVSS).

Tableau C.1 caractéristiques de volume de données, temps d'exécution, le volume de transfert de données et les coûts financiers de toutes les approches expérimentales. Les coûts financiers sont estimés à partir des politiques de tarification de CSP représentés dans le Tableau 5.1 (Chapitre 5) et le serveur d'index utilise le même prix de la CSP le plus cher pour obtenir le meilleur service. Notez que fVSS-I et fVSS-II sont les parties qui traitent le déséquilibre des données partagées et les stratégies de déséquilibre de données partagées dans fVSS.

Lorsque SSB DB dimensionnement 757 Mo est partagé avec des approches expérimentales, volume de stockage fVSS (tout type de données) est la plus faible. Bien bpVSS a réussi à minimiser le volume global de données partagées, son volume de stockage global (tous les types de données) est encore plus grande que celle de l'approche Hadavi et al, parce qu'il construit d'énorme volume de signatures extérieures.

Toutefois, lorsque le coût de stockage financier est estimé à partir du volume de stockage, le coût de stockage fVSS-II et bpVSS sont les moins importants. bpVSS coût

TABLE C.1: Comparaison des approches de partage de base de données

	Thompson	**Hadavi**	**bpVSS**	**fVSS-I**	**fVSS-II**
Storage volume (GB)	4.14	2.43	2.62	2.34	**2.27**
Data transfer volume (MB)	323.88	23.90	13.51	12.36	**12.34**
Data sharing time (min)	58.42	27.05	40.35	**23.62**	30.05
Data reconstruction time (min)	35.16	24.50	33.59	**17.95**	34.09
Data access time (min)	5.81	**2.34**	8.51	3.61	5.69
Storage cost ($/month)	$0.5584	$0.3023	$0.2973	$0.3168	**$0.2651**
Data transfer cost ($)	$0.0302	$0.0025	**$0.0012**	$0.0014	$0.0014
Data sharing cost ($)	$0.1214	$0.0935	**$0.0933**	$0.1242	$0.0996
Data reconstruction cost ($)	$0.4092	$0.2599	$0.3964	$0.2641	**$0.2071**
Data access cost ($)	$0.0887	**$0.0314**	$0.1204	$0.0520	$0.0372

de stockage est inférieure à celle de l'approche Hadavi et al bien bpVSS volume de stockage est supérieure à celle de l'approche Hadavi et al, parce bpVSS ne stocke rien au niveau du serveur d'index qui est le prix de l'unité la plus chère (le serveur d'index est pas dans la piscine CSP bpVSS). De même, étant donné que le volume de stockage de fVSS-I au niveau du serveur d'index est supérieure à celle de Hadavi et al approche, le coût de stockage de fVSS-I est plus élevée que celle de l'approche Hadavi et al bien que le volume de stockage global de fVSS-I est inférieure à celle de l'approche de Hadavi et al.

Pour le partage de données, notre SSS est inefficace pour le partage de données. Le temps de traitement de partage des données de nos approches se situe entre celle de l'approche la plus efficace (Hadavi et al) et la pire approche (Thompson et al). Cependant, le coût de calcul financier pour le partage de données est la plus faible, parce que bpVSS n'a pas de charge de travail au niveau du serveur d'index. De même, le coût de calcul financier de fVSS-I est le plus élevé, bien que les données de fVSS-I en temps partagé est inférieure à celle de l'approche de Thompson et al, parce que les données fVSS-I partage le temps au niveau du serveur d'index est plus élevée que celle de approche de Thompson et al. La stratégie de déséquilibre en fonction de la taille de la machine aide fVSS-II à réduire le coût de calcul financier par rapport à fVSS-I.

