Secure Cloud-of-Clouds Storage with Space-Efficient Secret Sharing

Ahad Niknia*, Miguel Correia†, Jaber Karimpour*
*Department of Computer Science, University of Tabriz, Tabriz, Iran
†INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

***Abstract*—Cloud storage services are top-rated, but there are often concerns about the security of the files there stored. Clouds-of-clouds or multi-clouds are being explored in order to improve that security. The idea is to store the files in several clouds, ensuring integrity and availability. Confidentiality, however, is obtained by encrypting the files with block ciphers that do not provide provable security. Secret sharing allows distributing files among the clouds providing information-theoretic security/secrecy. However, existing secret sharing schemes are space-inefficient (the size of the shares is much larger than the size of the secret) or purely theoretical. In this paper, we propose the first practical space-efficient secret sharing scheme that provides information-theoretic security, which we denominate PRactical Efficient Secret Sharing (PRESS). Moreover, we present the Secure CloUD storage (SCUD) service, a new cloud-of-clouds storage service that leverages PRESS to provide file confidentiality. Additionally, SCUD provides data integrity and availability, leveraging replication.**

***Index Terms*—Secret Sharing, Space Efficiency, Cloud Storage, Cloud Services**

I. INTRODUCTION

Cloud storage services like Amazon S3, Microsoft OneDrive, Google Drive, and Dropbox are extremely popular, both among companies and private users. However, the fact that with such services data is stored outside of the premises of the company/user, often raises concerns about data security, in terms of confidentiality, integrity, and availability [1], [2].

Clouds-of-clouds or *multi-clouds*, the idea of using several clouds to mask faults in some of them, is getting a lot of traction [3] with offers like Google Anthos[1] and startups like Vawlt[2] or Multcloud.[3] Clouds-of-clouds are being explored in order to improve that security [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. The approach consists in storing the data in a set of different cloud storage services, directly guaranteeing integrity and availability. However, *confidentiality* is obtained by encrypting the files, typically with *block ciphers* such as AES in CBC mode [5], [6], [8], [10], [12], [13], [18].

Block ciphers and, more generically, symmetric encryption are widely adopted today. However, their security is problematic. Katz and Lindell provide a proof that "any *perfectly-secret encryption scheme* must have a key space that is at least as large as the message [i.e., plaintext] space" [19]. The best-known of such schemes is the one-time pad. However, such a scheme is unusable in practice, because it requires solving the same problem we have in the first place: protecting the secrecy of data of a certain size. Therefore, block ciphers and other symmetric encryption schemes do not provide perfectly-secret encryption, but approximations that allow fixed-size keys (e.g., 256 bits). The practical consequence of using approximations is that these schemes are often vulnerable to attacks (e.g., known-plaintext attacks, chosen-plaintext attacks, differential cryptanalysis) or become insecure with time. For example, the previous block cipher security standard, DES, is no longer considered secure, and the current standard, AES, has been shown to be vulnerable to a few (non-critical) attacks that reduce its strength [20]. Moreover, if quantum computers become available, they may lead to an extreme reduction on the time to brute-force encrypted data by running the Grover algorithm [21]. Therefore, although using block cipher schemes like AES today is arguably a secure option, this may not be the case in the future.

Secret sharing is a set of methods to distribute a secret among a set of participants [22], [23]. Specifically, a (t, n) secret sharing like Shamir's [22] generates n shares of the secret in such a way that any t –but no less– can be used to recover the secret. On the contrary of block ciphers, many secret sharing schemes provide *information-theoretic security* [24], [25], i.e., they provide the same secrecy guarantees as a *perfectly-secret encryption* scheme; their security does not rely on approximations [26]. Secret sharing can be used to obtain confidentiality by breaking a file in shares and storing each share in a cloud of a multi-cloud [7], [9], [27].

Despite the security offered by this combination of secret sharing and cloud-of-clouds, most information-theoretic secure secret sharing schemes are *space-inefficient* in the sense that they generate shares of a size similar to the size of the secret. This means that storing a secret (a file) of size $|S|$ using such a scheme requires $n \times |S|$ storage space, which costs an average of n times the cost of storing the file in a single cloud, as storage space used in public clouds is paid.

There are a few solutions to this problem in the literature. A first class of solutions are multi-secret sharing schemes, i.e., schemes that improve space efficiency by sharing several secrets together [28], [29], [30], [31]. However, this approach is not suitable for cloud storage as files are typically stored one by one in the cloud, not in groups of several files of the same size. Moreover, these schemes reveal some information

Corresponding author: J. Karimpour (email: karimpour@tabrizu.ac.ir).
[1]https://cloud.google.com/anthos/
[2]https://vawlt.io
[3]https://www.multcloud.com/

[32].

A second set of solutions provide only *computational security* [31], so they raise similar concerns as those of block ciphers. A space-efficient scheme proposed by Krawczyk encrypts the secret (or file) with a block cipher and a random secret key, breaks the encrypted secret into redundant parts using an erasure code scheme, then generates shares of the secret key using an information-theoretic secure secret sharing scheme [33]. This scheme is used in cloud storage systems like DepSky and SCFS [5], [6]. There are also *deduplication schemes* that allow sparing much storage space, but that again use block ciphers to encrypt the files [34], [35], [36], [37], [15].

In this paper we seek to bring together the information-theoretic security of secret-sharing with space efficiency. The optimal *storage space bound* for secret sharing (the optimal share size) is $\frac{|S|}{t}$ as we need at least $t \times \frac{|S|}{t} \geq |S|$ bits to recover the secret S, assuming that the file cannot be compressed. Parakh and Kak presented a recursive scheme that has a near-optimal share size of $\frac{|S|}{t-1}$ [32]. They prove that the scheme provides information-theoretic security, but their scheme is not practical as it has several limitations: it works for reasonably small secrets, not files; the presentation of the dealing algorithm has obscure parts; the recovery scheme is not fully defined (the interpolation scheme must be different from Shamir's and this is not even discussed), and the main dealing algorithm is limited by the relation between the number of the data pieces and the number of cloud servers (share numbers) which are assigned to store the data shares. An obvious consequence of the scheme being incompletely defined is that there is no implementation available.

In this paper, we propose the first *practical space-efficient secret sharing scheme that provides information-theoretic security*, which we denominate *PRactical Efficient Secret Sharing* (PRESS). The scheme is inspired by Parakh and Kak's scheme but solves the limitations that make it unpractical. PRESS is fully defined and implemented. Utilizing *secret sharing* in *data security* and its boundaries, limitations, and the corresponding computation and storage costs are discussed extensively in [38], [39], and [40].

Moreover, we present the *Secure CloUD storage* (SCUD) service, a new cloud-of-clouds storage service that leverages our secret sharing scheme to provide file confidentiality. The service also provides data integrity and availability by storing files in several clouds. SCUD provides a block storage service similar to DepSky's, in which files are stored in several clouds. However, DepSky and similar systems use block ciphers to encrypt files, while SCUD uses PRESS that provides information-theoretic security.

We did an experimental evaluation of SCUD with files stored in Amazon S3 in 8 locations worldwide and with clients in Amazon EC2 in other 4. We concluded that SCUD, leveraging PRESS as its backing, can provide provable security for storing data in clouds in a space-efficient way, with a performance penalty of around double of using a single cloud (instead of the expectable 4x cost of replicating a file in 4 clouds). Therefore, this combination seems to be effective for practical scenarios of critical data outsourcing.

The paper is organized as follows. Section II presents additional background on secret sharing and additional related work. Section III presents the PRESS secret sharing scheme. Section IV presents the SCUD cloud-of-clouds storage service. Section VI covers the implementation of both the secret sharing scheme and the cloud storage service. The results of the experimental evaluation are presented in Section VII. Finally, Section VIII concludes the paper.

II. Background and Related Work

This section provides additional background on secret sharing and further discusses related work on cloud storage.

A. Secret Sharing

Secret sharing is a method for a dealer to break a secret in shares and distribute them to a group of parties. Secret sharing is an important tool in cryptography that is used in many security protocols [41]. A secret-sharing scheme for A is a method by which the dealer distributes shares to the parties such that: (1) any subset in A can reconstruct the secret from its shares, and (2) any subset not in A cannot reveal any partial information about the secret. For that reason, the term threshold secret sharing is also used to mean this kind of scheme. Threshold schemes, including secret sharing and threshold signatures and others, are well suited to applications in which a group of mutually suspicious parties with conflicting interests has to cooperate [22], [42], [43], [44]. This kind of scheme has been utilized in cloud and multi-cloud computing and storage lately. Shamir's threshold secret sharing scheme, although having appeared concurrently with Blakley's [23], is regarded as the basis for many subsequent schemes, including the scheme by Parakh and Kak's in which ours is inspired. Therefore, next, we present Shamir's scheme, then Parakh and Kak's. In the end we briefly discuss Thien and Lin's secret image sharing scheme [45].