Pour la reconstruction des données, fVSS-I reconstitue des données plus rapidement grâce à la stratégie de déséquilibre du coût de calcul financier pour reconstruire des données avec fVSS-II sont les plus bas, bien que le temps de la reconstruction fVSS-II est pas plus bas. bpVSS est inefficace pour reconstruire des données, parce que bpVSS vérifie l'exactitude des données partagées en cours reconstruire. Avec le temps d'exécution supplémentaire pour vérifier les données partagées, le temps de la reconstruction bpVSS et le coût de calcul financier sont un peu inférieure à celle de l'approche de Thompson et al, qui est la pire des solutions pour reconstruire des données.

Pour accéder à des données qui est le l'usage principal du DW, notre SSS est le plus efficace lorsque le volume et le coût financier de transfert de données de départ sont

respectées. Le volume de transfert des données et le coût financier de notre proposition d'attribution de signatures sont seulement environ la moitié de celle de l'approche Hadavi et al soit environ 4% de celle de l'approche de Thompson et al. Cependant, le volume de transfert des données est toujours un problème pour toutes les approches basées sur le partage de secrets. Le goulot d'étranglement du réseau peut être à cause l'utilisateur, car un volume énorme de données est transféré in/out de l'utilisateur. Nous allons illustrer ce problème par l'exemple. Lorsque le pire Q1.1 vol requête sont exécutés avec fVSS-II étant l'approche la plus efficace pour le volume de transfert de données, les données sont transférés environ 7,13 MB de l'utilisateur et transférés sur environ 28.51 les MB de l'utilisateur. Ainsi, le volume global de transfert de données dans et hors de l'utilisateur est 35.64 Mo soit environ 4,71% du volume SSB DW original (757 Mo). Si DW dimensionnement 10 TB est partagée et une seule requête est exécutée, les données sont transférées in/out à l'utilisateur d'environ 480 Go. Ce problème se produit parce que nos systèmes de règlement peuvent fonctionner seules les requêtes de reconstruction exactes, agrégées et triées sur les données partagées, mais ils ne peuvent pas exécuter des requêtes complexes sur les données partagées. Cependant, pour résoudre ce problème, nous envisageons l'exécution de requêtes complexes sur les données partagées en matière de recherche future.

Bien que notre attribution de signatures est la plus efficace pour réduire le volume de transfert de données de départ et le coût financier, ils ne sont pas efficaces pour les temps d'exécution et le coût de calcul financier lorsque l'accès aux données avec toutes les requêtes SSB. Le temps de réponse de la charge de travail bpVSS et le coût de calcul sont les plus élevés, car il doit vérifier les données partagées avant de reconstituer les données. Le temps de réponse de la charge de travail et le calcul des coûts fVSS se situe entre celle de l'approche la plus efficace de Hadavi et al et l'approche la moins efficace celle de Thompson et al. Le temps de réponse de la charge de travail fVSS n'est pas plus élevé que celui de l'approche de Hadavi et al, parce fVSS effectue le traitement sur des réels mais l'approche Hadavi et al effectue le traitement sur des entiers.

Notez que le partage de données et de temps d'accès de notre SSS n'est pas plus bas parce que les expériences sont menées avec n et t sont de petits entiers ($n = 5$ et $t = 4$). Si n et t sont assez grands, le volume de données partagées de chaque CSP sera réduit jusqu'à ce que le temps d'exécution pour le partage et l'accès aux modifications de données au plus bas, parce que le volume de données partagées et le temps d'exécution de chaque CSP ne diminuent pas lorsque n et t diminuer.

Enfin, fVSS-II prend en considération le déséquilibre du volume de données partagées et applique une stratégie de déséquilibre sur le modèle qui permet de réduire considérablement les coûts financiers à l' exception des coûts de transfert de données qui

sont calculés à partir de fVSS-I. Cependant, dans l'expérience, les données partagées fVSS-II et les données d'accès plus lent que fVSS-I parce que l'écart entre les temps d'exécution les plus bas et plus élevés au CSP est élevé. Cela arrive parce que le volume des données partagées ne correspond pas à la puissance de la machine qui les exécute. Ainsi, nous visons dans les travaux futurs la conception d'un outil semi-automatiquement qui aide les utilisateurs à ajuster le volume des données partagées de chaque CSP, par rapport aux coûts, mais également par rapport à la qualité de service. Bien que fVSS (deux stratégies) et la comparaison des approches garantie de disponibilité des données lorsque certains CSP sont défaillants, ils ne peuvent pas accéder aux données avec des requêtes SSB si le serveur d'index échoue. En revanche, c'est possible avec bpVSS p. En outre, seules nos approches empêchent le transfert de données partagées erronées lorsque les données sont accessibles par la vérification des signatures extérieures.

C.7 Perspective

Dans cette section, nous discutons de certaines questions en suspens qui devraient être abordées dans nos approches pour améliorer la sécurité, les performances et le coût.

Tout d'abord, nous avons l'intention d'améliorer la confidentialité des données, puisque dans bpVSS, les noms de table, les noms d'attributs, des clés primaires et étrangères sont diffusés en clair. Ainsi, les données chiffrées peuvent être attaqués avec, par exemple, banburismus, références ou méthodes de fréquence (avec l'aide d'une base de connaissances). Ainsi, pour atteindre une plus grande sécurité, les noms de table, les noms d'attributs (y compris dans fVSS) et les touches doivent être cryptées pour cacher le schéma de données. Aucune méthode SSS actuelle ne peut atteindre cet objectif parce que chaque nom de la table (après que son nom soit crypté) doit être différent des autres noms de tables si elles sont stockées dans le même CSP ou noeud. De même, les noms d'attributs chiffrés et les clés primaires chiffrés dans chaque table partagée doivent être différents. En outre, les touches chiffrées dans un couple des clefs étrangères primaire devraient être différentes pour cacher la relation entre les tables cryptées. Par conséquent, nous prévoyons d'utiliser une fonction injective et un moyen de crypter les noms de table, les noms des attributs et des clés dans chaque table à chaque noeud.

Deuxièmement, nous cherchons à minimiser le risque de goulot d'étranglement et d'optimiser le temps de réponse aux requêtes. Bien que nos approches permettent des requêtes de correspondance exacte, portée et d'agrégation en cours de traitement sur des données partagées, ils ne peuvent pas exécuter des requêtes composites, à savoir, les requêtes impliquant les deux opérateurs de correspondances et d'agrégation exactes. Par conséquent, lorsque les requêtes complexes sont exécutées, de grands volumes de données

sont transférés entre l'utilisateur et les CSPs (chapitre 6). Dans bpVSS, ce problème peut se produire parce que les deux résultats de type vrai positif et faux positif de requêtes de correspondance exacte sont transférés au filtre l'utilisateur. Ensuite, seuls vrais positifs sont transférés aux CSP pour traiter l'agrégation. Ainsi, nous prévoyons d'éliminer les faux positifs de la correspondance et d'utiliser que des requêtes précises afin d'améliorer les recherches des requêtes complexes exécutées sur les enregistrements. Dans fVSS, toutes les requêtes portées sur les correspondances exactes et certaines requêtes agrégées doivent fonctionner sur indices (bitmaps et B+ arbres) stockées sur le serveur d'index. En outre, le stockage des données partagées est aléatoire. Ainsi, nous devrions chercher une solution pour organiser le stockage des enregistrements sans l'aide d'indices.

Troisièmement, nous avons l'intention d'améliorer encore le coût de notre solution dans le nuage avec le modèle de paiement à la demande. Les expériences fVSS (Chapitre 6) montrent que les coûts élevés (coûts de stockage, de calcul et de transfert des données) sont payés pour le serveur d'index. Cependant, le temps d'exécution est élevé si le volume de données est déséquilibré avec la puissance de la machine, parce que le partage de données ou l'accès fonctionne parallèlement sur les CSPs et le temps total d'exécution est le temps d'exécution individuelle le plus important. Par conséquent, nous devrions éliminer le serveur d'index et de concevoir un outil qui aide les utilisateurs en semi-automatiquement à ajuster le volume des données partagées de chaque CSP, par rapport aux coûts, mais aussi la qualité de service. Cela est possible si l'emplacement des données partagées est organisé comme dans le paragraphe précédent.