1) Shamir's Scheme: The scheme is based on polynomial interpolation. Given t points in a 2-dimensional plane $(x_1, y_1), ..., (x_t, y_t)$ with distinct x_i's, there is one and only one polynomial $q(x)$ of degree $t-1$ such that $q(x_i) = y_i$ for all i. Without lack of generality, consider that the secret D is an integer number (i.e., $D \in \mathcal{Z}$). To partition D into D_i shares, choose a random $t-1$ degree polynomial $q(x) = a_0 + a_1x + ... + a_{t-1}x^{t-1}$ in which $a_0 = D$, and evaluate $D_1 = q(1), ..., D_i = q(i), ..., D_n = q(n)$. Given any subset of t of these D_i values (indicated with their indices as (D_i, i)), it is possible to interpolate the coefficients of $q(x)$ and obtain the value $D = q(0)$. Moreover, having any subset of $t-1$ of these values does not reveal (it is not sufficient to calculate) any information about D (perfect privacy).

Consider a dealer who is in charge of sharing a secret D between n participants for a (t, n) threshold scheme. Shamir's scheme has three steps:

1) *Generating the parameters*: Given an integer D, select a prime number p bigger than both D and n. Each participant gets an index i and p.
2) *Sharing shares of the secret with the participants*: The coefficients $a_1, ..., a_{t-1}$ of $q(x)$ are randomly picked

with a uniform probability distribution over the integers in $[0, p)$, and $q(x) = D + a_1 x + a_2 x^2 + ... + a_{t-1} x^{t-1} (mod\, p)$ is constructed. Then, the value of $q(i)$ for $i = 1, 2, ...n$ is calculated and transmitted to each participant i using a secure channel.

3) *Secret reconstruction*: t participants provide their shares, then interpolation is used to reconstruct the secret D.

2) Parakh and Kak's Scheme: One of the significant obstacles in secret sharing is the storage space. The optimal share size for these schemes is $\frac{|S|}{t}$. In 2011, Parakh and Kak presented a threshold secret sharing technique with the aim to reduce the storage size, making it close to the optimal bound [32]. Their proposed algorithm has a share size of $\frac{|S|}{t-1}$.

Their scheme operates as follows. Consider a secret d, n participants, and a threshold t. In the *dealing phase*, first the dealer splits d into $t-1$ pieces of data. Then, recursively, polynomials of degree 1 to $t-1$ are created and sampled for the following recursion. Finally, the last polynomial of degree $t-1$ is sampled in n points, each one becoming a share of the secret.

In the *reconstruction phase*, t shares are needed to interpolate the polynomial and reverse the dealing steps. Reversing the process uses the interpolated polynomial coefficients in each round as input points to repeat the new polynomial process as the next round. The constant terms of the polynomials in all rounds are accumulated to remake the desired secret. This scheme relies on modular calculus using a base equal to a prime number that must be greater than the secret. To overcome the problems of dealing with big prime numbers, the secret has to be split into smaller parts (e.g., in 2-digit parts), which are handled as individual secrets by the secret sharing dealing algorithm. The dealing algorithm works with secret pieces. There are a fixed number of these pieces (equals to $t-1$ and then their size is not flexible either. The size of these pieces defines the required storage size and the system's storage ratio. Any try to change their size would change the secret size and implementation issues, which are not practical. The PRESS also alters this scheme to a flexible approach on the secrets and pieces size and number.

Despite the benefits of Parakh and Kak's scheme, the scheme itself and its presentation have several limitations. First, the space efficiency is limited due to the dependence of the internal parameter P on the number of shares n (see Section III-A). Second, the paper does not present the secret reconstruction scheme (specifically the interpolation). Finally, the paper does not explain how it should be implemented in practice or evaluates its performance. Our scheme (PRESS) solves these limitations. Moreover, our paper is not a theoretical paper about a scheme, but a systems paper about a storage service.

3) Thien and Lin's Scheme: Thien and Lin introduced a scheme to share an image as a set of parts that they call shadow images [45]. On the contrary of Shamir's scheme and of our own work, their solution is not generic, i.e., does not allow sharing generic data, only images. Their approach is based on sharing the grey values of pixels of an image divided into several sections of pixels. It splits the original image into n shadow images that, regarding the threshold value of t, will make the result shadows size by the side of $|\frac{D}{t}|$ for the image of size D. So, the overall file size will be $n \times |\frac{D}{t}|$ for an image of size $|D|$ that is pretty close to the optimal, similarly to our approach. As mentioned, their scheme is targeted at image sharing, as it relies on sharing the grey values for pixels. Therefore, it needs permutations on the original image before it does the sharing process. To achieve better availability and integrity, the value of the t needs to be defined carefully regarding the size of n. The overall storage size will not be much bigger than the original image size. As this approach is limited to image sharing, they recommended hiding the resulted shadow images in different host images, which explicitly has meaningfully negative effects on the overall computation and storage costs.

B. Cloud Storage Services

Cloud storage services are popular, but security (confidentiality, integrity, and availability) and dependability (availability, reliability, integrity, maintainability) remain important concerns [2], [1], [46], [47]. The idea of using replication in several clouds for improving security and dependability first appeared in RACS [4], and DepSky [5]. Since then, many others appeared, providing similar guarantees, in some cases with a more practical programming interface [6], [7], [8], [10], [12], [13], [14], [15], [17], [18]. Interestingly, Amazon as the major cloud provider, also proposed replicating files in several regions / data centers to improve the availability of their S3 service [11].

These solutions trivially achieve *integrity* and *availability*, by merely having the files stored in different clouds, i.e., in the systems of different cloud service providers or in various data centers. In relation to *confidentiality*, three basic approaches are being followed. First, some practical and research systems simply encrypt the files storing them in the clouds [12], [13]. Second, systems like DepSky, SCFS, and Charon [5], [6], [18] use a variation of Krawczyk's scheme [33], i.e., a non-information-theoretically secure secret sharing scheme. Third, systems like Belisarius and fVSS [7], [9], use secret sharing schemes that provide information-theoretic security but are not space-inefficient. Table I summarizes the approaches on data storage for a representative set of systems.

Space inefficiency is one of the main problems of using secret sharing schemes for data confidentiality, as in Belisarius and fVSS (see Table I). Recursive methods based on Shamir's scheme use several polynomials to reduce the space usage [48], [32]. Our algorithm uses secret sharing for data distribution and is based on recursive algorithms to avoid space inefficiency.

III. Practical Efficient Secret Sharing – PRESS

PRESS is based on Shamir's scheme and on the idea of recursively building upon polynomial interpolation and sampling from Parakh and Kak [32]. PRESS aims to make practical use of such recursion for storing files efficiently in terms of the used storage space. The scheme has two phases: dealing with data, i.e., breaking it into shares, and reconstructing the data from a desired number of shares.

TABLE I
COMPARISON BETWEEN CLOUD STORAGE SYSTEMS IN THE LITERATURE AND OURS

System	Data handling approach	Confidentiality approach	Secret sharing role	Secret sharing scheme	Ref.
RACS	data replication with erasure coding	none	none	none	[4]
DepSky	data replication with erasure coding	data encryption	secret key storage	Shamir / Krawczyk	[5]
SCFS	data encoding	data encryption	secret key storage	Shamir / Krawczyk	[6]
Hybris	data replication by erasure coding	data encryption	none	none	[8]
Belisarius	data secret sharing	secret sharing	confidentiality and availability	Shamir	[7]
fVSS	db entry sharing with secret sharing	secret sharing	confidentiality and availability	Shamir / fVSS	[9]
SCUD	secret sharing	secret sharing	confidentiality and availability	*PRESS*	this

The PRESS scheme is based on two main parameters:

- n – the number of shares to obtain;
- t – the number of shares needed to recover the secret S.

Moreover, the scheme depends on an internal parameter P. P is a key parameter as it directly affects the space-efficiency of a system that uses PRESS. Therefore, it has to be adjusted carefully. Section VII-B1 discusses the tuning of the scheme's parameters, including P.