Enfin, puisque les politiques de tarification de CSP et d'entretien sont susceptibles d'évoluer rapidement, nous visons à concevoir une méthode pour ajouter et supprimer des CSPs de/vers l'ensemble CSP dans fVSS, avec les plus bas coûts de mise à jour possibles, tout en préservant l'intégrité des données. Dans tous les systèmes de règlement, y compris nos approches, tout CSP peut être immédiatement retiré sans aucun impact, même si $n \geq t$. En revanche, quand un nouveau CSP est ajouté, les données partagées de toutes les données déjà existantes doivent être partagées chez le nouveau CSP pour gérer la cohérence des données. Les données partagées enregistrées chez le nouveau CSP doivent être construits à partir de t CSP existants et donc, le processus de construction est coûteux, à la fois dans le temps d'exécution et de bande passante. Pour éviter cela, nous pouvons encore exploiter les méta-données partagées dans fVSS, mais les fonctions de construire des méta- données partagées dans la nouvelle et les CSPs existants doivent être redéfinies de manière dynamique afin de relier tous les CSPs, qui est un défi.

Nos approches peuvent être appliquées pour stocker et analyser les données massives. Les aspects de volumétrie des données, de la rapidité et l'hétérogénéité (variété), peuvent être abordées et traitées par nos approches.

Tout d'abord, notre approche est une approche de base de données distribuée où le volume de données partagées individuelles diminue lorsque le nombre de CSPs/participants augmente. Cependant, le nombre n de CSP est limité dans la pratique, et le volume des données partagées de chaque CSP peut ne pas être assez petit pour être stocké dans une base de données. Ainsi, pour stocker un grand volume de données, nous changeons la stratégie de conception de "un participant par un CSP" à "beaucoup de participants par un CSP".

Deuxièmement, dans nos approches, l'accès aux données fonctionne parallèlement avec plusieurs participants, et seuls les éléments de données nécessaires sont reconstruits par l'utilisateur. Cependant, lorsque le nombre de Participants augmente, le temps d'accès à chaque participant diminue, parce que le volume de données partagées diminue et le nombre d'enregistrements partagés peut également diminuer (en fVSS). Ainsi, nos approches, en particulier fVSS, peuvent interroger des flux de données. Toutefois, les flux de données partagés peuvent être un problème, parce que tous les éléments de données doivent être cryptés pour l'utilisateur des polynômes et des augmentations de temps de chiffrement avec le nombre de participant. En plus, le nombre de participant doit être suffisamment grand pour stocker un grand volume de données. Ainsi, la complexité de cryptage doit tomber sous polynomiale pour permettre aux données de flux de partage. Nous prévoyons que ce sera résolu par quelques recherches connexes (par exemple, des champs numériques de données, les champs de cryptographie) à l'avenir.

Troisièmement, nos approches peuvent partager tous les types de données si les éléments de données sont identifiés par une clé et leur type peut être converti en entiers, réels, des caractères, des chaînes ou des chaînes binaires. Par exemple, les fichiers de texte ne peuvent être partagés avec nos approches que si la clé générée est identique à chacun des fichiers et du cryptage des chaînes dans des fichiers texte. Par conséquent, nous pouvons partager ces types de données: textuelles ou de documents XML, des images, des vidéos ou des données graphiques.

Bibliography

[1] Abhishek Parakh and Subhash Kak. Online data storage using implicit security. *Information Sciences*, 179(19):3323–3331, September 2009.

[2] James Bret Michael, Phillip A. Laplante, Jeffery E. Payne, Paul E. Black, and Jeffrey M. Voas. Does security trump reliability? *IEEE Computer*, 46(11):84–86, November 2013.

[3] David S.L. Wei, San Murugesan, Sy-Yen Kuo, Kshirasagar Naik, and Danny Krizanc. Enhancing data integrity and privacy in the cloud: An agenda. *IEEE Computer*, 46(11):87–90, November 2013.