A. Dealing Phase

The dealing phase is implemented by function *DealSecret()* that works as defined in Algorithm 1. To share a secret S, the dealer first splits it into P data pieces $s_1, s_2, s_3, ..., s_P$ (Step 1). A prime number greater than any of the pieces is picked (Step 2). Then, the dealer randomly and uniformly chooses a number $a_1 \in \mathcal{Z}$ and generates a 1st degree polynomial $f_1(x) = a_1x + s_1$ (Step 3).

The polynomial $f_1(x)$ is sampled in two points D_{11} and D_{12} as the shares of s_1 (Step 4). These resulting numbers (D_{11} and D_{12}) are used as coefficients to generate the second round polynomial $f_2(x)$ for the second secret piece s_2: $f_2(x) = D_{12}x^2 + D_{11}x + s_2$ (Step 5.a). Then, $f_2(x)$ is sampled in three points to be used as coefficients in the next round for the next secret piece's corresponding polynomial.

This process is repeated recursively for all the secret pieces (Step 5.b.i) except the two lasts: s_{P-1} and s_P. The last polynomial $f_{P-1}(x)$ is generated in the same way as the previous, but sampled in $t-1$ points instead of P (t is the threshold number) (Step 5.b.ii). These sampling points define the degree of the next round's polynomial as its coefficients. Also, to have a polynomial that works with t points at the next round (that corresponds to the threshold number of given points in the reconstruction phase), we need to have a polynomial of degree $t-1$, which requires $t-1$ coefficients sampled from f_{P-1}.

These steps handle one of the main defects of Parakh and Kak's algorithm. In their algorithm, P has a fixed relation with t: $P = t-1$. Then for a system with the appointed number of participants and thresholds, all the other parameters are fixed, and there is no flexibility for change. In contrast to their algorithm, we split the secret freely and with more flexibility in number and size and not fixed to the threshold number, i.e., in our case $P \leq t-1$.

Finally, to share s_P the dealer needs to generate $f_P(x)$, which is a polynomial of degree $t-1$. Then it samples that polynomial in n points as $(i, D_i); i = 1, 2, ..., n$ (Step 5.b.iii).

Algorithm 1 Algorithm for sharing a secret S

```
function DealSecret(S){
  1) Split secret $S$ to desired number of pieces ($P$), as:
     $s_1, s_2, .., s_P$
  2) Choose prime number $p$ so that,
     ($\forall s_i, i \in [0, ..., P] : p > s_i$) and ($p > n$)
  3) Choose random number $a_1 \in \mathcal{Z}_p$ and generate the
     polynomial $f_1(x) = a_1x + s_1 \ (mod\ p)$
  4) Sample $f_1(x)$ in two points $D_{11} = f_1(1)$ and $D_{12} =$
     $f_1(2)$ as two shares for $s_1$
  5) Do for the $i \in [2, ..., P]$:
     a) Generate polynomial $f_i(x)$ as:
        $f_i(x) = \sum_{j=1}^{i} (D_{(i-1)j}x^j) + s_i \ (mod\ p)$
     b) Sample $f_i(x)$ and make shares of $s_i$ as follows:
        i) If $i \leq P-2$, sample it in $i+1$ points.
        ii) If $i = P-1$, sample it in $t-1$ points.
        iii) If $i = P$, sample it in $n$ points as final shares.
  6) return($\{f(i), i = 1, ...n\}$)
}
```

These points are the shares of the secret S, which will typically be sent to the n participants to be stored (Step 6).

B. Reconstruction Phase

Secret reconstruction is an altered version of Shamir's scheme plus recursive interpolations. This recursion is designed to revert the dealing phase and produce the secret pieces $s_1, s_2, s_3, ..., s_P$. The secret is reconstructed by concatenating these generated pieces. This process is implemented by function *ReconstructData()* shown in Algorithm 2.

The algorithm needs t shares to be able to interpolate and find out the polynomial and its coefficients. The free term of the polynomial is the last piece of the secret, and the coefficients regarding their order are the input points for the next round of interpolation (Step 1).

The polynomial resulting from Step 1 has degree $t-1$, with $t-1$ coefficients and the last secret piece (the P^{th}) as the free term. In the next step, these coefficients are used to interpolate the next polynomial. The second polynomial has degree $P-1$ and the $(P-1)^{th}$ piece of the secret as the free term. This recursive process is repeated to reveal the secret piece by piece (Steps 2-3).

Secret sharing schemes derived from Shamir's are based on linear interpolation of a set of points, i.e., of k shares, to recover the secret (the free term of the polynomial). However,

Algorithm 2 Reconstruction Algorithm

```
function ReconstructData(data shares $\{Ds_i\}$){
    1) Interpolate the first polynomial ($f_P(x)$) of degree $t-1$ by at least $t$ shares $\{(i, Ds_i), i = 1...t\}$ as:
       $f_P(x) = \sum_{j=1}^{t-1} (D_{(P-1)(j)} x^j) + s_P \ (mod\ p)$
       $s_P = f_P(0)$, as the free term.
    2) For all $i \in [P-1, ..., 1]$:
       use the coefficient of the previous step's polynomial as points of the form $(j, D_{ij}), 1 \leq j \leq i$ to interpolate $f_i(x)$ of degree $i$:
       $f_i(x) = \sum_{j=1}^{i} (D_{ij} x^j) + s_i \ (mod\ p)$
       which $s_i = f_i(0)$.
    3) The desired secret $S$ is the combination of the $s_1, s_2, ..., s_P$
    4) Reversing the process, it is possible to get to the original data which $S$ was made from.
    5) return($S$)
}
```

typical Lagrange interpolation approaches cannot interpolate polynomials with the corresponding coefficients (i.e., steps 1-2 of the algorithm). In contrast to the usual schemes, our recursive scheme specifically exploits the coefficients of the interpolated polynomial in addition to the free term. Parakh and Kak do neither present a scheme to obtain these coefficients nor discuss or consider this complication.

There are different ways to get these coefficients, but dealing with the much data (with files) demands an efficient solution. Moreover, the method must always yield exact values without any approximations. The solution we propose is a direct interpolation. This method transforms the interpolation problem into the problem of solving a system of $n+1$ linear equations. The Gaussian elimination method can solve this problem efficiently, as it is the algorithm of choice for solving dense linear systems of equations [49]. It can be implemented using matrix calculus.

This method operates based on the matrix operations (e.g., multiplication and inversion), which are considered complex. Therefore, to avoid this complexity, it is necessary to transform the matrices into diagonal matrices. This transform needs some operations on entries and vectors of the matrix (e.g., pivoting) that can be done efficiently using primitive matrix operations, which are based on simple addition and multiplication. We use these techniques to solve the system of equations and get the coefficients.

C. Performance

Shamir's dealing and reconstruction algorithms work in a single round: obtain a single polynomial and do a single interpolation to reconstruct the secret, respectively. Like Parakh and Kaks, our algorithms do P of such operations, i.e., have time complexity $\Theta(P)$.

IV. Secure CloUD storage – SCUD

SCUD is designed to work as a cloud-of-clouds data storage service. In terms of interface and use cases, it is similar to commercial services like Amazon S3 and Google Cloud Storage and to research cloud-of-clouds storage services like DepSky or Belisarius. In terms of guarantees, it aims to provide stronger data security guarantees (information-theoretic security), plus integrity and availability, with space efficiency. For that purpose, it leverages the PRESS scheme.

SCUD mainly writes and reads files to/from a set of clouds, using PRESS to protect the confidentiality of the files. SCUD uses the PRESS dealing algorithm for breaking the file into shares to be written in the different clouds. It uses the reconstruction algorithm to recover the file from the shares that reads from a set of clouds. The details of these algorithms and components are described in the following sections. The system parameters are summarized in Table II for reference purposes.

TABLE II
System Parameters

Parameter	Meaning
CSP_i	ith-cloud service provider
n	number of participants/clouds
t	secret sharing threshold
f	fault bound, the maximum number of faulty clouds
S	secret
s_i	secret data pieces $S = \{s_1, s_2, ..., s_P\}$
P	number of secret pieces
P_s	size of the data pieces $P_s = \lvert s_i \rvert$ (in bytes)
fn	file name
F_i	ith-file share of file F
Du_z	file data units $F = \{Du_z\}$
I	size of the data units $I = \lvert Du_z \rvert$ (in bytes)
f_{ij}	ith-data share of the secret S_j
A	size of data shares $A = \lvert f_{ij} \rvert$ (in bytes)
R_S	storage ratio, total share size divided by original data size
p	prime number used in PRESS
Pu, Pr	public / private key pair for digital signatures
M_i	meta-data for file share number i

A. Architecture and System Model

Figure 1 shows the architecture of a system running SCUD. This system involves two main players: the *users* that use the cloud to store files, and the *cloud service providers* (CSPs) that run clouds. The set of clouds forms a cloud-of-clouds or multi-cloud. The system uses a single type of service provided by the CSPs: storage.[4] These services are passive in the sense that they do not run any code on behalf of the users and only store data. They provide a well-defined interface, typically REST over HTTPS.

The SCUD software is the client agent represented in the figure. That agent includes the PRESS code. Both SCUD and PRESS are software libraries that can be used to implement applications. The client agents have access to the data stored in the clouds based on their user's access credentials. The format of these credentials may depend on the CSP. The interface of each cloud may also be specific to its CSP, so the client software has adapters to access different clouds. The clients have to manage the operations over files (e.g.,

[4]Commercial clouds offer many more services. At the time of this writing, the Amazon AWS console lists 106 services divided in to 19 categories. In the experiments, we also used a second service to run the users/clients: EC2.

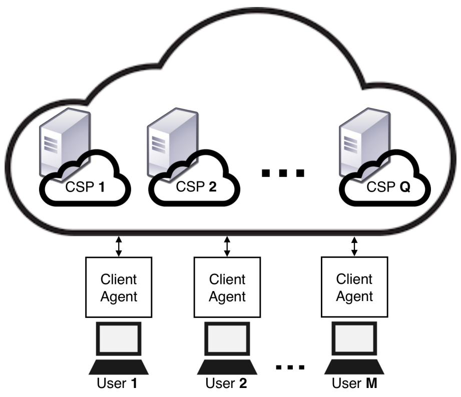

Fig. 1. Architecture of SCUD

read, write), their meta-data (e.g., name), and access control (e.g., set permissions). For simplicity of presentation, in the algorithms, we consider that files have a single version, but this limitation can be trivially removed by concatenating the version in the name or making it part of the meta-data.

We assume that the system is asynchronous, i.e., that there are no time bounds on processing or communication. We also assume that the clients and the clouds communicate over secure channels (e.g., HTTPS).

SCUD is based on a few main parameters:

- n – the number of clouds, that is also the number of shares in which every file is broken;
- f – the maximum number of faulty clouds;
- t – the number of shares needed to recover the files.

Files/data are stored in a set of n clouds, one share of the file per cloud. We assume at most f of these clouds can be faulty, i.e., may individually or in collusion attempt to disclose the contents of files stored in the system, not reply to requests, send corrupt answers, and other arbitrary behaviors. These faults are usually designated Byzantine or arbitrary faults [46], [50]. We denominate f the *fault bound*.

To avoid f faulty clouds from being able to disclose the content of files, the parameter t must be set to $t > f$. Moreover, to avoid that the faulty clouds block the recovery of files, we must set n to $n \geq t + f$. Examples of two instantiations of these parameters are:

- $f = 1$, $t = 2$, $n = 3$
- $f = 2$, $t = 3$, $n = 5$

The authentication and access control provided by the different clouds play an important role in protecting the confidentiality of the data. In fact, if an adversary can download the shares from the clouds, he can reconstruct the file simply by applying the PRESS reconstruction scheme, which we assume is publicly known. However, authentication and access control mechanisms are employed for decades in computing systems [51] and are also well established in the cloud computing domain [52]. Therefore, we assume that only authorized users can get t or more shares to form any file.

We assume the existence of a secure digital signature scheme, e.g., the Elliptic Curve Digital Signature Algorithm (ECDSA). Notice that such schemes do not have to rely on pseudorandom permutations as block ciphers do [19]. Moreover, the secrecy of the data stored in the cloud would not compromised if this scheme was attacked, as it is used only to assess the integrity of the data returned. This scheme provides two functions, *SignData()* and *Verify()*, respectively to sign a block of data and to verify a signature.

Algorithm 3 File Writing protocol

```
function WriteFile(file F){
    1) Decompose $F$ in data units $\{Du_z\}$ of size $I$
    2) Transform data units $\{Du_z\}$ into secrets $\{S_j\}$
    3) $\forall S_j : f_{ij} =$ DealSecret($S_j$)
    4) $\forall i \in [1, ..., n]$ :
        a) $F_i =$ ComposeFileShare($\{f_{ij}, \forall j\}$)
        b) $sF_i =$ SignData($F_i, Pr$)
        c) $M_i =$ MetaData($F_i, sF_i, Pu$)
    5) $\forall i \in [1, ..., n]$ :
       ConcurrentUploadFile($F_i, M_i, CSP_i$)
    6) return($\{M_i\}$)
}
```

B. Write Operation

The first protocol we present is the write operation. The protocol has two sequential phases. The first phase prepares and shares the file using PRESS; the second stores the file in the CSP storage services.

As explained before, the system is mostly client-side, so the agents are responsible for the whole procedure. The write protocol is presented in the Algorithm 3 (function *WriteFile()*).

The first phase is implemented by Steps 1 to 3 of the algorithm. First, the file is broken down into data units Du_z of a configurable size I, e.g., 2 or 4 bytes (Step 1). The data unit size ($I = |Du_i|$) is an important factor for the performance of SCUD. The next step consists of defining that each of these data units is a secret S_j to be processed by PRESS, possibly involving some kind of format transformation if that is needed (Step 2). Each secret S_j is shared using the PRESS algorithm for sharing a secret (Algorithm 1), which returns n data shares $\{f_{ij} : i \in [1, ..., n]\}$ per secret (Step 3).

The second phase starts with Step 4. The data shares are aggregated to compose file shares $\{F_i : i \in [1, ..., n]\}$ (Step 4.a) to be uploaded to the CSPs (function *ComposeFileShare()*. As a result, each file F is broken down in n file shares F_i. Each file share is signed using the digital signature algorithm and the private user's private key Pr (Step 4.b). A set of meta-data about the file (e.g., its name), including the signature and the public key Pu (corresponding to Pr), is prepared to be sent to the clouds (Step 4.c). It is noticeable that meta-data is not created and stored for each part of the data. However, it is generated and stored for each part of a file (which our experiments demonstrate that it is less than 500 bytes per file part and is perfectly irrelevant when compared with the corresponding file size). Finally, all file shares F_i and the corresponding meta-data M_i are uploaded to the CSPs, one share per CSP (Step 5). Notice that the *ConcurrentUploadFile()* function tries to upload shares to all the CSPs.

C. Read Operation

The second protocol implements the *read* operation. The client agent is also responsible for reading data shares from CSPs and utilizing the secret reconstruction algorithm (Algorithm 2, function *ReadFile()*) to reveal the secrets and retrieve the requested corresponding file.

Algorithm 4 File Reading protocol

```
function ReadFile(fn){
  1) GetMetaData($fn, CSP_i$)
  2) set $Pu$ to the $Pu$ that comes in $f+1$ items $M_i$
  3) $\forall i \in [1, ..., n]$ :
     a) $(F_i, Pu)$ = ConcurrentDownloadFile(fn, $CSP_i$)
     b) Cancel upon $|\{$Verify($F_i, Pu$)=$True\}| \geq t$
  4) $\forall i$ : Decompose $F_i$ to $\{f_{ij}, \forall j\}$
  5) $\forall j : S_j$ = ReconstructData($\{f_{ij}, i \in [1, ..., t]\}$)
  6) Transform $\{S_j\}$ to data units $\{Du_z\}$
  7) $F$ = ComposeFile($\{Du_z : \forall z\}$)
  8) return($F$)
}
```

The algorithm (Algorithm 4) takes the filename *fn* as input. Reading a file starts with accessing the corresponding meta-data $\{M_i\}$ (Step 1). Function *GetMetaData()* retrieves the meta-data from at least $n - f$ CSPs, but does not block trying to retrieve it from all, similarly to what we explained for function *ConcurrentUploadFile()*. Faulty clouds may also return corrupted meta-data, so the public key Pu is set to the single one that can come from at least $f + 1$ different clouds, as we assume no more than f are faulty (Step 2).