[4] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus 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. ISBN 978-1-60558-784-4.

[5] Radu Sion. Towards secure data outsourcing. pages 137–161, 2008.

[6] Dorothy Elizabeth Robling Denning. *Cryptography and data security.* Addison-Wesley, 1982.

[7] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, berkeley, USA*, pages 44–55, 2000.

[8] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. Secure ranked keyword search over encrypted cloud data. In *30th IEEE International Conference on Distributed Computing Systems (ICDCS 2010), Genoa, Italy*, pages 253–262, 2010.

[9] Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1467–1479, December 2011.

[10] Chun-I Fan and Shi-Yuan Huang. Controllable privacy preserving search based on symmetric predicate encryption in cloud storage. *Future Generation Computer Systems*, 29(7):1716–1724, September 2013.

[11] Hakan Hacıgümüş, Bala Iyer, and Sharad Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *9th International Conference DASFAA 2004, Jeju Island, Korea*, pages 125–136, 2004.

[12] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *23th ACM Symposium on Operating Systems Principles (SOSP 2011), Cascais, Portugal*, pages 85–100, 2011.

[13] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, March 2013.

[14] Carlos Aguilar Melchor, Guilhem Castagnos, and Philippe Gaborit. Lattice-based homomorphic encryption of vector spaces. In *IEEE International Symposium on Information Theory (ISIT 2008), Toronto, Canada*, pages 1858–1862, 2008.

[15] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *41st annual ACM symposium on Theory of computing (STOC 2009), Bethesda, USA*, pages 169–178, 2009.

[16] Nadia Bennani, Ernesto Damiani, and Stelvio Cimato. Toward cloud-based key management for outsourced databases. In *34th Annual Computer Software and Applications Conference Workshops (COMPSACW 2010), Seoul, Korea*, pages 232–236, 2010.

[17] Haibo Hu, Jianliang Xu, Chushi Ren, and Byron Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *27th IEEE International Conference on Data Engineering (ICDE 2011), Hannover, Germany*, pages 601–612, 2011.

[18] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *14th Annual International Cryptology Conference (CRYPTO 1994),Santa Barbara, USA*, pages 216–233, 1994.

[19] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In *27th Annual ACM Symposium on Theory of Computing (STOC 1995), Las Vegas, USA*, pages 45–56, 1995.

[20] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Ensuring data storage security in cloud computing. In *17th International Workshop on Quality of Service (IWQoS 2009), Charleston,USA*, pages 1–9, 2009.

[21] Shiyuan Wang, Divyakant Agrawal, and Amr El 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] Amos Beimel. Secret-sharing schemes: A survey. In *3rd International Conference on Coding and Cryptology (IWCC 2011), Qingdao, China*, pages 11–46, 2011.

[23] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[24] G. R. Blakley. Safeguarding cryptographic keys. In *National Computer Conference (AFIPS 1979), Monval, USA*, pages 313–317, 1979.

[25] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29(2):208–210, March 1983.

[26] Sorin Iftene. General secret sharing based on the chinese remainder theorem with applications in e-voting. *Electronic Notes in Theoretical Computer Science*, 186: 67–84, July 2007.

[27] Lein Harn and Changlu Lin. Strong (n, t, n) verifiable secret sharing scheme. *Information Sciences*, 180(16):3059–3064, August 2010.

[28] Abhishek Parakh and Subhash Kak. Space efficient secret sharing for implicit data security. *Information Sciences*, 181(2):335–341, January 2011.

[29] Yan-Xiao Liu, Lein Harn, Ching-Nung Yang, and Yu-Qing Zhang. Efficient (n, t, n) secret sharing schemes. *Journal of Systems and Software*, 85(6):1325–1332, January 2012.

[30] Chou-Chen Yang, Ting-Yi Chang, and Min-Shiang Hwang. A (t,n) multi-secret sharing scheme. *Applied Mathematics and Computation*, 151(2):483–490, April 2004.