Then the agent starts to download file shares $\{F_i\}$ concurrently in order to optimize the download time (Step 3.a, function *ConcurrentDownloadFile()*). Again it is not possible (nor needed) to download shares from all the clouds, so the download process finishes when t correctly signed shares were downloaded (Step 3.b).

Being enough file shares available, they are decomposed into secret shares (Step 4), the secrets that compose the file are reconstructed using PRESS (Step 5). Then, the secrets are transformed into data units (Step 6), and the file is reconstructed *ComposeFile()* (Step 7). Steps 4 to 5 essentially do the opposite of the first phase of Algorithm 3. Figure 2 explains how SCUD and PRESS handle the write and read protocols regarding a single file as the input.

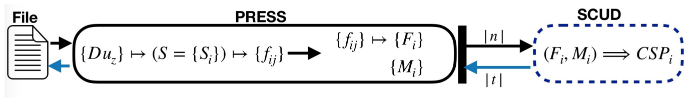
File
PRESS
$\{Du_z\} \mapsto (S = \{S_i\}) \mapsto \{f_{ij}\}$
$\{f_{ij}\} \mapsto \{F_i\}$
$\{M_i\}$
$|n|$
$|t|$
SCUD
$(F_i, M_i) \Longrightarrow CSP_i$

Fig. 2. Write and Read protocols with SCUD and PRESS

D. Other Operations

Full implementation of SCUD requires other operations such as sharing a file with another user, revoking access to a file from a user, or deleting a file. These operations are simple to implement by accessing the APIs provided by the different CSP storage services. The main challenge is supporting a diversity of APIs, which is a topic for the next section.

V. Security Analysis

This section shows that our approach satisfies the security property that common to all secret sharing algorithms [22]:
Secretness: *Knowledge of any $t-1$ or less file shares leaves the file completely undetermined.*

Proof: File F is reconstructed from the file shares $\{Du_z : \forall z\}$ by function *ComposeFile* in Algorithm 4, Step 7. These shares $\{Du_z\}$ are obtained from $\{S_j\}$ that are produced by the *ReconstructData()* function (Algorithm 4, Steps 5-6).

Function *ReconstructData()* is the function that deals with the interpolations. The function starts with the interpolation of the polynomial of degree of $t-1$ from the t shares, i.e., from the function input $\{f_{ij}, i \in [1, ..., t]\}$ (Algorithm 2, Step 1). The Interpolation Theorem states that given t points in a 2-dimensional space (shares in our case), there is one and only one polynomial of degree $t-1$ that passes on those points [22]. A corollary is that any number of shares ($\{f_{ij}\}$) lower than t cannot return the correct interpolation and the desired polynomial (of degree $t-1$); in fact, there would be infinite polynomials of degree $t-1$ that pass on those points.

Therefore, given less than t points it is not possible to proceed with the recursive interpolation algorithm and retrieve the section $\{S_j\}$ of the file F, so it is also not possible to obtain the file F itself. As long as these $\{f_{ij}\}$ are decomposed of $\{F_i|\forall j\}$ (Algorithm 4, Step 4), there is an one-to-one relation between the number of $|\{F_i\}|$ and the number of $|\{f_{ij}, \forall j\}|$, and thus any $t-1$ or less number of file shares F_i could not provide us the required number of $\{f_{ij}, i \in [1, ..., t]\}$ and thus cannot give enough information to retrieve the original file F.

VI. Implementation

PRESS and SCUD are implemented in Java, essentially as two packages that provide two libraries. A third package was developed to provide data integrity control using elliptic curves digital signatures.

We implemented a few simple applications in Java to demonstrate and evaluate the performance of SCUD. These applications use the SCUD library, which itself uses the PRESS library. The main application executes the system with predefined scenarios and measures the system outcome and performance depending on the evaluation parameters.

To develop SCUD, we leveraged Apache jclouds [53]. jclouds is an open source multi-cloud Java toolkit that supports the creation of portable cloud applications with full cloud-specific feature control. The main advantage is that it provides a single interface to clouds with heterogeneous interfaces, greatly simplifying the effort of implementing software like SCUD.

This toolkit is used in both the write and read operations. SCUD is configured with a list of the cloud storage endpoints of the services that are supposed to store the file shares. These servers are selected automatically regarding the system's configurations. To increase the speed, uploading and downloading the file shares are implemented as concurrent threads.

Algorithm 3 produces a set of file shares that it signs (Step 4.b). Each file share is signed using ECDSA with SHA256. These algorithms are chosen due to their level of security that is adequate for future use, good performance, lower power consumption, and memory and bandwidth savings [20], [54]. Also, the ECDSA algorithm is used for generating the key pairs (public and private keys) with 256-bit keys, which is a value recommended by ENISA for future use [20]. Our prototype uses Java SE JDK 8 standard security platform (Cryptography Architecture, JCA) to implement the related components [55]. The data unit size is set to $I = 4$ bytes (will be discussed next).

VII. Experimental Evaluation

This section presents an evaluation of the system in two steps. PRESS and SCUD are designed to be general and configurable, so their performance depends on the initial and running configurations. Therefore the first question we aim to answer is: what should be the systems' parameters for good performance? (Section VII-A)

The second step consists of answering the question: how is the whole system's performance? We do an analysis of the write (share, sign, upload) and read (download, verification, and retrieval) operations (Section VII-B) to assess the system performance.

A. Configuration and Cost

Let us define *storage ratio* as the ratio between the used storage space and the input data size. The storage ratio is given by $R_S = \frac{n \times A}{I}$, where n is the number of participants/clouds, I is the input data unit size $I = |Du_z|$ (4 bytes in our prototype), and A is the required storage size for each share in bytes, i.e., $A = |f_{ij}|$. These three parameters I, A, and n are positive integers, but the storage ratio is a positive float.

The storage ratio is not a fixed parameter of the system (SCUD). As it will become clear soon, changing one parameter does not have a uniform and monotonous effect on the ratio; it may increase or decrease the ratio depending on the other parameters, and the ratio vacillates regarding the parameters. To get better values for R_S, we need to reduce the numerator ($n \times A$) and/or increase the denominator of the fraction (I).

The algorithm works with data pieces of the given input data unit size (I), and increasing the input data unit size corresponds to increasing the number (P) or size of the data pieces ($Ps = |s_k|$), which will affect the size of the required storage space A. So, maintaining a trade-off between these highly connected parameters and decreasing the storage ratio is the challenge. Figure 3 shows how these parameters vary and how their values can be regulated to get the best storage ratio. The figure was obtained by measuring the storage ratio by fixing I for three values ($I = 4$, $I = 5$ and $I = 6$) and different values for secret pieces' sizes. The best values per each setting are presented in this figure.

Parameter n is inherently related to P and consequently Ps. On the other hand, both P and Ps are directly connected to the I. As the parameter P is bounded by n, any increase in the I would increase Ps. PRESS relies on matrix calculations

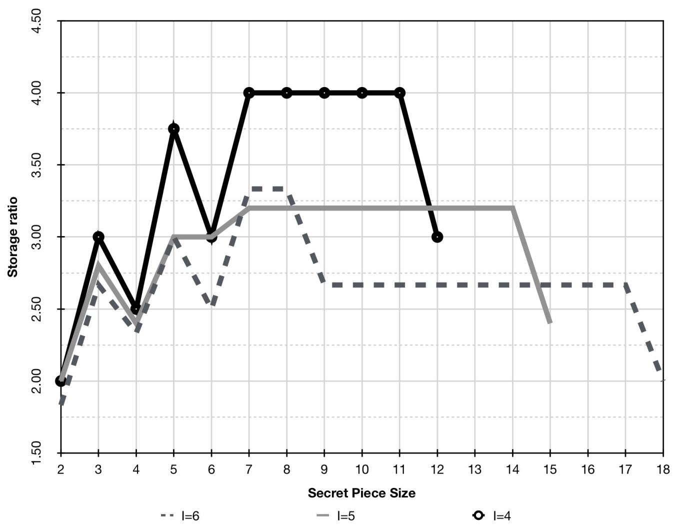

Fig. 3. Storage ratio and the system's parameters

(Section III-B) with entries of size Ps as the input. Thus any increase on Ps would reduce the performance of the system. Therefore, in the different settings for the system parameters with the same results for R_S, the configuration with the smaller size for I is preferred.