[31] Chao-Wen Chan and Chin-Chen Chang. A scheme for threshold multi-secret sharing. *Applied Mathematics and Computation*, 166(1):1–14, July 2005.

[32] Atsushi Waseda and Masakazu 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.

[33] Shi Runhual, Huang Liusheng, Luo yonglong, and Zhong Hong. A threshold multi-secret sharing scheme. In *IEEE International Conference on Networking, Sensing and Control (ICNSC 2008), Sanya, China*, pages 1705–1707, 2008.

[34] Satoshi Takahashi and Keiichi Iwamura. Secret sharing scheme suitable for cloud computing. In *27th international conference on advanced information networking and applications (AINA 2013), Barcelona, Spain*, pages 530–536, 2013.

[35] Bu ShanYue and Zhou Hong. A secret sharing scheme based on NTRU algorithm. In *Wireless Communications, Networking and Mobile Computing (WiCom 2009), Beijing, china*, pages 1–4, 2009.

[36] Ren-Junn Hwang and Chin-Chen Chang. An on-line secret sharing scheme for multi-secrets. *Computer Communications*, 21(13):1170–1176, September 1998.

[37] Dawei Zhao, Haipeng Peng, Cong Wang, and Yixian Yang. A secret sharing scheme with a short share realizing the (t,n) threshold and the adversary structure. *Computers and Mathematics with Applications*, 64(4):611–615, August 2012.

[38] Z. Eslami and J. Zarepour Ahmadabadi. A verifiable multi-secret sharing scheme based on cellular automata. *Information Sciences*, 180(15):2889–2894, August 2010.

[39] Jian-jie Zhao, Jianzhong Zhang, and Rong Zhao. A practical verifiable multi-secret sharing scheme. *Computer Standards and Interfaces*, 29(1):138–141, January 2007.

[40] Massoud Hadian Dehkordi and Samaneh Mashhadi. An efficient threshold verifiable multi-secret sharing. *Computer Standards and Interfaces*, 30(3):187–190, March 2008.

[41] Massoud Hadian Dehkordi and Samaneh Mashhadi. New efficient and practical verifiable multi-secret sharing schemes. *Information Sciences*, 178(9):2262–2274, May 2008.

[42] Guiqiang Chen, Huanwen Wang, Liqin Wang, and Yue Jin. A distributed multi-secret sharing scheme on the (t,n) threshold. In *2nd International Conference on Network Computing and Information Security (NCIS 2012), Shanghai, China*, pages 358–364, 2012.

[43] Chunqiang Hu, Xiaofeng Liao, and Xiuzhen Cheng. Verifiable multi-secret sharing based on LFSR sequences. *Theoretical Computer Science*, 445:52–62, August 2012.

[44] Shudong Li, Hong Lai, Wiaobo Wu, and Shuwu Jiang. Novel space efficient secret sharing for implicit data security. In *8th International Conference on Information Science and Digital Content Technology (ICIDT 2012), Jeju, Japan*, pages 283–286, 2009.

[45] Ting-Yi Chang, Min-Shiang Hwang, and Wei-Pang Yang. An improvement on the Lin-Wu (t,n) threshold verifiable multi-secret sharing scheme. *Applied Mathematics and Computation*, 163(1):169–178, April 2005.

[46] T.-Y.Lin and T.-C.Wu. (t,n) threshold verifiable multi secret sharing scheme based on factorisation intractability and discrete logarithm modulo a composite problems. *IEEE Computer and Digital Techniques*, 146(5):264–263, September 1999.

[47] Jun Shao and Zhenfu Cao. A new efficient (t,n) verifiable multi-secret sharing (VMSS) based on YCH scheme. *Applied Mathematics and Computation*, 168(1): 135–140, September 2005.

[48] Wei Chen, Xiang Long, Yuebin Bai, and Xiaopeng Gao. A new dynamic threshold secret sharing scheme from bilinear maps. In *International Conference on Parallel Processing Workshops (ICPPW 2007), Xi-An, China*, page 19, 2007.