Figure 3 shows this relation between the best measured values for R_S (with different settings) and the corresponding parameters of I and Ps. This figure shows the optimum value for R_S is 2 for $I = 4$, but only $R_S = 1.83$ for $I = 6$. Therefore, regarding the slight improvements in R_S for $I = 6$, the first setting (with smaller I) is preferred.

Figure 4 shows the relation between the best measured storage ratio (R_S) and the participants number (n).

With respect to these figures (3 and 4), for a system with 8 clouds ($n = 8$), the optimum storage ratio would be $R_S = 2$ with $I = 4$ and $Ps = 2$. This is the configuration which is used in the evaluation in the next section.

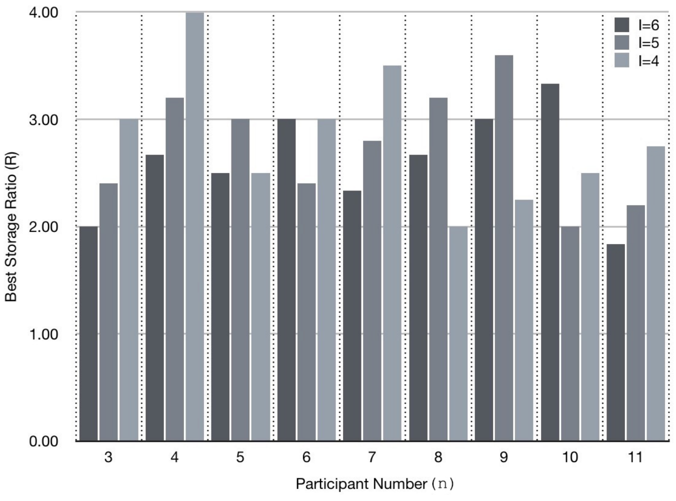

Fig. 4. Storage ratio and the system's parameters

B. SCUD Performance

SCUD is essentially a client side library that uses PRESS as the kernel element to break down files in shares. To assess the system's performance in a real deployment, we initiated it on several clients dispersed in different physical locations. We used a single CSP, Amazon, and the main storage service, Amazon S3, with different regions in 8 locations around the world.

1) System Parameters: Defining the parameters is the initial step in system execution and evaluation. In our case, the relevant parameters are the number of cloud storage servers n, the fault bound f (the maximum number of faulty cloud storage providers), and the secret sharing threshold t. Regulating these parameters may change the storage ratio R_S.

There are different settings used as the system initial by the other systems. DepSky, SCFS, and the other systems based on Byzantine quorum systems focus on the case of $n = 4$ and $f = 1$ as the common deployment setup [5], [6], [16], [50]. To get this setting, there are two options for our system. The first option is to setup the parameters exactly as this ($n = 4, f = 1$). This setting leads to a storage ratio of $R_S = 3$ for PRESS (if remaining parameters are well selected), which is sub-optimal, as seen in the previous section. The second option is to increase the number of shares that each cloud stores and let the virtual number of clouds increase to drop the storage ratio and get better results (regarding the fact that increasing the participants' number may decrease the storage ratio). In this scenario, it is possible to have 4 real clouds, each one storing 3 or more shares. However, such a scenario requires carefully considering the validity of the fault bound (e.g., if a single cloud fault does not violate the fault bound).

In the experiments, we used the Amazon S3 storage service with 8 different regions, i.e., $n = 8$. The fault bound was set to $f = 1$, the threshold number to $t = 7$, and the storage ratio was $R_S = 2$ (Figure 4, with $I = 4$).

2) Methodology: We developed a logger application to measure the latencies. This application tries to repeat several complete writes and read cycles for three different sizes of data units, 100KB, 1MB, and 10 MB, at the client side. To have a baseline for comparison, this application also logs latency of simple upload/download process for these data units at the same clients with 6 different regions of data cloud storage service (see III). Each test of latency in any client consists of 35 rounds of complete read and write operations, but the first 5 rounds are discarded due to the well-known Java warming up issues [56]. To start the evaluation, the clients and CSPs should be appointed.

TABLE III
AMAZON S3 INSTANCES USED FOR SIMPLE FILE OPERATION EXPERIMENTS [57]

Location	Region Name
US	West-1 (N. California)
EU	Central-1 (Frankfurt)
AC	Amazon Center
SA	East-1 (São Paulo)
AP	South East-2 (Sydney) North East-2 (Seoul)

As client terminals, we used four virtual servers on the Amazon EC2 service [58] with the same configurations in 4 different AWS regions: Central Europe (Frankfurt), East US (Virginia), East Asia (Seoul), and South America (São Paulo). The system (PRESS and SCUD) and the logger are installed on all of the clients. Before starting the evaluation, we monitored the client operations and observed that it does not need a considerable amount of resources for executing the system (CPU time, memory space). Therefore, our virtual clients have moderate processing and memory resources, as shown in Table IV.

TABLE IV
VIRTUAL SERVERS' HARDWARE CONFIGURATION AS THE CLIENTS THAT RUN THE SYSTEM.

Model	#vCPU	RAM	Network Bandwitch
Amazon EC2-m5.xlarge	4	16GB	Up to 10Gb

We made two types of measurements. The first is about logging our SCUDs latencies. The system as a cloud-of-clouds selects cloud storage servers and scatters data shares in them (Table V). In this case, the logger just measures the latency between giving the data units to the system and the system's response about the successful upload, and the latency between giving read/download requests to the system and the system's response on successful download. The results of these measurements will be discussed in the next sections.

TABLE V
AMAZON S3 INSTANCES USED IN THE SCUD EXPERIMENTS [57]

Location	Region Name
US	East-1 (N. Virginia) East-2 (Ohio) West-1 (N. California) West-2 (Oregon)
EU	Central-1 (Frankfurt) West-1 (Ireland) West-2 (London)
CA	Central Canada
SA	East-1 (São Paulo)
AP	South East-2 (Sydney) North East-1 (Tokyo) North East-2 (Seoul) South-1 (Mumbai)

The second type of measurement is about logging the latencies of individual cloud uploads/downloads. For this purpose, the logger chooses 5 different data storage servers of Amazon S3 in different regions [57] to store the data units, in addition to one upload/download request without assigning a special server of Amazon, which lets Amazon choose an appropriate server itself (table III). The results of these measurements are shown in Figure 5 covering both the simple upload/download average latencies on the assigned configurations.

3) Writes: As described in the previous section, the logger did the two types of measurements for three sizes of data units. The writing process in our system is based on the algorithms described before (including PRESS and keys and digital signature generations), and all the data shares are

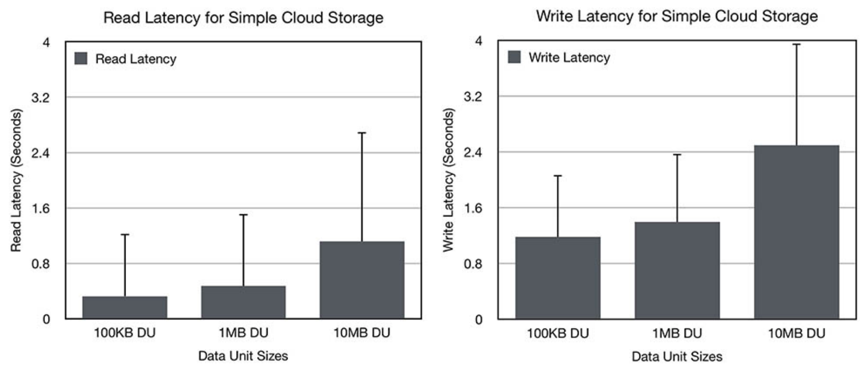

Fig. 5. Latencies of Simple Cloud Storage (write/read)

uploaded concurrently to minimize the upload latency. The confirmation is generated after the successful upload of all the shares and corresponding signatures and meta-data (the algorithm requires writing only to $n - f$ clouds, but we configured the prototype to enforce writing in all clouds). Figure (6) summarizes the measurements, with average and standard deviation for each par client / cloud. To evaluate the efficiency of the system with respect to the different file sizes, its behaviour (in terms of the system latency) toward the change in input file size would be sensible (see Figure 7).

For 100KB data units, the system latency is almost twice the simple data uploads (considering the standard deviations). Increasing the data unit sizes by 10 times to 1 MB data units just makes around 3 times worse the latency of the system. In the case of 10MB files, it is less than 6 times of latency increase for 10 times rise in input data units. It worth ~~to~~ mentioning that our system uses a set of techniques to provide confidentiality and guarantee data integrity, whereas the baseline does not provide any security guarantees besides those provided by S3. Also, the data is uploaded to just one CSP for the single cloud cases, whereas SCUD uploads it to 8 different CSPs, which typically have different access and handling times.