[49] Angsuman Das and Avishek Adhikari. An efficient multi-use multi-secret sharing scheme based on hash function. *Applied Mathematics Letters*, 23(9):993–996, September 2010.

[50] Shiuh-Jeng Wang, Yuh-Ren Tsai, and Chien-Chih Shen. Verifiable threshold scheme in multi-secret sharing distributions upon extensions of ECC. *Wireless Personal Comunications*, 56(1):173–182, January 2011.

[51] Ziba Eslami and Saideh Kabiri Rad. A new verifiable multi-secret sharing scheme based on bilinear maps. *Wireless Personal Comunications*, 63(2):459–467, March 2012.

[52] Shanyue Bu and Ronggeng Yang. Novel and effective multi-secret sharing scheme. In *International Conference on Information Engineering and Applications (IEA 2012)*, pages 461–467, 2013.

[53] Shanyue Bu and Ronggeng 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.

[54] Fatih Emekci, Divyakant Agrawal, and Amr El Abbadi. [abacus.

[55] Fatih Emekci, Divyakant Agrawal, Amr El Abbadi, and Aziz Gulbeden. Privacy preserving query processing using third parties. In *22nd IEEE International Conference on Data Engineering (ICDE 2006), Atlanta, USA*, pages 27–37, 2006.

[56] Divyakant Agrawal, Amr El Abbadi, Fatih Emekci, and Ahmed 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.

[57] Mohammad Ali Hadavi, Ernesto Damiani, Rasool Jalili, Stelvio Cimato, and Zeinab 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.

[58] Brian Thompson, Stuart Haber, William G. Horne, Tomas Sander, and Danfeng 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.

[59] Mohammad Ali Hadavi and Rasool Jalili. Secure data outsourcing based on threshold secret sharing: Towards a more practical solution. In *VLDB 2010 PhD Workshop, Singapore*, pages 54–59, 2010.

[60] Mohammad Ali Hadavi, Morteza Noferesti, Rasool Jalili, and Ernesto 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.

[61] Graham Cormode and Divesh Srivastava. Anonymized data: Generation, models, usage. In *26th IEEE International Conference on Data Engineering (ICDE 2010), Long Beach, USA*, pages 1015–1018, 2010.

[62] Erin E. Kenneally and Kimberly Claffy. Dialing privacy and utility: A proposed data-sharing framework to advance internet research. *IEEE Security and Privacy*, 8(4):31–39, July/August 2010.

[63] Latanya Sweeney. K-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, October 2002.

[64] Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, October 2002.

[65] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *ACM SIGMOD international conference on Management of data (SIGMOD 2005), Baltimore, USA*, pages 49–60, 2005.

[66] Rinku Dewri and Indrajit Ray. K-anonymization in the presence of publisher preferences. *IEEE Transactions on Knowledge and Data Engineering*, 23(11):1678–1690, November 2011.

[67] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1), Mar 2007.

[68] Jeff Sedeyao. Enhancing cloud security using data anonymization; intel report, 2012. URL `http://www.intel.ie/content/www/ie/en/it-management/intel-it-best-practices/enhancing-cloud-security-using-data-anonymization.html`.

[69] Prasanna Padmanabhan, Le Gruenwald, Anita Vallur, and Mohammed Atiquzzaman. A survey of data replication techniques for mobile ad hoc network databases. *The VLDB Journal*, 17(5):1143–1164, August 2008.

[70] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *SIAM Journal of Applied Math*, 8(2):300–304, June 1960.

[71] J. Thomas and E. Schwarz. Generalized reed solomon codes for erasure correction in SDDS. In *Workshop on Distributed Data and Structures (WDAS 2002), Paris, France*, pages 75–86, 2002.

[72] Mohamed Helmy Megahed, Dimitrios Makrakis, and Bidi Ying. SurvSec: A new security architecture for reliable network recovery from base station failure of surveillance [wsn.