The systems storage ratio for this setting is calculated as 2 (as it is supposed to be), which is better than other systems based on secret sharing that would have 8 (e.g., Belisarius) and equal to DepSky that does not provide information-theoretic security.

4) Reads: The latency measurements for reading operations were done similarly to those of the writes, also using the logger application. Figure (8) summarizes the results of 735 successful data read requests. Similar to what we experienced for write latencies, for 100KB data units, the systems latency is around twice the simple downloads, and increasing the data unit size to 10 times larger just made less than four times rise in the systems latency. For data units of 10MB size, the systems latency was 75 seconds, which is an increase between 7.5 and 10 times in data unit size (See Figure 9).

VIII. Conclusion

Outsourcing data to cloud services is frequently associated with security concerns. Despite all the advantages of multi-clouds, many solutions are based on block ciphers that suffer from uncertainty about their future and their unprovable security. Therefore, information-theoretic security approaches are important. Among different information-theoretic methods, secret sharing schemes can provide security with a high level of confidentiality. They allow sharing files breaking them into different shares by mathematically proved theories. Even though their combination with multi-cloud can provide a high level of security, these schemes –at least those that have been used in previous multi-cloud research– are space inefficient as they multiply the data size n times. In theory, recursive methods could be considered as solutions to resolve this concern. However, these schemes are far beyond practicality.

In this paper, we introduced PRESS, the first practical, efficient secret sharing scheme. This algorithm leverages a recursive approach but makes it practical. With *PRESS*, the storage ratio for secret sharing is reduced to 2 (from n). Furthermore, *SCUD* is introduced as a new cloud-of-clouds storage service that leverages PRESS to provide data confidentiality, integrity, and availability.

The practicality of the proposed architecture and algorithms (PRESS and SCUD) is evaluated extensively, with a world-wide deployment. The key conclusion is that SCUD leveraging PRESS as its kernel provides information theoretic security (confidentiality) for multi-clouds for the first time. This advantage is achieved by leveraging secret sharing schemes (which are considered space-inefficient inherently) at a cost almost double of using a single cloud for a practical scenario. This achievement seems to be a good compromise for critical applications.

To provide the corresponding coefficients, the proposed reconstruction algorithms utilize matrix calculus for polynomial interpolations. Direct interpolation employing matrix calculus and the Gaussian elimination method is efficient and precise and yields exact answers (and not the approximate). On the contrary, it puts an overhead on the system performance. Therefore, it could be a sensible choice to identify different interpolation approaches as *future work*. Also, implementation could be improved by alternative and novel techniques, not only in the data representation and manipulations but also regarding the recursive methodologies efficiency.

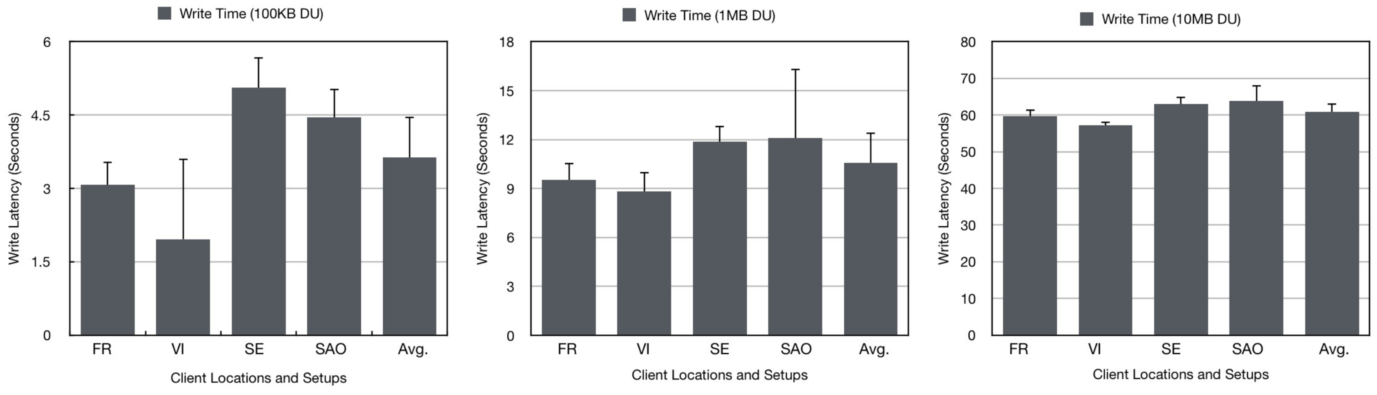

Fig. 6. Write latency for 100KB, 1MB and 10MB data units.

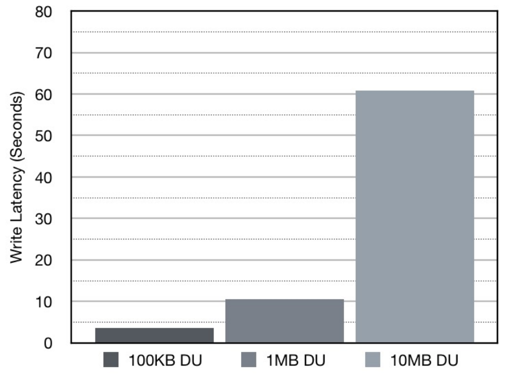

Fig. 7. Average write latency with different file sizes.

Acknowledgment

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 (INESC-ID) and PTDC/EEI-COM/29271/2017 (Cosmos).

References

[1] Cloud Security Alliance, "The notorious nine: Cloud computing top threats in 2013," Feb. 2013.

[2] F. Rocha and M. Correia, "Lucy in the sky without diamonds: Stealing confidential data in the cloud," in *1st International Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments*, 2011.

[3] C. McLellan, "Multicloud: Everything you need to know about the biggest trend in cloud computing," *ZDnet*, Jul. 2019.

[4] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: a case for cloud storage diversity," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 229–240.

[5] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," *ACM Transactions on Storage*, vol. 9, no. 4, pp. 1–33, 2013.

[6] A. Bessani, R. Mendes, T. Oliveira, N. Neves, M. Correia, M. Pasin, and P. Verissimo, "SCFS: a shared cloud-backed file system," *Proceedings of the 2014 USENIX Annual Technical Conference*, pp. 169–180, 2014.

[7] R. Padilha and F. Pedone, "Belisarius: BFT storage with confidentiality," in *Proceedings of the 2011 IEEE International Symposium on Network Computing and Applications*, 2011, pp. 9–16.

[8] D. Dobre and P. Viotti, "Hybris: Robust hybrid cloud storage," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014.

[9] V. Attasena, N. Harbi, and J. Darmont, "fVSS: A new secure and cost-efficient scheme for cloud data warehouses," *Proceedings of the ACM 17th International Workshop on Data Warehousing and OLAP*, pp. 81–90, 2014.

[10] S. Han, H. Shen, T. Kim, A. Krishnamurthy, T. E. Anderson, and D. Wetherall, "Metasync: File synchronization across multiple untrusted storage services." in *USENIX Annual Technical Conference*, 2015, pp. 83–95.

[11] Amazon Web Services Inc, "Amazon S3 introduces cross-region replication," Mar 2015.

[12] D. Burihabwa, R. Pontes, P. Felber, F. Maia, H. Mercier, R. Oliveira, J. Paulo, and V. Schiavoni, "On the cost of safe storage for public clouds: an experimental evaluation," in *Proceedings of the 35th IEEE Symposium on Reliable Distributed Systems*, 2016, pp. 157–166.

[13] S. Pereira, A. Alves, N. Santos, and R. Chaves, "Storekeeper: A security-enhanced cloud storage aggregation service," in *Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on*, 2016, pp. 111–120.

[14] M. Lacoste, M. Miettinen, N. Neves, F. M. Ramos, M. Vukolic, F. Charmet, R. Yaich, K. Oborzynski, G. Vernekar, and P. Sousa, "User-centric security and dependability in the clouds-of-clouds," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 64–75, 2016.

[15] S. Wu, K.-C. Li, B. Mao, and M. Liao, "DAC: improving storage availability with deduplication-assisted cloud-of-clouds," *Future Generation Computer Systems*, vol. 74, pp. 190–198, 2017.

[16] D. R. Matos, M. L. Pardal, G. Carle, and M. Correia, "Rockfs: Cloud-backed file system resilience to client-side attacks," in *Proceedings of ACM Middleware'18*, 2018.

[17] M. Chen and E. Zadok, "Kurma: Secure geo-distributed multi-cloud storage gateways," in *Proceedings of the 12th ACM International Systems and Storage Conference*, 2019, pp. 109–120.

[18] R. Mendes, T. Oliveira, V. Cogo, N. Neves, and A. Bessani, "Charon: A secure cloud-of-clouds system for storing and sharing big data," *IEEE Transactions on Cloud Computing*, May 2019.

[19] Y. Lindell and J. Katz, *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014.

[20] European Union Agency for Network and Information Security, *Algorithms, key size and parameters report - 2014*. ENISA, 2014.

[21] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Physical Review Letters*, vol. 79, no. 2, p. 325, 1997.

[22] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[23] G. R. Blakley, "Safeguarding cryptographic keys," in *Proceedings of the 1979 AFIPS National Computer Conference*, 1979, pp. 313–317.

[24] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual International Cryptology Conference*. Springer, 1991, pp. 129–140.

[25] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Annual International Cryptology Conference*. Springer, 1995, pp. 339–352.

[26] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[27] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti, "Potshards: secure long-term storage without encryption," in *USENIX Annual Technical Conference*, 2008.

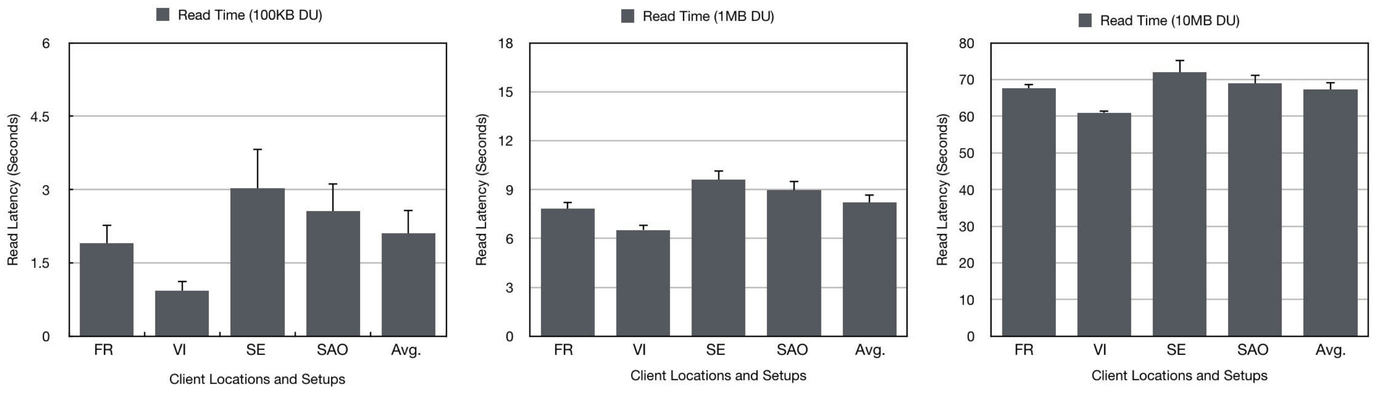

Fig. 8. Read latency for 100KB, 1MB and 10MB data units.

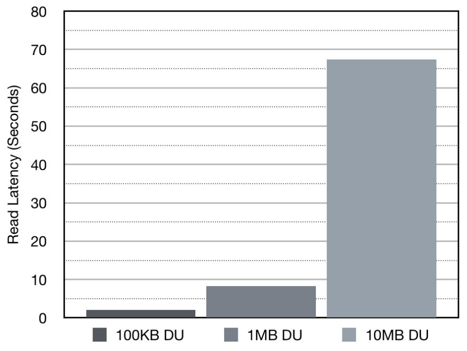

Fig. 9. Average read latency with different file sizes.

[28] C. Blundo, A. De Santis, G. Di Crescenzo, A. G. Gaggia, and U. Vaccaro, "Multi-secret sharing schemes," in *Annual International Cryptology Conference*. Springer, 1994, pp. 150–163.

[29] H.-Y. Chien, J.-K. Jan, and Y.-M. Tseng, "A practical (t, n) multi-secret sharing scheme," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 83, no. 12, pp. 2762–2765, 2000.

[30] C.-C. Yang, T.-Y. Chang, and M.-S. Hwang, "A (t, n) multi-secret sharing scheme," *Applied Mathematics and Computation*, vol. 151, no. 2, pp. 483–490, 2004.

[31] Y.-X. Liu, L. Harn, C.-N. Yang, and Y.-Q. Zhang, "Efficient (n, t, n) secret sharing schemes," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1325–1332, 2012.

[32] A. Parakh and S. Kak, "Space efficient secret sharing for implicit data security," *Information Sciences*, vol. 181, no. 2, pp. 335–341, 2011.

[33] H. Krawczyk, "Secret sharing made short," in *Advances in Cryptology: CRYPTO '93*, D. R. Stinson, Ed., vol. LNCS 773. Springer, 1994, pp. 136–146.

[34] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, 2001, pp. 174–187.

[35] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication." in *Proceedings of 24th USENIX Large Installation System Administration Conference*, 2010.

[36] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security and Privacy*, vol. 8, no. 6, pp. 40–47, 2010.

[37] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Proceedings of the 22nd USENIX Conference on Security*, 2013, pp. 179–194.

[38] M. K. Franklin and M. Yung, "Communication complexity of secure computation (extended abstract)," in *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, Eds. ACM, 1992, pp. 699–710. [Online]. Available: https://doi.org/10.1145/129712.129780

[39] R. M. Capocelli, A. D. Santis, L. Gargano, and U. Vaccaro, "On the size of shares for secret sharing schemes," *J. Cryptol.*, vol. 6, no. 3, pp. 157–167, 1993. [Online]. Available: https://doi.org/10.1007/BF00198463

[40] K. G. Larsen and M. Simkin, "Secret sharing lower bound: Either reconstruction is hard or shares are long," in *International Conference on Security and Cryptography for Networks*, 2020, pp. 566–578.

[41] A. Beimel, "Secret-sharing schemes: A survey," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6639 LNCS, pp. 11–46, 2011.

[42] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, Aug. 1990, pp. 307–315.

[43] Y. Desmedt, "Threshold cryptography," *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 449–457, 1994.

[44] P. S. Gemmell, "An introduction to threshold cryptography," *Criptobytes - The Technical Newsletter of RSA Laboratories*, vol. 2, no. 3, pp. 7–12, 1997.

[45] C.-C. Thien and J. Lin, "Secret image sharing," *Coputers and Graphics*, vol. 26, pp. 765–770, 2002.

[46] A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[47] V. Attasena, J. Darmont, and N. Harbi, "Secret sharing for cloud data security: a survey," *The VLDB Journal*, vol. 26, no. 5, pp. 657–681, 2017.

[48] A. Parakh and S. Kak, "Online data storage using implicit security," *Information Sciences*, vol. 179, no. 19, pp. 3323–3331, 2009.

[49] G. W. Stewart, "Gauss, statistics, and gaussian elimination," *Journal of Computational and Graphical Statistics*, vol. 4, no. 1, pp. 1–11, 1995.

[50] M. Castro and B. Liskov, "Practical Byzantine fault-tolerance and proactive recovery," *ACM Transactions Computer Systems*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[51] W. H. Ware, "Security controls for computer systems (U): Report of defense science board task force on computer security," RAND Corporation, Tech. Rep., Feb. 1970.

[52] G. Brunette, R. Mogull *et al.*, "Security guidance for critical areas of focus in cloud computing v2. 1," *Cloud Security Alliance*, pp. 1–76, 2009.

[53] A. S. Foundation. (2018) Apache jclouds. [Online]. Available: http://jclouds.apache.org/

[54] Oracle. Java platform, standard edition security developers guide. [Online]. Available: https://docs.oracle.com/javase/9/ security/oracleproviders.htm

[55] ——. Java standard edition security platform, developer's guide. [Online]. Available: https://docs.oracle.com/javase/9/security/toc.htm

[56] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous Java performance evaluation," in *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems*, 2007, pp. 57–76.

[57] Amazon S3 regions and endpoints. [Online]. Available: https://docs.aws.amazon.com/general/latest/gr/rande.html

[58] Amazon. Amazon EC2 regions list. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html