[73] Witold Litwin and Rim Moussa. LH: A highly available distributed data storage. In *30th International Conference on Very Large Data Bases (VLDB 2004), Toronto, Canada*, page 1289–1292, 2004.

[74] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT 2008), Melbourne, Australia*, pages 90–107, 2008.

[75] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *ACM Workshop on Cloud Computing Security (CCSW 2009), Chicago, USA*, pages 43–54, 2009.

[76] Juan A. Garay, Rosario Gennaro, Charanjit Jutla, and Tal Rabin. Secure distributed storage and retrieval. *Theoretical Computer Science*, 243(1-2):363–389, July 2000.

[77] Ari Juels and Burton S. Kaliski Jr. PORs: Proofs of retrievability for large files. In *14th ACM conference on Computer and Communications Security (CCS 2007), Alexandria, USA*, pages 584–597, 2007.

[78] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *14th European Conference on Research in Computer Security (ESORICS 2009), Saint-Malo, France*, pages 355–370, 2009.

[79] Kevin D. Bowers, Ari Juels, and Alina Oprea. HAIL: A high-availability and integrity layer for cloud storage. In *16th ACM conference on Computer and Communications Security (CCS 2009), Chicago, USA*, pages 187–198, 2009.

[80] Cong Wang, Sherman S.M. Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE Transactions on Computers*, 62(2):362–375, December 2011.

[81] J. He and E. Dawson. Multistage secret sharing based on one-way function. *Electronics Letters*, 30(19):1591–1592, September 1994.

[82] Torben P. Pedersen. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT 1991), Brighton, UK*, pages 522–526, 1991.

[83] Tang Chunming and Zheng an Yao. A new (t,n) threshold secret sharing scheme. In *International Conference on Advanced Computer Theory and Engineering (ICACTE 2008), Phuket, Thailand*, pages 920–924, 2008.

[84] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *15th Annual International Cryptology Conference (CRYPTO 1995), Santa Barbara, USA*, pages 339–352, 1995.

[85] Josh Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *8th Annual International Cryptology Conference (CRYPTO 1988), Santa Barbara, USA*, pages 27–35, 1990.

[86] Mehrdad Nojoumian, Douglas R. Stinson, and Morgan Grainger. Unconditionally secure social secret sharing scheme. *Information Security, IET*, 4(4):202–211, December 2010.

[87] Mehrdad Nojoumian and Douglas R. Stinson. Socio-rational secret sharing as a new direction in rational cryptography. In *3rd International Conference Decision and Game Theory for Security (GameSec 2012), Budapest, Hungary*, pages 18–37, 2012.

[88] Tao Zheng, Haitong Wu, Hao Wen Lin, and Jeng-Shyang Pan. Application of belief learning model based socio-rational secret sharing scheme on cloud storage. In *6th*

International Conference on Genetic and Evolutionary Computing (ICGEC 2012), Kitakyushu, Japan, pages 15–18, 2012.

[89] Mehrdad Nojoumian and Douglas R. Stinson. Social secret sharing in cloud computing using a new trust function. In *10th Annual International Conference on Privacy, Security and Trust (PST 2012), Paris, france*, pages 161–167, 2012.

[90] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science, Toronto, Canada*, pages 162–167, 1986.

[91] Chumming Tang and Zheng an Yao. Definition and construction of multi-prover zero-knowledge arguments. In *International conference on communications and mobile computing (CMC 2009), Yunnan, China*, pages 375–379, 2009.

[92] Guang Gong and L. Harn. Public-key cryptosystems based on cubic finite field extensions. *IEEE Transactions on Information Theory*, 45(7):2601–2605, November 1999.

[93] Guang Gong, Lein Harn, and Huapeng Wu.

[94] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. The star schema benchmark and augmented fact table indexing. In *1st Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2009), Lyon, France*, pages 237–252, 2009.

[95] Hotea Solutions. TPC-H, 2015. URL `http://www.tpc.org/tpch/`.

[96] Hotea Solutions. TPC-DS, 2015. URL `http://www.tpc.org/tpcds/`.